CS Exit Ethiopia

Complete Study Notes

Computer Science Exit Examination

16 selected courses | 18 November 2026

Selected courses

01. Computer Programming
02. Fundamental of Database Systems
03. Advanced Database Systems
04. Object Oriented Programming
05. Computer Organization and Architecture
06. Data Communication and Computer Networking
07. Data Structure and Algorithms
08. Web Programming
09. Operating System
10. Software Engineering
11. Design and Analysis of Algorithms
12. Introduction to Artificial Intelligence
13. Computer Security
14. Network and System Administration
15. Automata and Complexity Theory
16. Compiler Design

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Computer Programming	6
Fundamental of Database Systems	22
Advanced Database Systems	44
Object Oriented Programming	68
Computer Organization and Architecture	87
Data Communication and Computer Networking	107
Data Structure and Algorithms	126
Web Programming	148
Operating System	172
Software Engineering	194
Design and Analysis of Algorithms	216
Introduction to Artificial Intelligence	237
Computer Security	258
Network and System Administration	276
Automata and Complexity Theory	298
Compiler Design	318

How to use these study notes

START01 A practical study cycle

Start with a small set of questions before reading.

Mark the exact thing that stopped you: an unknown term, a rule, a calculation, a trace or a choice between alternatives.

Use the question lookup to open the relevant section.

Read one section, close it, and explain the rule in your own words.

Solve the question again and say why each other choice fails.

Then change one assumption or input and solve the variation.

This checks understanding beyond recognition.

A useful 60-minute session: 10 minutes of cold questions, 20 minutes of focused reading, 20 minutes of closed-book solving and 10 minutes updating an error log.

Adjust the timing to your needs.

Revisit mistakes after a gap, for example the next day, several days later and the following week.

These intervals are a practical starting plan, not a fixed learning law.

Record the section ID, your mistaken rule, the corrected rule and one boundary case.

Keep the log short enough to review.

Mix related courses during revision: programming with data structures; databases with security; networking with administration.

Explain how the same word changes meaning across contexts.

START02 What to emphasize

Emphasize definitions, key differences, algorithm costs, SDLC models, databases, networking and security.

Use short oral questions to check understanding and correct common misconceptions.

Combine those habits with the tracing, calculation, design and diagnosis demanded by the Ministry learning outcomes.

Allocate revision by both exam share and your error log.

A course with fewer items can still deserve more time if your foundations are weak.

The blueprint is a planning baseline, not a prediction of exact questions.

Definitions: distinguish nearby concepts, not just their names.

Calculations: write units, formulas, assumptions and intermediate steps.

Code and SQL: check validity before tracing the text; state any changes before analyzing a corrected version.

Algorithms and protocols: know the input, decision rule, state change, termination and failure cases.

Design choices: state the criterion.

A choice can change when cost, reliability, consistency or latency matters more.

Course group	Items in the 100-item blueprint	Revision implication
Web Programming	9	Reserve extra practice for HTML, CSS, JavaScript, PHP, client/server behavior and safe data handling.
Data Structures and Algorithms	7	Trace operations and compare their assumptions and costs.
Each of the other fourteen courses	6 each	Study all sixteen; do not abandon a whole course based on a short note or an alleged repeated-question list.

START03 How to handle exam traps

Underline NOT, FALSE, BEST and EXCEPT before comparing choices.

Check whether the stem supplies enough facts to select one answer.

A long scenario is not automatically an analysis question, and a familiar answer letter is not evidence.

When code is invalid, a diagram is absent, or two choices express the same rule, identify the clarification needed before answering.

Learn the correct rule and the smallest clarification needed.

Do not memorize an arbitrary key as a universal fact.

Distinguish worst case from average case, and an upper bound from a tight bound.

Check language version, SQL dialect, addressing assumptions, graph weights and tie-breaking rules.

For subnetting and scheduling, calculate rather than guess from a familiar number.

Use timed mixed papers only after you can explain your errors.

Revisit the reading sections behind recurring mistakes.

Computer Programming

Complete Study Notes

CP01 | Turn a problem into a program

Identify inputs, outputs, legal ranges and edge cases before coding.

Solve one small case manually, choose an algorithm, express it as pseudocode or a flowchart, implement, test and maintain.

For the largest of three numbers, initialize best to the first input, compare the second and third, then report best.

Initializing best to zero fails when all inputs are negative.

Programming includes writing, testing, debugging and maintaining instructions.

A program can execute without solving its intended problem; compare its actual result with the required result.

Computer Programming module

CP02 | Algorithm properties and paradigms

An algorithm gives precise, executable steps and terminates for every legal input.

Definiteness means each step has an unambiguous meaning; effectiveness means a step can actually be performed.

Inputs may be absent, but a useful algorithm specifies a result.

Correctness means the result meets the specification.

Efficiency is desirable and measurable, but an inefficient terminating algorithm is still an algorithm.

Efficiency is also a desirable property of an algorithm.

Procedural programming organizes operations in procedures; structured programming combines sequence, selection and iteration; object oriented programming organizes state and behavior in objects; functional programming emphasizes function evaluation.

A conditional statement is a construct, not a separate programming paradigm.

Computer Programming module

CP03 | Language levels, generations and portability

Machine instructions are conventionally first generation; symbolic assembly is second; general high-level languages such as C++ and Java are third.

Database query and application generators such as SQL are conventionally fourth generation.

Fifth generation is a loose historical category associated with declarative logic, constraints and AI/expert systems.

Generations do not determine whether an implementation uses an interpreter.

Machine and assembly code depend on an instruction set.

High-level source is easier to read and often portable after recompilation, provided it avoids platform assumptions.

People can write both assembly and high-level programs.

Python, Perl and PHP are commonly called scripting languages; C++ is conventionally compiled.

These labels describe common use, not an absolute technical prohibition.

Computer Programming module

University of Rhode Island: language levels and generations

CP04 | Compilation, linking, loading and Java translation

In the conventional C++ toolchain: source → preprocessing → compilation proper → assembly → object files → linking → executable → loading → execution.

Preprocessing handles directives such as #include; compilation may emit assembly; the assembler emits machine instructions in an object file.

The linker resolves external symbols and combines object files and libraries.

The loader places an executable and required components in memory and starts it.

GCC documents the first four stages explicitly.

A compiler need not emit a complete executable or even native machine code.

Java javac produces bytecode, which a JVM interprets or compiles at runtime.

Interpreters execute a program representation; “one source line at a time” is a beginner approximation.

Compilation uses multiple passes, so “whole program in one step” is not literally universal.

Computer Programming module

GCC manual: compilation stages

CP05 | Identifiers, keywords and comments

In the basic ASCII identifier rules, start with a letter or underscore; continue with letters, digits or underscores.

Do not use a keyword.

C++ is case sensitive: exitExam and ExitExam differ; delete and float are keywords, but FLOAT is a legal identifier. variable_1234 is valid; 4thInput, T-3 and names containing spaces are not single identifiers.

Avoid reserved underscore patterns in application code.

Comments use // to the line end or /* ... */ for a block.

They explain intent and assumptions for maintenance; ordinary comments do not execute or automatically improve speed, reliability or portability. main is the conventional entry function name, not a C++ keyword.

Computer Programming module

CP06 | Declarations, initialization, constants and types

Write type name; or type name = value; for ordinary variables: int age = 20;.

Declaration introduces a name and type; initialization gives its initial value; later assignment replaces a value. a=b copies a scalar value, so later changes to b do not change a. A pointer or Java object reference instead copies an address/reference.

Use int for whole numbers, float/double for fractional approximations, char for a character and bool for logical values in C++.

Type sizes depend on the implementation; do not import Java widths into C++. short n, short int n and int short n denote the same signed short type. const int n=5 and int const n=5 are equivalent; an ordinary const int needs initialization. typedef creates an alias, not a distinct new type.

Computer Programming module

CP07 | Literals, conversions and sizeof

Single quotes denote a character literal, such as 'A'; double quotes denote a string literal, such as "A".

A string includes more storage than just its visible characters when a terminator is required. sizeof reports size in bytes, with sizeof(char)==1; the byte width and other sizes are implementation dependent. sizeof(array) gives total array storage before conversion to a pointer.

With int x=5,y=2, x/y is 2.

Assigning that quotient to double gives 2.0, not 2.5. static_cast<double>(x)/y performs floating division and yields 2.5.

Converting an in-range fractional floating value to int truncates toward zero.

Cast before the operation if a fractional result is needed.

Computer Programming module

CP08 | Arithmetic and compound updates

Multiplication, division and remainder have higher precedence than addition and subtraction; equal-precedence arithmetic groups left to right.

Integer division truncates toward zero.

For integers, a == (a/b)*b + a%b when the division is valid. % requires integral operands in C++; (8.0*3)%5 is invalid.

Division by zero and signed integer overflow cannot be given ordinary numeric outputs.

x+=y, x-=y, x*=y, x/=y and x%=y update x using the operation and conversion to its type.

For simple variables, trace them as x=x op y. Starting p=q=n=8, p+=n gives 16 and p/=q gives 2.

Starting x=10, x+=5 then x/=2 gives 7. x=x+x doubles; it does not generally add one.

Computer Programming module

CP09 | Prefix, postfix and evaluation safety

Prefix ++x changes x and yields the new value; postfix x++ yields the previous value and also increments x. Separate the value used in the expression from the stored value after it.

Starting a=8,b=5, a++ + --b uses 8+4, gives 12, and leaves a=9,b=4.

Starting x=1,y=10,z=-3, (x++*y--)+ ++z gives 8 and leaves x=2,y=9,z=-2.

Precedence determines grouping, not a universal evaluation order.

Avoid expressions that modify a variable multiple times without sequencing.

In C++17 and later, chained operator-form stream insertions are sequenced left to right: with x=5, cout<<x++<<" "<<++x prints 5 7.

Before C++17 this expression has undefined behavior.

For the C++17 expression above, 5 6, 5 5, 6 5 and 6 6 are all incorrect outputs.

Computer Programming module

C++ draft: shift sequencing

CP10 | Boolean, relational, bitwise and short-circuit logic

Relational and equality operators test values; && and || combine truth values; ! negates.

In C++, zero converts to false and a nonzero number to true.

Streams display bool as 0 or 1 unless boolalpha is set.

With a=3,b=5, cout<<(a==b) prints 0.

&& evaluates its right operand only if the left is true; || only if the left is false.

Thus false && dangerousCall() skips the call.

Bitwise & operates on bits and does not short circuit.

With x=1,y=0,z=4, (x&y)||++z evaluates ++z, leaving z=5 and the Boolean result 1.

The precedence ladder is grouping/postfix, unary, multiplicative, additive, shifts, relations, equality, bitwise &, ^, |, &&, ||, conditional, assignment, comma.

Computer Programming module

CP11 | Conditional and comma expressions

condition ? yes : no evaluates the condition and exactly one alternative.

The comma operator evaluates its left expression, then its right, and yields the right value.

Do not confuse it with commas separating function arguments.

For a=6,b=8,x=2,y=4, c=(x>y ? (a--,x) : (b--,y)) takes the false branch. b becomes 7, c receives 4 and a remains 6.

Both alternatives are not executed.

A semicolon in place of the internal comma is invalid syntax; a trace using a comma applies only after that syntax change.

Computer Programming module

CP12 | Trace decisions and recognize flowcharts

A standalone if may execute independently of the following if.

An else binds to its nearest unmatched if; an else-if ladder executes the first true branch only.

In nested if(c1){if(c2) S1; else S2;}, S2 requires c1 true and c2 false.

Braces determine grouping.

With ph=15.3, an invalid-range if can print its warning, and a separate classification ladder can also print Alkaline.

They are not mutually exclusive unless linked by else or an early return.

A flowchart decision is a diamond, processing a rectangle, input/output a parallelogram and start/end a terminator.

A pre-test loop checks before its body; a post-test loop checks after.

Identify a flowchart’s loop from its arrows and the placement of its conditions.

Computer Programming module

CP13 | Switch selection and fall-through

A traditional C++ switch selects using an integral or enumeration value. case labels are constant expressions.

Execution starts at the matching label, then continues through subsequent statements until a break, return or the end. default handles an unmatched value but need not be the last physical label.

If case 2 prints Bus. and case 3 prints Truck. before the first break, input 2 prints Bus.Truck.

A break exits the nearest switch or loop; it does not by itself terminate the entire program.

A case label is an entry point, not an automatically isolated if branch.

Computer Programming module

CP14 | While, do-while and for mechanics

while(condition) checks before each body execution and may run zero times. do {body;} while(condition); runs the body once before checking; it repeats while the condition remains true.

Use it when a menu or input attempt must occur at least once.

for(initialization; condition; update) executes initialization once, then condition → body → update → condition.

A missing condition in for(;;) acts as true.

A for loop is convenient for a known count but can also be condition controlled.

To print indices 0 through n-1, use i=0; i<n; ++i. Starting i=0 with i<5 and i+=2 prints 0 2 4.

Computer Programming module

CP15 | Break, continue, return and goto

break exits the nearest enclosing loop or switch. continue skips the rest of the current loop iteration: a for loop then runs its update; a while loop goes directly to its condition. return leaves the current function and may supply a result.

In main it ends the program normally.

A goto transfers to a label within its function; follow the label reached rather than reading lines in source order.

In the sequence a: print Graduating; goto b; c: print Year; goto a; b: print fourth;, entry at a prints Graduating fourth and reaches the end.

Year is not printed.

Structured alternatives usually make such logic easier to reason about.

Computer Programming module

CP16 | Count iterations and diagnose nontermination

Make a table with condition inputs, body output and updated values.

For i=0,j=8; i<j; i++,j--, body entries are (0,8),(1,7),(2,6),(3,5).

If odd i triggers continue, printing occurs twice.

With outer i=1; i<5; i+=5, the outer loop runs once; an inner j=10; j>i; --j prints nine times.

For n=1; n<=18; n+=2, print n unless n%7==0 then break: output is 1 3 5.

A while(a>b) with fixed a=6,b=5 never changes its true condition.

In data[10]={5,10,...,50}, i=0, advancing i only if data[i]%10==0 causes no progress because data[0]=5.

It prints nothing and loops indefinitely; changing <=10 to <10 alone does not repair progress.

Computer Programming module

CP17 | Functions, parameters and reusable modules

A function groups a named task.

C++ definition: int add(int a,int b){return a+b;}.

Call it with add(x,y);, without parameter type declarations at the call site.

Parameters are names in the definition; arguments are supplied expressions or values.

A void function produces no return value but can have side effects and return; to exit early.

A method is a function associated with a class or object; methods can return values.

Modules group related responsibilities behind interfaces.

Reuse reduces duplication; independent functions are easier to understand and test.

A library provides reusable components; a framework also supplies an organizing structure and often calls application callbacks.

Computer Programming module

CP18 | Value, reference and pointer parameters

A C++ value parameter is a separate object initialized from the argument.

In func(int a,int b){a*=5;b+=5;return a+b;}, calling func(9,3) returns 53 and leaves the caller values 9 and 3 unchanged.

An int& parameter aliases the caller variable, so swap(int& a,int& b) changes both originals.

An int* parameter copies an address; *p=value changes the pointed-to object, but p=anotherAddress only changes the local pointer.

An array parameter int a[] is adjusted to int*: elements can change, and the length must be passed separately.

Java always passes by value, including a copy of an object reference; it does not pass caller variables by reference.

Computer Programming module

CP19 | Overloads, default arguments and inline

Overloads share a name but differ in parameter count or types/order.

Return type alone cannot distinguish them.

Choose a candidate using argument types, then required conversions; report ambiguity if there is no unique best match. int sum(int x,int y,int z=0) allows two or three arguments; defaults occupy trailing positions in ordinary declarations.

inline does not compel the compiler to copy a function body at a call.

Small frequently used functions may be good optimization candidates, but actual expansion is the optimizer's decision.

Inline also has language linkage/definition rules, so it is not simply a speed command.

Computer Programming module

CP20 | Scope, lifetime and persistent static locals

Scope is where a name can be used; lifetime is when its object exists.

A local name is restricted to its block; an outer name may be shadowed.

A namespace/global variable is available where its declaration is visible, not magically in every file.

C++ ::name selects a global name hidden by a local one.

An automatic local is created anew on entry.

A static local is initialized once and retains its value across calls even though its name remains local.

If static x=0 changes by +1 when even and +2 otherwise, five calls return 1,3,5,7,9, whose sum is 25.

Namespace/static storage integers are zero initialized; an ordinary uninitialized local int has no usable assumed zero value.

Computer Programming module

CP21 | Recursion, base cases and call tracing

A recursive function calls itself directly or indirectly.

It needs a base case and progress toward it for all supported inputs. factorial(0)=1 and factorial(n)=n*factorial(n-1) for positive n. factorial(3) expands to 3*2*1*1=6.

Negative inputs require rejection or a separate definition; they do not reach zero under n-1.

Evaluate conditions before expanding a recursive call.

For if(n<m) return 0;, f(1,4) returns immediately; unreachable recursive branches add nothing.

Record each call’s local parameters and pending operation, then unwind returned values.

Recursion commonly uses smaller instances, but merely having a base-case branch does not prove it will be reached.

Computer Programming module

CP22 | Arrays and bounds

A built-in C++ array has a fixed number of same-type elements in contiguous storage: int a[4]={10,20,30,40};.

Valid indices are 0 through 3. a[4] is outside the array; accessing it has undefined behavior.

Java checks bounds and throws ArrayIndexOutOfBoundsException.

Do not predict a C++ out-of-range value from nearby memory.

A standard built-in array bound must be a compile-time constant; variable-length local arrays are compiler extensions. int a[5]={1,2,3}; fills the remaining elements with zeros.

An uninitialized automatic array of int is different. a[i] equals *(a+i), so elements can be accessed using pointer notation without explicitly writing brackets.

Computer Programming module

CP23 | Array algorithms and two-dimensional indexing

For a sum, start total=0 and add each element once.

For a maximum/minimum, initialize from a[0] only after confirming the array is nonempty.

For 50 inputs where nonpositive values are skipped, use a count-controlled loop and either if(x>0) total+=x; or continue for x<=0.

These constructs solve different parts of the task.

int a[R][C] has R rows, each with C elements. a[r][c] uses zero-based row then column.

Its element offset is r*C+c in row-major storage.

In the 4x4 matrix example, a[1][1]=6, a[2][3]=6 and a[3][3]=2.

In Java, length belongs to each array; a traversal ending at arr.length-2 visits fewer elements than a full traversal.

Computer Programming module

CP24 | C strings and C++ string objects

A C string is a character array ending in '\0'. "Hello" needs six char positions; strlen counts five visible characters and excludes the terminator. sizeof a fixed char array reports its allocated bytes, which can exceed strlen+1. cout<<a prints a terminated char sequence, not its numeric address.

<cstring> supplies strcpy(destination,source), strcat(destination,source), strlen and strcmp.

Supply enough destination capacity; strcmp returns zero for equal content, otherwise a sign, not necessarily exactly -1 or 1.

C++ std::string in <string> owns resizable text; + concatenates, == compares content, size()/length() counts stored characters and substr(start,count) extracts a segment.

Standard std::string has no Java-style toUpperCase() or trim() member.

Computer Programming module

CP25 | Pointers, dereferencing and assignment

int x=10; int* p=&x; stores x's address in p. *p reads or writes x; &p is the address of the pointer variable itself.

With p=&a,q=&b, p=q redirects p to b and leaves a and b unchanged.

By contrast *p=*q copies the pointed-to value.

Never dereference an uninitialized, null or dangling pointer.

An array expression commonly converts to a pointer to its first element, but the array is not itself an assignable pointer variable.

A pointer-to-pointer int **pp=&p allows** pp to reach x. A void* can hold an object address but requires conversion to a suitable object pointer before dereferencing in C++.

Computer Programming module

CP26 | Scaled pointer arithmetic and dynamic allocation

If p points inside an array, p+k advances k elements, or k*sizeof(*p) bytes.

Under the explicit assumption sizeof(float)==4 and first address 7000, p+3 has numeric address 7012. cout<<p++ displays the old address then advances p; cout<<*p++ displays the old element then advances p. ++*p changes the element.

Pointer addition p+q is invalid; subtracting pointers within the same array gives an element distance.

Arithmetic must stay within an array or its one-past position; one-past may not be dereferenced.

C++ new T creates a dynamic object, delete releases it; new T[n] pairs with delete[].

Prefer resource-owning standard containers in real programs.

Allocation does not automatically initialize every primitive value unless initialization is requested.

Computer Programming module

CP27 | Structures and member access

A struct groups named members, possibly of different types. struct House{double price; int rooms;}; defines a type; House h{150000,3}; creates an object. h.rooms uses a dot.

House* p=&h; p->rooms is equivalent to (*p).rooms.

The pointer must refer to a live object before member access.

A type definition alone does not create an ordinary instance.

Aggregate initialization assigns members in declaration order.

Modern C++ permits default member initializers such as int rooms=3;.

C++11 list initialization rejects narrowing; a constant floating value exactly representable in the target floating type can be permitted, so 25.00 to float is not necessarily narrowing.

Use 25.0f for clarity and check the language version.

Structures can be copied, passed and returned; a structure cannot contain a full instance of itself, but may contain a pointer to its own type.

Computer Programming module

CP28 | Console input, output and headers

Include <iostream> and qualify std::cin/std::cout or introduce the names explicitly. cout is the standard output stream; << inserts values. cin is standard input; >> extracts formatted values. std::endl inserts a newline and flushes.

C++ names are case sensitive: Cout, Int and std: :cout are not equivalent valid spellings of cout, int and std::cout.

Read a full line with getline(cin,text).

After formatted numeric extraction, consume the pending newline before a line read when required.

Input failure must be tested before trusting a value.

Console output formatting matters: by default a double storing an exact 4 may print 4 rather than 4.0.

Text, spaces and newlines are part of an output trace.

Computer Programming module

CP29 | File streams, modes and safe reading

Include <fstream>. ofstream writes, ifstream reads, and fstream can do both.

The constructor's default ofstream mode argument is ios::out; its ordinary effect is to create/truncate the file.

Distinguish the default mode argument from effective out|trunc behavior. ios::app places each write at the end; ios::ate initially seeks to the end; ios::binary requests binary mode.

Check if(!file) after opening.

Read using while(file>>record) or while(getline(file,line)), because end-of-file is discovered by an attempted read. while(!file.eof()) can process an invalid extra record.

Close or rely on stream destruction to release the resource. "ofstream" is a class, whereas <fstream> is the header.

Computer Programming module

C++ draft: ofstream constructors

CP30 | Mathematical functions and iterative powers

Use <cmath> for sqrt, pow, floor and ceil; integer abs is also provided by <cstdlib>. sqrt(16)=4; pow(3,4)=81; abs(-7)=7. ceil(x) is the least integer-valued result >=x; floor(x) the greatest <=x. Thus ceil(-2.7)=-2 and floor(-2.7)=-3; truncation toward zero differs from floor for negatives.

C++ has no built-in exponentiation operator: ^ is bitwise XOR and ** is not a power operator.

To compute b^e for a nonnegative integer e, start r=1 and repeat r*=b exactly e times; b=4,e=3 gives 64.

Large powers and factorials can overflow fixed-width types; specify a valid input range.

Computer Programming module

CP31 | Correctness by logic, invariants and induction

To prove an implication P⇒Q, a direct proof derives Q assuming P; a contrapositive proves not Q⇒not P; a contradiction assumes the claim false and derives an impossibility.

A single counterexample disproves a universal claim.

Induction proves a base case and proves P(k+1) assuming P(k).

For sum=0,i=1; while(i<=n){sum+=i;++i;}, use invariant sum=(i-1)*i/2.

It holds initially.

Adding i gives i*(i+1)/2, then incrementing i restores the invariant.

Exit gives i=n+1 and sum=n*(n+1)/2.

Termination needs a separate decreasing nonnegative measure n-i+1 while the guard holds.

Assume n>=0 and no arithmetic overflow.

This connects mathematical proof to actual code.

Western University: program verification

Computer Programming module

CP32 | Debugging, errors, exceptions and maintenance

A syntax error violates grammar; a type error can be syntactically well formed but rejected during semantic checking.

A runtime error occurs during execution; a logic error gives the wrong behavior.

A bug is a defect; debugging locates and corrects it.

Read the first useful compiler diagnostic, inspect nearby tokens and declarations, repair and rebuild; a debugger is useful after a runnable program exists.

Test ordinary, boundary and invalid inputs and trace the first point where actual state differs from expected state.

Exceptions transfer control to a matching handler for exceptional conditions; they do not make arbitrary undefined behavior recoverable.

Refactoring improves internal structure while preserving observable behavior.

Version control records changes, supports comparison and rollback; documentation explains use, design and maintenance assumptions.

Computer Programming module

CP33 | Static typing and Java-specific details

Static typing checks types before execution; type inference can still infer a static type, as with C++ auto.

Dynamic typing checks operations using runtime values; it does not mean variables must have explicit static declarations.

Static typing does not always require explicit type annotations.

Java uses boolean, fixed-width numeric primitive types and references to objects.

Java String is immutable, substring(begin,end) uses an exclusive end, equals compares content and == compares references.

Java arrays have a length field.

Scanner nextInt() can leave a newline for nextLine().

C++ std::string is mutable and uses substr(start,count); the two APIs must not be merged.

Computer Programming module

CP34 | Check syntax and trace control flow

A program trace requires the complete code, and a diagram question requires its edges, conditions and labels.

When entry reaches the body rectangle before the decision diamond and a decision branch returns to the body, the body executes before the condition is tested.

This is a post-test do-while loop.

Check that a program is valid before tracing it; state any syntax changes before analyzing the resulting program.

Match pointer-member names exactly: Cout/Cin, C1 and car_mode do not match cout/cin, c1 and car_model.

In the goto sequence above, Year is not printed because control does not reach its label.

With valid C++ syntax for the stated assignments, the trace yields z=5 then a=1.

Printing Z when only z is declared refers to a different, undeclared identifier.

Use <0 for a less-than-zero comparison and the standard int main() declaration for a C++ entry point.

Check int and cout case, quotation marks and comma-expression punctuation before tracing a C++ expression.

Predict output only after the program and its execution conditions are clear.

Computer Programming module

CP35 | Interpret programming questions precisely

Evaluate every choice against the language rules; more than one statement may be valid, or none may be valid.

Distinguish linking from loading.

Distinguish binary representation, machine code and executable files.

Assess class membership and possible initialization separately; both statements can be correct.

Check whether alternative short declarations are equivalent.

Check the C++ version and narrowing rules before evaluating an initialization.

Array subscripting and equivalent pointer access can refer to the same element.

Use the definition of static typing rather than equating it with explicit annotations.

An ordinary integer declaration can be written int x;.

A void function definition can be complete without returning a value; int myFunction(){} can compile, but reaching its end is undefined behavior.

A break statement exits the enclosing loop or switch.

Check required initialization when evaluating const declarations, including alternative const spellings.

Distinguish counting iterations from skipping inputs, and terminating a loop from ending only the current iteration.

After inspecting and changing code, rebuild and retest to complete debugging.

Computer Programming module

CP-S01 | Empty braces initialize aggregate members

For an aggregate with no default member initializers, empty braces initialize its members recursively. Numeric scalar members become zero, and the elements of a character-array member become zero characters. A zero character at the beginning of such an array makes it an empty, terminated C string. This differs from leaving an ordinary automatic scalar or array uninitialized.

C++ working draft - Aggregate initialization

CP-S02 | High-level languages and scripting usage

Python is a high-level programming language with built-in high-level data types. Calling a language a scripting language describes a common way it is used; it does not make that language machine code or assembly. Language level and execution style are different classifications.

Python tutorial - Whetting Your Appetite

CP-S03 | Keywords and names have different roles

A keyword is a reserved word with a predefined meaning in the language grammar. It cannot be repurposed as an ordinary variable or function name. An identifier is a name used to refer to a data object or function; its spelling must obey the identifier-character rules and avoid keywords.

Computer Programming module · page 5

CP-S04 | Java print and println preserve output layout

Java System.out.print writes the supplied value without automatically ending the line. System.out.println writes its value and then a line separator. Successive println calls therefore place their values on separate lines, even when each printed expression also contains a trailing space.

Java SE 17 - PrintStream print and println

CP-S05 | An assignment expression can accidentally become a condition

In C++, simple assignment changes its left-hand object, and the assignment expression refers to that updated object. When an ordinary numeric assignment is used as a condition, the newly assigned value is converted to truth: zero is false and nonzero is true. Equality testing with == instead produces a Boolean result without assigning either operand. Confusing the two can both change program state and choose an unintended branch.

C++ working draft - Assignment and compound assignment operators

Fundamental of Database Systems

Complete Study Notes

FDB01 | Database approach, benefits and costs

A database is a coherent collection of related facts with meaning for an application.

A DBMS is software that defines, stores, retrieves, updates, protects and shares that data.

The database is the content; the DBMS manages it.

A database system also includes applications, people and infrastructure.

Traditional file applications define their own formats and access procedures.

Separate files can repeat facts, disagree after updates, isolate useful data and make shared integrity or concurrent updates difficult.

A DBMS centralizes metadata and supports multiple views, authorization, transactions, backup and recovery.

It controls redundancy rather than guaranteeing zero duplication.

Complexity, administration, storage and licensing can outweigh benefits for a tiny fixed application; lightweight embedded DBMSs also exist.

Concurrent access itself is useful; uncontrolled concurrent access causes anomalies.

The DBA manages access authorization, backup/recovery, availability, monitoring, tuning and routine maintenance, coordinating database use; designers define the schema and application developers build user-facing programs.

Fundamental of Database Systems module

Advanced Database Systems module

FDB02 | Data models and product names

A data model supplies concepts for structure, relationships, constraints and operations.

Conceptual ER models describe entities and relationships.

Relational models use tables; hierarchical models organize records in a parent-child tree; classic network models connect records through pointer-based owner/member links, allowing multiple parents.

Object models represent objects with identity, state and behavior.

A physical model specifies record layouts and access paths.

These are different representation choices, not stages of one query.

PostgreSQL, MySQL, Oracle Database, Microsoft SQL Server and IBM Db2 are relational DBMS products.

MongoDB is a document DBMS.

Java is a programming language.

IBM alone names a vendor, not a database product.

Distinguish a vendor such as IBM from a specific database product.

Fundamental of Database Systems module

Advanced Database Systems module

FDB03 | Relations, schemas, instances and metadata

A relation is a set of tuples, represented as a table.

A tuple/row/record contains related attribute values; an attribute/column/field describes a property.

Degree counts columns; cardinality counts rows.

Classical relations have distinct tuples, atomic cells and domains for attributes; row order has no meaning.

Column order has no logical meaning when attributes are identified by name.

SQL tables can contain duplicate rows unless constrained.

A schema describes names, structures and constraints; an instance, state or snapshot is the data at a particular time.

Inserting a student changes the state, while adding a column changes the schema.

Metadata is data describing data.

The catalog or data dictionary records table definitions, types, constraints and often access privileges and optimizer statistics.

A schema diagram draws tables and key references; labels such as database diagram can overlap.

Fundamental of Database Systems module

Advanced Database Systems module

FDB04 | Three-schema architecture and data independence

The external level gives each user group a selected view.

The conceptual level describes the whole logical database: entity types, relationships, data types and constraints.

The internal/physical level describes files, record placement and access paths.

These are abstraction levels, distinct from client/server deployment tiers.

Physical data independence permits changing internal storage without changing the conceptual schema, for example adding an index.

Logical data independence permits conceptual changes while preserving affected external views and application interfaces, for example splitting a table while maintaining a compatible view.

It is harder because applications depend on logical meaning.

Data abstraction hides irrelevant details; program-data insulation separates data definitions from application code.

Fundamental of Database Systems module

Advanced Database Systems module

FDB05 | Requirements through implementation

First collect business facts, required transactions, reports, volumes, privacy rules and quality targets.

Conceptual design turns them into an ER/EER model.

Logical design maps the conceptual model to relations, keys and constraints and checks normalization.

Physical design chooses file organization, indexes and storage.

Implementation creates and loads the actual objects; testing checks requirements.

Logical relational design is tied to a data model but can remain independent of a particular DBMS.

Vendor-specific implementation details may be introduced during later refinement.

Whether logical design is DBMS-dependent depends on the stated design methodology.

Conceptual design usually proceeds from a whole enterprise view downward; normalization can refine structures from dependencies upward.

These are useful approaches rather than mandatory directions for every modeling activity.

Fundamental of Database Systems module

Advanced Database Systems module

FDB06 | Entities, attributes and ER notation

An entity is a distinguishable real or conceptual thing, such as one student or one course.

An entity type describes common properties; its current collection of entities is an entity set.

A relationship associates entities.

In Chen notation rectangles represent entity types, diamonds relationships and ellipses attributes; double ellipses denote multivalued attributes and dashed ellipses derived attributes.

Always inspect the diagram legend.

Simple versus composite asks whether a value has meaningful components: Address may split into City and Street.

Single-valued versus multivalued asks how many values one entity may have: a student may have several phone numbers.

These dimensions differ: one address may be composite and single-valued.

Age derived from birth date is derived, even if also single-valued.

Attribute names alone do not establish multiplicity.

Fundamental of Database Systems module

Advanced Database Systems module

FDB07 | Superkeys, candidate keys and primary keys

A superkey uniquely identifies a tuple.

A candidate key is a minimal superkey: removing any component loses uniqueness.

Minimal means no unnecessary attribute, not necessarily the fewest columns among all keys.

A primary key is one selected candidate key; remaining candidates are alternate keys.

A composite key contains multiple attributes and need not be the chosen primary key.

If ID is unique, both {ID} and {ID,City} are superkeys, but only {ID} is minimal.

A table can have several candidates and foreign keys but at most one declared primary-key constraint, possibly containing several columns.

String columns can be keys.

Prefer stable, compact identifiers when practical, but Email, SSN or EmployeeID is a candidate only if business rules guarantee uniqueness.

A composite-key example must include multiple columns; a single-column key is not composite.

Example

Worked example

- Given candidate keys CustID, Email and ContactNo, all are eligible primary keys. CustID is preferable only with an explicit stability/size policy; uniqueness alone does not choose it.

Fundamental of Database Systems module

Advanced Database Systems module

FDB08 | Weak entities and identifying relationships

A weak entity lacks a complete independent key.

Its owner is normally a strong entity, so owner entity set and strong entity set may both describe the associated set.

Its partial key, also called discriminator, distinguishes weak entities of the same owner.

Identification combines the owner key and partial key.

For Dependent(EmployeeID,DependentName,BirthDate), suppose names are unique only within one employee's dependents.

Then (EmployeeID,DependentName) is the key, and EmployeeID is a foreign key.

The weak entity participates totally in its identifying relationship.

Chen notation uses a double rectangle, double identifying diamond and dashed underline for a partial key.

A weak entity is not merely any table with a foreign key.

Fundamental of Database Systems module

Advanced Database Systems module

FDB09 | Cardinality, participation and direction

Cardinality specifies maximum associated occurrences: 1:1, 1:N or M:N. Minimum cardinality specifies optional/partial participation (zero permitted) or mandatory/total participation (at least one).

A maximum of one does not mean participation is compulsory.

Relationship degree counts participating entity roles: unary recursive, binary or ternary.

Read business rules in both directions.

If each employee belongs to one department and each department has many employees, Employee→Department is N:1; Department→Employee is 1:N. Building→Location could also be N:1, so names alone cannot select a unique example.

Under the common arrow convention, arrows from advisor to both Instructor and Student impose maximum-one at both ends, giving 1:1.

Without a notation convention, arrows are unsafe evidence.

Fundamental of Database Systems module

Advanced Database Systems module

FDB10 | Map entities, composite and multivalued attributes

Create one relation per strong entity type, preserve its simple attributes and select a candidate key.

Expand composite attributes into their useful simple components; do not copy a composite ER structure unchanged into a flat atomic design.

Derived values can usually be calculated, avoiding stale stored copies.

Map each multivalued attribute to a separate relation containing the owner key and one value.

StudentPhone(StudentID,Phone) has primary key (StudentID,Phone) and a foreign key to Student.

Store separate rows for two phone numbers, not a comma-separated list or repeated Phone1/Phone2 columns.

Include the relational mapping step when translating an ER/EER design into tables.

Table names may follow entity names; naming does not invalidate a mapping.

Fundamental of Database Systems module

Advanced Database Systems module

University of Texas: ER-to-relational mapping

FDB11 | Map 1:1 and 1:N relationships

For 1:N, place the one-side key as a foreign key in the many-side table; store relationship attributes there.

Department(DeptID) and Employee(EmpID,DeptID,HireDate) implement many employees per department.

If participation is total, make the reference NOT NULL.

If partial, a nullable reference can represent no assignment.

For 1:1, one side may hold the other key with a UNIQUE constraint.

Prefer the totally participating side to reduce NULLs; NOT NULL enforces that side's mandatory reference.

Merging entities is reasonable only when their lifecycle and participation rules permit it.

For A→B many-to-one, A can absorb the relationship reference under either total or partial participation; requiring total participation as the only possible case overstates the rule.

Fundamental of Database Systems module

Advanced Database Systems module

FDB12 | Map M:N, recursive and n-ary relationships

A binary M:N relationship needs an association table in addition to the two entity tables.

Enrollment(StudentID,CourseID,Grade) references Student and Course; (StudentID,CourseID) is a key if one enrollment per pair is allowed.

Repeated attempts require Term or AttemptNo in the key.

This ordinarily produces three tables.

A recursive M:N friendship uses UserLink(UserID,OtherUserID), two foreign keys to User; decide whether links are directed and whether self-links are allowed.

A recursive 1:N supervision uses a nullable SupervisorID reference in Employee.

For a ternary Supply(SupplierID,PartID,ProjectID,Quantity), preserve all participating references; binary replacements can lose which triple occurred.

Choose the association key from actual cardinalities rather than automatically assuming every reference is necessary.

Fundamental of Database Systems module

Advanced Database Systems module

University of Texas: ER-to-relational mapping

FDB13 | EER inheritance and category constraints

EER extends ER with superclass/subclass structure.

A subclass inherits the superclass attributes and relationships.

Specialization divides Student into Undergraduate and Graduate; generalization combines common properties into Student.

Disjoint subclasses cannot overlap; overlapping subclasses can.

Total specialization requires every superclass member in some subclass; partial specialization does not.

Map Person(PersonID,Name) and Student(PersonID,Program) with Student.PersonID both primary key and foreign key to Person.

This preserves inherited identity.

A category/union subclass draws members from a union of different superclasses, such as Owner drawn from Person or Company; it differs from an intersection subclass requiring membership in both.

Object models also support inheritance; distinguish basic ER notation from EER when comparing models.

Structural EER inheritance alone does not define executable object methods.

Fundamental of Database Systems module

Advanced Database Systems module

FDB14 | Fan traps and chasm traps

A fan trap occurs when two 1:N relationships fan from a common entity and a path suggests an association it cannot determine.

Department→Employee and Department→Project do not reveal which employee works on which project; joining through Department pairs every employee with every project in that department.

Record WorksOn explicitly when that is the fact needed.

A chasm trap occurs when an apparently useful path fails for some occurrences because an optional relationship breaks it.

For example, a branch's property may be lost from a path that requires an assigned staff member when some properties have none.

Add the required direct relationship or revise the model.

Multiple 1:N relationships are legitimate; eliminating all fans is not a universal repair.

A missing path describes a chasm, not the defining ambiguity of a fan.

Fundamental of Database Systems module

Advanced Database Systems module

Dr G. Russell, Napier University: Database Systems notes

FDB15 | Integrity and constraint taxonomy

Domain integrity restricts type and allowed values.

Key integrity enforces uniqueness.

Entity integrity requires every primary-key component to be non-NULL.

Referential integrity requires a non-NULL foreign-key value to match an eligible referenced unique/candidate key; self-references are allowed.

A parent need not have a child.

Inherent constraints come from the data model; schema-based constraints are declared through the DDL; semantic/application rules may need triggers or application logic when DDL cannot express them.

Integrity improves data validity; authorization controls who may access it.

Uniqueness and domain validation are both integrity mechanisms.

If a Department primary key is NULL and Manager=107 lacks a referenced employee, entity integrity certainly fails and referential integrity also fails if that foreign key is declared.

One INSERT can violate multiple constraints.

Fundamental of Database Systems module

Advanced Database Systems module

FDB16 | Write and enforce SQL constraints

Example: CREATE TABLE Enrollment (StudentID INTEGER NOT NULL REFERENCES Student(StudentID), CourseID INTEGER NOT NULL REFERENCES Course(CourseID), Grade INTEGER CHECK (Grade BETWEEN 0 AND 100), PRIMARY KEY (StudentID,CourseID)); Each column declaration gives its name then type, with commas between declarations.

CHECK is a constraint clause, not a standalone GRANT-like statement.

POSITIVE is not a general SQL constraint keyword; CHECK (Amount > 0) expresses positivity.

PRIMARY KEY implies uniqueness and non-NULL.

UNIQUE does not generally imply NOT NULL; NULL treatment differs by DBMS.

A CHECK can pass when its result is unknown, so add NOT NULL if missing values must be forbidden.

Parent updates/deletes may use NO ACTION/RESTRICT, CASCADE or SET NULL.

A referenced key is not inherently unchangeable: legality depends on matching children and the declared action.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: Constraints

FDB17 | Functional dependencies and attribute closure

X→Y means every legal pair of tuples agreeing on X also agrees on Y. It is a business constraint, not a correlation observed in one sample.

Full dependency requires all determinant attributes; partial dependency holds through a proper subset.

A prime attribute belongs to some candidate key; a nonprime attribute belongs to none.

To compute X^+, start with X and repeatedly add the right-hand side of any dependency whose left side is already present.

Stop when nothing changes.

X is a superkey exactly when X^+ contains the whole relation.

Remove attributes and recompute to test minimality.

Reflexivity gives X→Y when Y⊆X; augmentation permits XZ→YZ from X→Y; transitivity derives X→Z from X→Y and Y→Z.

Fundamental of Database Systems module

Advanced Database Systems module

FDB18 | Find all keys before deciding normal form

For R(A,B,C,D), let C→D, C→B and B→C. A never appears on a right-hand side, so every key needs A. (AB)+ gives ABCD through B→C→D; (AC)+ also gives ABCD.

AB and AC are minimal keys.

BC cannot derive A and is not a key.

A, B and C are prime; D is nonprime.

D depends on C alone and also on B through transitivity, proper subsets of those keys.

Thus 2NF fails; assuming atomic values, the highest form is 1NF.

Use the stated functional dependencies to justify each key and normal-form conclusion.

Confirm the actual dependency set before calculating closures; do not infer dependencies from unclear symbols.

Example

Worked example

- For F={A→B,B→C} on R(A,B,C), A+=ABC makes A the key. The chain through nonkey B causes a 3NF violation. If instead B were a candidate key, a dependency chain would not by itself prove such a violation.

Fundamental of Database Systems module

Advanced Database Systems module

FDB19 | Insertion, deletion and update anomalies

Suppose Enrollment repeats StudentName for every course.

An update anomaly requires changing every copy of the student's name; missed copies disagree.

An insertion anomaly prevents recording a new student until some enrollment exists.

A deletion anomaly loses the only record of a student's details when their last enrollment is deleted.

Revoke anomaly is not a normalization anomaly.

Normalization separates facts according to their determinants, reducing avoidable repetition and associated anomalies.

It does not require seeing bad current data: declared dependencies suffice.

Adding a generated row ID may make insertion technically possible, but does not remove repeated facts or fix the dependency design.

Reduced redundancy can improve integrity and storage use, but extra keys and joins mean neither storage savings nor faster retrieval is guaranteed.

Fundamental of Database Systems module

Advanced Database Systems module

FDB20 | First and second normal forms

1NF requires atomic values in the chosen relational domains and no repeating groups in cells.

It does not mean merely declaring a primary key.

A mapped ER design that correctly splits composite and multivalued structures gives ordinary 1NF tables; ER drawing alone guarantees no higher form.

2NF requires 1NF and no nonprime attribute partially dependent on any candidate key.

For Enrollment(SID,CID,SName,CTitle,Grade), assume key (SID,CID), SID→SName and CID→CTitle.

Split into Student(SID,SName), Course(CID,CTitle) and Enrollment(SID,CID,Grade).

Grade still depends on the whole pair.

A relation whose every candidate key is single-attribute has no nonprime partial-key dependency.

Higher forms inherit 2NF, so no partial dependencies establishes the minimum guaranteed stage only when 1NF is also assumed.

Fundamental of Database Systems module

Advanced Database Systems module

FDB21 | Third normal form, BCNF and higher forms

For every nontrivial FD X→A, 3NF permits it when X is a superkey or A is prime.

BCNF requires X to be a superkey.

The mnemonic whole key and nothing but the key describes the common single-key case; it cannot replace the full tests when keys overlap.

Remove dependencies of nonprime facts on nonkey determinants.

In Sale(Date,Customer,Product,Vendor,VendorCity,SalesRep), assume key (Date,Customer,Product), Product→Vendor and Vendor→VendorCity.

Product is a proper subset of the key, so Vendor and VendorCity violate 2NF.

With atomic values the highest form is 1NF, even though a transitive dependency also exists. 4NF addresses nontrivial multivalued dependencies with nonsuperkey determinants; 5NF addresses join dependencies.

Do not confuse a multivalued dependency with a list stored inside a cell.

Example

Worked example

- Employee(EID,DeptID,DeptName), EID→DeptID and DeptID→DeptName, splits into Employee(EID,DeptID) and Department(DeptID,DeptName). Each department name is stored once.

Fundamental of Database Systems module

Advanced Database Systems module

FDB22 | Decomposition: lossless and dependency preserving

A lossless decomposition reconstructs exactly the original legal tuples by joining its parts; a lossy one creates spurious combinations.

For a binary decomposition into R1 and R2 under FDs, it is lossless when their common attributes determine all of R1 or all of R2.

Sharing a column without this condition is insufficient.

Dependency preservation means the original constraints can be enforced on the individual relations without joining them.

Losslessness and dependency preservation are separate properties.

A practical 3NF synthesis uses a minimal FD cover, creates a relation for each determinant and its dependent attributes, removes contained relations, and adds a relation containing a candidate key if none already does.

BCNF decomposition removes an offending X→Y by separating XY and the remaining attributes with X; it can sacrifice dependency preservation.

Fundamental of Database Systems module

Advanced Database Systems module

Jan Chomicki, University at Buffalo: Relational database design

FDB23 | Choose data types and storage levels

CHAR(n) is fixed length; VARCHAR(n) has variable length up to a limit.

DATE represents a date and BLOB binary content such as images.

INTEGER represents whole numbers; DECIMAL(p,s) represents exact decimal values, making it suitable for money when scale is specified.

FLOAT/REAL/DOUBLE are approximate; names, widths and precision depend on the DBMS.

Numeric type widths depend on the DBMS and type; do not assume one universal width.

Primary storage includes RAM and processor cache and is usually volatile.

Secondary storage includes persistent SSDs/HDDs used online; database pages are copied into buffers before CPU processing.

Tertiary storage includes tape or removable archive media.

Choose types from meaning, range and accuracy requirements, not a blanket rule that integers are always faster than strings.

Fundamental of Database Systems module

Advanced Database Systems module

FDB24 | Records, pages and primary file organization

A page/block is a unit of database I/O and holds several records.

For fixed unspanned records of size r in usable page space B, the blocking factor is floor(B/r), and n records occupy ceil(n/blocking factor) pages.

Page headers and slots reduce usable space.

A heap has no search-key order and supports simple inserts, but searches may scan all pages.

A sorted file supports efficient range scans and binary search on the ordering field, with more costly insertions or overflow handling.

A hashed file chooses a bucket by hash(key), supports equality lookup well and requires collision/overflow handling; hash order does not support ordinary ranges.

Choose organization from actual reads, writes and predicates.

Partitioning splits a large dataset; replication copies it; migration moves it; archiving removes older data from active service.

Fundamental of Database Systems module

Advanced Database Systems module

FDB25 | Ordered, dense, sparse and multilevel indexes

An index stores search keys with pointers or row identifiers.

It often speeds selective retrieval but consumes space and must be maintained on writes.

A full scan can win for a tiny table or a query returning most rows.

Indexes do not themselves establish referential integrity.

In ordered-file indexing terminology, a primary ordered index is on the file's ordering key and can have one entry per page, hence sparse.

A clustering index is on a nonunique ordering field, often with an entry per distinct value.

Secondary indexes provide other access paths and must represent all searchable values with record pointers or pointer lists.

A dense index represents each search value; sparse indexes rely on ordered data between entries.

Multilevel indexes index lower index levels.

B/B+ trees support ordered and range search; hashing targets equality; bitmap indexes suit some analytic workloads.

No one type is universally best.

Fundamental of Database Systems module

Advanced Database Systems module

FDB26 | SQL command categories and exact scope

DDL changes structure: CREATE, ALTER, DROP, commonly RENAME and TRUNCATE in textbook classifications.

DML retrieves or changes data: SELECT, INSERT, UPDATE, DELETE.

Some texts separate SELECT as DQL.

DCL manages privileges through GRANT and REVOKE.

TCL includes COMMIT, ROLLBACK and SAVEPOINT.

Classify a statement by its effect, not its length.

DELETE removes rows and keeps the table; DROP TABLE removes the table object; ALTER changes its definition.

TRUNCATE empties a table using implementation-specific mechanisms.

PostgreSQL allows transactional rollback of TRUNCATE, so no rollback information is retained is not a universal SQL property.

SQL Server permits DROP DATABASE Sales, suppliers; that comma-list syntax is vendor-specific.

SQLite's .databases lists attached names/files in its command-line shell; .database is not the documented command.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: TRUNCATE

Microsoft: DROP DATABASE (Transact-SQL)

SQLite: Command-line shell

FDB27 | SELECT and its result columns

SELECT Name,Address FROM Customer; produces only those mailing-label fields.

SELECT * selects all columns; DISTINCT removes duplicate result rows; AS supplies an alias; ALL permits duplicates and does not mean all columns.

SQL normally preserves duplicates without DISTINCT.

A query reading table columns needs an appropriate FROM source.

A literal-only query such as SELECT 1 may legally omit FROM.

Typical written clause order is SELECT … FROM … WHERE … GROUP BY … HAVING … ORDER BY ….

Conceptually form FROM/JOIN rows, filter WHERE, group, filter HAVING, select expressions and then order the result.

SELECT alone normally reads without changing the database state.

WHERE alone is a clause rather than a complete read-only statement, so it cannot be compared as an equivalent executable command.

Fundamental of Database Systems module

Advanced Database Systems module

FDB28 | Predicates, strings and NULL

Use single quotes for SQL string literals: WHERE ID = 'B001'.

Standard double quotes delimit identifiers, although some vendor modes accept them for strings.

Qualify ambiguous columns with aliases such as E.Department.

Conditions use =, <>, <, >, AND, OR and NOT; use parentheses to make precedence clear.

LIKE 'Ab%' matches a prefix, and BETWEEN includes its endpoints.

NULL represents a missing/unknown value.

Comparisons such as Salary = NULL are unknown, not true; use IS NULL or IS NOT NULL.

WHERE retains only true conditions.

An index-friendly year predicate for timestamps is BirthDate >= DATE '2000-01-01' AND BirthDate < DATE '2001-01-01', avoiding omitted times on December 31.

Expression indexes can support functions, so functions do not universally prevent indexing.

Fundamental of Database Systems module

Advanced Database Systems module

FDB29 | INSERT, UPDATE and DELETE

INSERT INTO Hotel(HotelID,HotelName,SubCity) VALUES (1,'Hilton','Yeka'); pairs each named column with a compatible value.

Counts and types must match; keys and other constraints still apply.

Without a column list, values must follow the table's declared column order.

UPDATE Book SET Price = 600 WHERE ID = 'B001'; changes matching rows.

UPDATE without WHERE targets every row.

DELETE FROM Publisher WHERE Name = 'XWZ'; deletes matching publisher rows, subject to reference policies.

DELETE FROM r; removes every row but leaves r defined.

Both clear relation entries and delete rows describe that effect.

Several UPDATE statements can all be valid while performing different tasks; most appropriate requires a requested change and a specified quoting dialect.

Fundamental of Database Systems module

Advanced Database Systems module

FDB30 | CREATE and ALTER tables

CREATE TABLE Employee (EmpID INTEGER PRIMARY KEY, Name VARCHAR(80)); declares two columns.

The name and its type stay together; the comma separates whole column declarations.

To add a primary key later: ALTER TABLE Employee ADD PRIMARY KEY (EmpID); existing data must be unique and non-NULL.

A table cannot gain a second primary key without replacing the existing constraint.

ALTER TABLE Employee ADD COLUMN Email VARCHAR(120); adds a column in PostgreSQL-style syntax.

ALTER TABLE Employee DROP COLUMN Email; removes it.

ALTER TABLE Employee ADD CONSTRAINT uq_email UNIQUE (Email); adds a named constraint.

Use ALTER rather than DALTER, and do not separate column_name and column_type with a comma; validate the resulting SQL before execution.

Fundamental of Database Systems module

Advanced Database Systems module

FDB31 | Join behavior and preservation

An inner join returns matching pairs satisfying ON.

An equijoin uses equality; a theta join permits another comparison.

A natural join is ordinarily an inner equality join on every same-named column and merges those duplicate columns.

JOIN ... USING (DeptID) specifies shared join columns explicitly.

CROSS JOIN forms every pair without a matching condition.

LEFT OUTER JOIN also preserves unmatched left rows, padding right columns with NULL; RIGHT preserves right rows and FULL preserves both.

Therefore inner and natural inner joins both discard unmatched rows.

A self-join uses aliases for two occurrences of one base table.

Each ordinary join operator combines two inputs, but its inputs can be earlier joins, so one query can join many tables.

Distinguish base-table count, table occurrences and binary operator arity.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: Table expressions

FDB32 | Aggregation, GROUP BY and HAVING

COUNT(*) counts rows, COUNT(x) counts non-NULL x values; SUM and AVG summarize non-NULL numeric values.

SQL's usual average function is AVG, not AVERAGE.

GROUP BY creates one group per distinct grouping value; HAVING tests aggregate groups after WHERE has filtered individual rows.

SELECT Department, AVG(Salary) FROM Employee WHERE Department <> 'Sales' GROUP BY Department HAVING AVG(Salary) > 35000; lists qualifying department averages.

An ordinary AVG(Salary) predicate cannot be placed directly in WHERE.

Selecting EmpID, FirstName and LastName while grouping only by Department is also invalid unless the DBMS can establish the required functional dependence.

The grouping column need not itself appear in SELECT; check the functional dependence of other selected columns.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: GROUP BY and HAVING

FDB33 | Subqueries and views

A subquery is a SELECT nested inside another statement.

It may appear in WHERE, HAVING or FROM, not only the first two.

IN tests membership; EXISTS tests whether at least one qualifying row exists.

Correlated subqueries refer to the outer row.

A scalar subquery must produce an appropriate single value.

Beware NOT IN when its subquery can return NULL; NOT EXISTS often expresses the intended absence more safely.

CREATE VIEW CSStudent AS SELECT StudentID,Name FROM Student WHERE Major = 'CS'; stores a query definition and presents a virtual table.

Views can simplify repeated queries, expose derived values and restrict access when configured correctly.

An ordinary view does not necessarily store its result; a materialized view does.

Rewriting a subquery as a join can change duplicates, and does not guarantee faster execution.

Fundamental of Database Systems module

Advanced Database Systems module

FDB34 | Set operations and type conversion in SQL

SELECT ID FROM A UNION SELECT ID FROM B; combines compatible result columns and removes duplicate rows by default.

UNION ALL retains duplicates.

Explicit UNION DISTINCT is accepted in some dialects, including PostgreSQL; it is not a separate algebra rule.

INTERSECT returns common rows; EXCEPT returns first-result rows absent from the second.

Inputs need the same column count and compatible corresponding types.

CAST('42' AS INTEGER) converts an expression to a requested type if the value permits it.

CONVERT is vendor-specific and can also perform type conversion; specify the SQL dialect when comparing it with CAST.

In SELECT xyz FROM table1 UNION xyz FROM table2, the second SELECT is missing.

In the corrected query, xyz is a selected column/expression, not a table name.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: Combining queries

PostgreSQL: Value expressions

FDB35 | Relational algebra: calculate results

Selection σ filters tuples; projection π retains attributes and removes duplicate tuples under set semantics.

Rename ρ changes relation or attribute names.

Union, difference and intersection require union-compatible schemas.

Difference R−S is not commutative; union and intersection are.

Inner joins are commutative up to attribute order when predicates are adjusted; left outer joins are not generally commutative.

Identify the join variant before predicting its behavior.

A Cartesian product has $|R|\times|S|$ tuples and degree degree(R)+degree(S), renaming conflicting attributes if necessary.

Thus $200\times100=20{,}000$ and $5\times7=35$.

Join equals selection over a product; intersection can be derived from difference.

A common minimal basis contains selection, projection, union, difference, product and rename.

Assignment is a useful notation, not an indispensable primitive; lists of basic operations vary across textbooks.

Fundamental of Database Systems module

Advanced Database Systems module

FDB36 | Relational calculus: state the desired facts

Relational algebra specifies a sequence of operations and is procedural at the logical level.

Relational calculus specifies a predicate describing desired results and is declarative/nonprocedural.

A tuple variable ranges over tuples; a domain variable ranges over individual attribute values.

Safe/range-restricted expressions produce results bounded by the database's active domain.

Tuple form {s | Student(s) AND s.Major='CS'} selects student tuples with that major.

A domain form can request names n satisfying ∃i,m (Student(i,n,m) AND m='CS').

Existential quantification means some matching tuple exists; universal quantification means every required case meets a condition.

SQL is declarative even though an optimizer later chooses a procedural execution plan.

Fundamental of Database Systems module

Advanced Database Systems module

FDB37 | Transactions and the ACID properties

A transaction groups operations into one logical unit.

Atomicity applies all its effects or none.

Consistency means a correct transaction preserves declared invariants.

Isolation controls interference between concurrent transactions; serializable isolation provides behavior equivalent to some serial execution.

Durability preserves committed effects after a crash, typically through durable logging rather than forcing every data page immediately.

For a transfer, subtracting 100 from one balance and adding 100 to another must commit together; losing either half violates atomicity and may violate the total-balance invariant.

BEGIN starts work, COMMIT completes it, ROLLBACK undoes uncommitted work and SAVEPOINT marks an intermediate rollback point.

A failed transaction may roll back while other users' committed transactions remain.

SQL transaction syntax and error-handling constructs differ by DBMS; @@ERROR/IF is SQL Server-specific.

Fundamental of Database Systems module

Advanced Database Systems module

FDB38 | Choose a design technique from the requirement

Normalization reduces dependency-driven anomalies.

Indexing supplies search paths.

Denormalization deliberately repeats or precomputes facts to reduce selected read costs, increasing maintenance work.

Partitioning divides rows or columns; replication creates copies to improve locality or availability while complicating updates.

These techniques can coexist.

An e-commerce or gaming label alone does not select one.

Privacy requires a threat model.

Authentication verifies identity; role-based authorization limits allowed records/actions; encryption protects data at rest or in transit; auditing records activity.

Both encryption and access control can protect patient data.

Scalability, availability, fast retrieval, redundancy control and security are all reasonable design considerations.

Functional requirements describe behavior; nonfunctional requirements quantify qualities such as 99.9% uptime or response time.

Authentication and encryption mechanisms alone do not uniquely establish that distinction.

In a CMS, tags/categories describe meaning, while foreign keys preserve valid references between content and tags.

A ContentTag(ContentID,TagID) association models multiple tags per item; these functions complement each other.

Fundamental of Database Systems module

Advanced Database Systems module

FDB39 | Integrated design: student registration and dormitory allocation

State the rules first: each student belongs to one department, may enroll in many courses, and has at most one current dorm assignment.

Each course accepts many students; each dorm holds at most six students.

Identify Department, Student, Course and Dorm entities; Enrollment is the M:N relationship with Grade.

Use Department(DeptID,Name), Student(SID,Name,DeptID,DormID), Course(CID,Title), Enrollment(SID,CID,Grade) and Dorm(DormID,FloorNumber).

Declare all entity IDs primary keys; Student.DeptID and Student.DormID reference their parents; Enrollment has composite key (SID,CID) and both references.

Make DormID nullable until assignment.

Separate StudentPhone(SID,Phone) for multiple contacts.

Normalize repeated department/course facts into their owning tables.

A simple foreign key does not enforce the six-student capacity; use a concurrency-safe trigger, allocation transaction or explicit slot design with six unique slots per dorm.

Example

Worked example

- Complete the reference as FOREIGN KEY (DormID) REFERENCES Dorm(DormID). STUDENT is the referencing child; DORM is referenced. Consistent names and an added Student.DormID column are required before execution.

Fundamental of Database Systems module

Advanced Database Systems module

FDB40 | Check SQL syntax and question assumptions

Justify each answer using the schema, SQL rules and stated assumptions.

Owner and strong entity can both describe the entity supporting a weak entity; row and record are synonyms in a relational table; difference and set-difference are the same operator.

DELETE, UPDATE and the invalid command REMOVE are not DROP; a question asking which is not DROP can therefore admit several choices.

Compare all choices with the relevant rule before selecting an answer.

For invalid SQL, identify the syntax error and validate the corrected statement before analyzing its result.

For unsupported cardinalities, request business rules and direction.

For best/efficient/important questions, specify workload, quality goal or comparison baseline.

A diagram-dependent conclusion needs the missing diagram or an explicit description of its relevant symbols.

Identify the smallest clarification needed when a question does not support a unique answer.

Fundamental of Database Systems module

Advanced Database Systems module

FDB-S01 | Algebra operator lists and derivable operations

A list of basic relational-algebra operators is a teaching convention, distinct from a claim that every listed operation is logically indispensable. The course module uses selection, projection, Cartesian product, set difference, union and intersection as its basic-operation list. Natural join and assignment are not members of that particular list.

Under set semantics, union includes tuples from either compatible input, intersection retains tuples common to both, and difference keeps first-input tuples absent from the second. Intersection can be expressed using difference, while a matching join can be expressed as selection over a Cartesian product. A smaller primitive basis may therefore omit intersection even when it is introduced as a basic operation.

Fundamental of Database Systems module · page 24

FDB-S02 | Relational systems with richer types

An object-relational DBMS retains relational data management while supporting richer extensions such as user-defined data types, functions and operators. PostgreSQL is an example. Richer relational types do not by themselves require all data to form a parent-child tree; that representation is the defining concern of hierarchical modeling.

PostgreSQL documentation - What Is PostgreSQL?

Advanced Database Systems

Complete Study Notes

ADB01 | Objects, identity, classes and persistence

An object combines identity, state and behavior.

A class specifies the attributes and operations of its instances; an object is one instance.

The object identifier (OID) distinguishes objects independently of their attribute values and remains stable over that object's lifetime.

Two objects can have equal values and different identities; changing a name does not create a new identity.

Persistence lets an object survive beyond the program execution that created it.

An OODBMS adds durable storage, transactions, querying and controlled sharing to object concepts.

It does not primarily present objects as rows and tables.

An object-relational DBMS extends a relational system with richer types; ORM maps programming objects into relational tables.

These three approaches differ.

Complex CAD or multimedia structures can suit an object model, but many-to-many relationships alone do not require abandoning relational design.

University of Pittsburgh: Object-oriented databases

ADB02 | Encapsulation, abstraction and object associations

Encapsulation groups state with operations and controls access to implementation details.

Account.withdraw(amount) can enforce a balance rule while hiding its storage.

Abstraction presents the relevant contract-what withdrawal does-without exposing how it is implemented.

These ideas overlap in information hiding, but combining state and methods specifically describes encapsulation.

An association links objects, often through references; Department.employees can reference Employee objects.

It is different from inheritance, which means an is-a relationship between types.

Methods can invoke operations on other objects, supporting collaboration.

A class defines structure and behavior; an attribute describes state; a method implements an operation.

Do not treat each of these as an interchangeable name for the whole object.

University of Pittsburgh: Object-oriented databases

ADB03 | Inheritance, polymorphism and safe substitutability

Inheritance derives a subtype from a superclass, reusing applicable attributes and behavior.

GraduateStudent is a Student; association instead says Student belongs to Department.

EER teaches inherited attributes and relationships, while a programming object model also specifies behavior.

Subtype polymorphism lets a superclass/interface reference denote objects of different compatible subclasses.

The same operation can dispatch to different implementations: Shape.area() calls Circle.area() or Rectangle.area() according to the actual object.

This is not arbitrary conversion of an object into an unrelated class, nor dynamic class mutation.

Inheritance establishes a subtype relationship; polymorphism uses that relationship through a common interface.

Distinguish inheritance as a structural relationship from polymorphism as a use of that relationship at runtime.

Oracle Java Tutorials: Polymorphism

ADB04 | Object type constructors

In a common object model, an object is described by (OID,type constructor,state).

An atom is one basic value, such as an integer, real, Boolean or string.

A tuple constructor groups named component values into one structured record, such as (Street:'A',City:'B').

It is not a collection of records.

A set is an unordered collection without duplicates; a bag allows duplicates; a list has order; an array provides indexed positions.

Constructors can be nested to represent complex objects, such as a tuple with a set of phone numbers.

An array is not an atomic constructor merely because stored as one programming variable.

A type constructor defines the shape of a value; it is not the same as an executable constructor operation that creates a new object.

University of Pittsburgh: Object type constructors

ADB05 | ODL, OQL and inverse relationships

Object Definition Language (ODL) is the ODMG schema-definition language for object databases; it describes interfaces/classes, attributes, operations and relationships.

Object Query Language (OQL) expresses queries.

Host-language bindings connect these definitions and operations to languages such as Java or C++.

Defining a logical schema is different from implementing its storage.

An inverse declaration pairs opposite directions of the same relationship.

Conceptually, Department.employees is inverse to Employee.department: when an employee belongs to department D, the corresponding employee appears in D's collection.

The declaration expresses relationship consistency; it is not an inheritance keyword or an ordinary scalar attribute.

An MCQ calling ODL both a DDL and a language for developing logical schemas offers overlapping true descriptions, so its exact wording matters.

Juniata College: Object Databases

ADB06 | Transactions, ACID and state transitions

A transaction is a logical unit of database work, including read-only work.

Atomicity means all effects or none; consistency means correct work preserves integrity rules; isolation controls concurrent interference; durability preserves committed effects despite failure.

A DBMS cannot automatically make incorrect business logic consistent.

Active means operations are executing.

Partially committed means the last operation finished but durable commitment is not yet assured.

Committed means commitment succeeded.

A failure leads to Failed, then Aborted after rollback; a transaction may terminate or be restarted as a new attempt.

Concurrency control primarily enforces isolation; recovery supports atomicity and durability.

Transaction management is the broader subsystem, so both it and recovery management can be legitimate answers to broad responsibility questions.

Example

Worked example

- A=5,000 and B=8,000 become 4,250 and 8,750 after transferring 750 from A to B. Both balances total 13,000. Unchanged or reverse-transfer balances can also preserve that sum; only the requested debit/credit implements the specified successful transfer.

Fundamental of Database Systems module

Advanced Database Systems module

ADB07 | Read/write operations, buffers and log records

read_item(X) locates X's page, brings it into a buffer if absent, and copies X into a program variable. write_item(X) copies the new program value into the buffered page, marking it dirty.

Writing that page to disk can occur later according to the buffer policy; write_item is not inherently an immediate disk flush.

Insert and delete also modify data; sorting is a query-processing operation rather than a primitive item update.

A conceptual log can contain START, UPDATE(T,X,old,new), COMMIT, ABORT and END records.

Before-images support undo; after-images support redo.

Real systems may use physical or logical records, and need not log every read.

COMMIT records successful completion; an ABORT record can mark the start of rollback rather than prove every undo step is complete.

Fundamental of Database Systems module

Advanced Database Systems module

ADB08 | Diagnose concurrency anomalies from a trace

Lost update: T1 and T2 both read 500; T2 adds 200 and writes 700; T1 adds 100 to its stale value and writes 600.

Correct serial execution yields 800.

Dirty read: T2 reads a value written by uncommitted T1; if T1 aborts, dependent work must be reconsidered.

Reading one's own changes is not a dirty read.

Nonrepeatable read means a repeated read of one row sees a committed change.

A phantom means repeating a predicate query finds a changed qualifying set after another transaction inserts, deletes or changes matching rows.

Incorrect summary mixes mutually inconsistent versions: a transfer moves 10 from A=100 to B=100; a sum reading old A=100 and new B=110 gets 210 instead of the invariant 200.

A single committed update from B=20 to B=50 does not alone prove an invalid summary of 90.

Fundamental of Database Systems module

Advanced Database Systems module

ADB09 | Serial schedules, conflicts and precedence graphs

A serial schedule executes each transaction's operations consecutively.

A serializable schedule is equivalent to some serial order; it may interleave transactions.

Equality of one final value is insufficient evidence because different executions can accidentally coincide.

Two operations conflict when they belong to different transactions, access the same item and at least one writes.

Read/read pairs do not conflict.

For conflict serializability, make one graph vertex per transaction.

For every earlier conflicting operation of Ti and later operation of Tj, add Ti→Tj.

Acyclic means conflict-serializable; a topological ordering gives a valid serial order.

Cyclic means not conflict-serializable.

This test is stronger than view serializability, which also compares read-from relationships and final writers and may permit some schedules with blind writes.

Example

Worked example

- r1(X), w2(X), r2(Y), w1(Y) gives T1→T2 on X and T2→T1 on Y, a cycle. r1(X), w1(X), r2(X), w2(X) gives only T1→T2 and is conflict-serializable.

Fundamental of Database Systems module

Advanced Database Systems module

ADB10 | Recoverable, cascadeless and strict schedules

Recoverable means if Tj reads Ti's value, Tj cannot commit before Ti commits.

Otherwise Ti might abort after Tj already committed dependent work.

Cascadeless means a transaction reads only values written by committed transactions, avoiding cascading aborts.

Strict means no other transaction reads or overwrites an item written by an uncommitted transaction.

Strict ⇒ cascadeless ⇒ recoverable.

For w1(X), r2(X), c2, a1, T2 committed a dirty dependency, so the schedule is unrecoverable.

Moving c2 after c1 makes that dependency recoverable, but the earlier dirty read still makes it not cascadeless.

Serializability and recoverability are separate tests: one concerns equivalent transaction order and the other safe commit/rollback dependencies.

Fundamental of Database Systems module

Advanced Database Systems module

ADB11 | Locks, compatibility, conversion and managers

A shared (S) lock permits reading; multiple S locks on the same item can coexist.

An exclusive (X) lock permits its holder to read and write and conflicts with another S or X on that item.

Locks on different items do not conflict merely because one is exclusive.

These are the basic lock-based rules; multiversion systems can permit additional snapshot-reader behavior.

A transaction must own the appropriate lock before access and can release only its own locks.

S→X is an upgrade, possible only after conflicting other holders leave.

X→S is a downgrade.

The lock manager decides requests; the lock table stores granted locks and waiting requests.

One transaction may request a lock on an already locked item: it is granted if compatible, otherwise waits or aborts under the policy.

No double-locking is not a prohibition on shared readers.

Held / requested	S	X
S	Compatible	Conflict
X	Conflict	Conflict

Fundamental of Database Systems module

Advanced Database Systems module

ADB12 | Two-phase locking and its variants

Basic 2PL has a growing phase that acquires/upgrades locks and a shrinking phase that releases/downgrades them.

Once any lock is released, no new lock may be acquired.

The lock point is the instant the final lock is obtained. 2PL guarantees conflict serializability for well-formed transactions, but can deadlock and basic 2PL alone need not prevent cascading aborts.

Strict 2PL retains exclusive locks until commit/abort.

Rigorous 2PL retains both shared and exclusive locks until that point.

Conservative/static 2PL obtains all needed locks before execution and does not begin while only a subset is available, preventing hold-and-wait deadlock.

It requires knowing the lock set and can reduce concurrency.

Two-phase locking orders access to data; two-phase commit coordinates a distributed commit decision.

Fundamental of Database Systems module

Advanced Database Systems module

ADB13 | Deadlock detection, prevention and starvation

Deadlock requires a waiting cycle: T1 holds X and wants Y; T2 holds Y and wants X. In a wait-for graph, an edge Ti→Tj means Ti waits for a lock held by Tj.

A cycle indicates deadlock under the ordinary lock-wait model.

Abort a suitable victim, undo its effects and release its locks; repeat if other cycles remain.

Aborting every deadlocked transaction also breaks cycles but wastes more work.

Fixed resource ordering and conservative 2PL prevent cycles.

A no-wait policy aborts immediately on an incompatible lock request and needs no age timestamp.

A timeout aborts long waits but may mistake slow progress for deadlock.

Starvation is indefinite postponement while other work proceeds, without a required cycle.

Fair queues, sensible victim selection and retaining original transaction timestamps on retry reduce starvation; deadlock freedom alone does not prove fairness.

Fundamental of Database Systems module

Advanced Database Systems module

ADB14 | Wait-die and wound-wait: compare requester and holder

A smaller timestamp means an older transaction.

In wait-die, an older requester may wait for a younger holder; a younger requester needing an older holder aborts itself.

In wound-wait, an older requester aborts the younger holder; a younger requester waits for the older holder.

Always identify requester versus holder before applying the names.

Retain the original timestamp when restarting an aborted transaction so it gains relative priority as new transactions arrive.

Starvation avoidance additionally assumes holders eventually finish and scheduling is fair.

In wait-die, an older transaction may wait for a younger one; a younger transaction requesting a resource held by an older one aborts.

Deadlock-prevention priority rules are also different from basic timestamp ordering's read/write validity tests.

Requester relative to holder	Wait-die	Wound-wait
Older	Wait	Abort younger holder
Younger	Abort requester	Wait

Fundamental of Database Systems module

Advanced Database Systems module

San Jose State University: Deadlocks and distributed databases

ADB15 | Basic timestamp ordering: execute the tests

Assign each transaction TS(T).

For item X track read_TS(X), the largest timestamp of a successful reader, and write_TS(X), that of its latest successful writer.

A read by T aborts if TS(T)<write_TS(X); otherwise it succeeds and updates read_TS.

A write aborts if TS(T)<read_TS(X) or TS(T)<write_TS(X); otherwise it succeeds and sets write_TS.

These tests preserve timestamp order.

For TS(T)=10 and write_TS(X)=20, T cannot read X: it would observe a logically later write.

Thus the transaction aborted can be older, contrary to the guides' blanket younger-always-aborts claim.

The Thomas write rule can ignore an obsolete write instead of aborting in one case, but that is a different protocol and must be stated.

Basic timestamp ordering avoids lock-wait deadlocks but may cause repeated restarts.

Fundamental of Database Systems module

Advanced Database Systems module

Virginia Tech: Timestamp protocols

ADB16 | Optimistic control, versions and lock granularity

Optimistic/validation control reads and computes privately, validates its read/write sets against overlapping transactions, then publishes if validation succeeds; otherwise it retries.

It works well when conflicts are uncommon, but high contention wastes work on aborts.

Multiversion control keeps versions so a reader can use an appropriate snapshot.

Snapshot isolation is not automatically serializable: concurrent writes to different rows can violate a shared invariant.

Fine-grained row locks permit more independent operations but require tracking many locks.

Coarse table/database locks simplify management but block unrelated rows and reduce concurrency.

Intention locks advertise planned descendant locks in a hierarchy; escalation replaces many small locks with a coarser one.

Small granularity does not inherently cause more logical conflicts.

Choose granularity from contention and management overhead, not a universal rule.

Fundamental of Database Systems module

Advanced Database Systems module

ADB17 | Query processing: representation before execution

Parsing checks SQL grammar; semantic analysis resolves names, types and access permissions.

Translation constructs a logical representation, typically relational algebra or a logical query tree.

Optimization chooses equivalent operations, join order and physical access methods.

Evaluation executes that plan and returns results.

Not every implementation exposes exactly the same phase boundaries.

A query tree has base relations at leaves and operators at internal nodes; a query graph emphasizes relations and predicates.

A physical execution plan adds scan/index choices and algorithms.

Query processing is the whole activity; selecting a suitable efficient strategy is optimization.

DDL is not a mandatory stage before each SELECT, so authentication→DDL→DML permutations do not describe the processing sequence.

Fundamental of Database Systems module

Advanced Database Systems module

ADB18 | Heuristic and semantic query rewrites

Heuristics reduce intermediate work: push selective row filters toward base tables; replace product followed by a join predicate with a join; project unneeded columns away while retaining join/filter columns; reorder inner joins when equivalence permits; reuse common expressions when worthwhile.

Split conjunctive selections when each predicate can be applied earlier to its own input.

Semantic optimization additionally uses trusted constraints, such as a key proving a join cannot multiply matching rows.

Cost-based optimization estimates candidate plans instead of relying on rules alone.

Outer joins, NULLs, duplicates and aggregation restrict equivalences: moving a right-table filter from ON to WHERE after a left join may delete its preserved unmatched rows.

Fewer intermediate rows is a useful heuristic, not proof of minimum runtime.

Fundamental of Database Systems module

Advanced Database Systems module

ADB19 | Estimate costs, cardinalities and selectivity

A cost model estimates resources for a candidate plan: page I/O, CPU, memory, temporary storage and network traffic.

Catalog statistics include row/page counts, distinct values and distributions.

Selection with fraction s predicts s×N rows.

For non-NULL equijoin keys, a common estimate is $|R|\times|S|/\max(V(R,k),V(S,k))$, assuming uniform values and containment of the smaller distinct-value domain within the larger.

Adjust counts for NULLs; if either non-NULL input is empty, the result is zero.

Skew, correlations or incomplete domain overlap can invalidate this estimate; known disjoint domains give zero matches.

For a simple rotating-disk model, cost = seeks×seek_time + transferred_blocks×block_transfer_time.

Five seeks at 5 ms plus ten blocks at 1 ms cost 35 ms.

Disk I/O often dominates textbook local-disk examples; CPU can dominate in-memory work and network can dominate distributed plans.

State the assumed cost objective before ranking strategies.

Fundamental of Database Systems module

Advanced Database Systems module

CMU 15-445/645 Spring 2024 Query Optimization slides

Leis et al., How Good Are Query Optimizers, Really?

ADB20 | Join algorithms and access-path tradeoffs

Tuple nested loops scan the inner relation for each outer tuple.

Page nested loops approximately cost B(R)+B(R)×B(S) page reads.

With M buffers, block nested loops can cost B(R)+ceil(B(R)/(M−2))×B(S), reserving an inner and output buffer.

Try both outer choices.

An indexed nested-loop join probes an inner index for each qualifying outer tuple; it suits a small outer result and a useful index.

Sort-merge joins sorted join keys and must include any sorting cost.

Hash join partitions/builds a hash structure and probes it; equality joins suit it, ranges generally do not.

A one-pass in-memory hash join can read each input once when the build side fits.

Include output/materialization cost if the exercise requires it.

Fundamental of Database Systems module

Advanced Database Systems module

ADB21 | Compare actual query strategies

Compare only plans returning the same result under the same assumptions.

Add scans, probes, sorting, intermediate writes and communication as applicable; do not count an omitted cost in only one plan.

Record memory limits and whether inputs are already ordered.

Choose the lowest estimated cost for the specified objective, then test estimates against actual execution.

Example: R has 100 pages, S has 1,000 pages and M=12 buffers.

Block nested loops with R outer cost 100+ceil(100/10)×1,000=10,100 reads; S outer costs 1,000+ceil(1,000/10)×100=11,000.

If a selective R index costs 3 reads and returns 20 outer tuples, with each S index probe costing 4 reads including matches, indexed nested loops cost 3+20×4=83 reads.

That wins under these stated assumptions; it is not a universal join ranking.

Fundamental of Database Systems module

Advanced Database Systems module

ADB22 | Pipelining, parallelism and caching

Materialization stores an intermediate result before the next operator consumes it.

Pipelining passes tuples as they become available, overlapping producer and consumer work and reducing temporary I/O. A sort is normally blocking; a hash join must build its hash table before probing can output, though later probe/output can stream.

Pipelining does not require separate CPUs.

Parallel execution uses multiple workers, for example partitioned scans/joins, independent subplans or operator pipelines.

Interleaving on one CPU is concurrency; simultaneous cores provide parallelism.

Result caching reuses answers and needs invalidation when underlying data changes.

Plan caching reuses an execution strategy, not the rows returned.

Buffer caching keeps data pages; materialization retains intermediate rows.

Specify the kind of query caching being discussed.

Fundamental of Database Systems module

Advanced Database Systems module

ADB23 | Indexes, statistics and re-optimization

B/B+ tree indexes support ordered and range predicates; hash indexes target equality; bitmap indexes can help some analytic selections.

Clustered storage keeps related rows close.

An index may reduce reads but adds space and update work; scanning can be faster when most rows qualify.

Commonly useful is not a unique property of one index type.

Re-optimization chooses a plan again when statistics, parameters or observed cardinalities suggest the old plan is unsuitable.

Updating statistics does not itself execute the query.

Use EXPLAIN and measured execution to compare estimates, spills, scans and join choices.

Avoid universal rules that joins always beat subqueries, functions can never use indexes or more indexes always improve performance.

Fundamental of Database Systems module

Advanced Database Systems module

ADB24 | Security, integrity and inference control

Authentication establishes identity; authorization/access control determines permitted actions.

Encryption protects stored or transmitted content; integrity constraints restrict valid data; auditing records actions.

A user who can view but cannot change a record has restricted access privileges.

A view can restrict visible rows/columns, but its privilege and execution rules must support that restriction.

Inference control concerns learning protected individual facts indirectly from allowed aggregate answers.

Releasing SUM for a group and SUM for the same group excluding one known person may disclose that person's value by subtraction.

Suppressing individual rows alone is insufficient; protections may restrict queries, aggregate sizes or add appropriately designed noise.

Access-restricted views can expose aggregates while hiding details; inference control additionally concerns preventing indirect disclosure.

Fundamental of Database Systems module

Advanced Database Systems module

Jagannathan and Wright: Private Inference Control for Aggregate Database Queries

ADB25 | GRANT, REVOKE and privilege scope

GRANT SELECT ON Catalogue TO Mohammed; permits reads.

REVOKE DELETE, UPDATE (Price,Year) ON Catalogue FROM Mohammed; removes the specified table deletion and column update grants in SQL dialects supporting that form.

DELETE applies to rows/table access rather than particular columns; UPDATE can be column-scoped.

REVOKE removes permission, whereas GRANT gives it.

Roles group privileges.

WITH GRANT OPTION permits a recipient to grant a privilege onward; removing an upstream grant can cascade or be restricted according to the chosen command and DBMS.

Removing one direct grant may leave effective access through another role or grant path.

A view is a stored query definition, not encryption.

Least privilege assigns only the access needed for the task.

Fundamental of Database Systems module

Advanced Database Systems module

ADB26 | Classify failures and choose the starting resource

Transaction failures include invalid operations, constraint violations and concurrency-control aborts.

Enforcing concurrency can deliberately abort work; it does not prevent every transaction failure.

A system crash loses volatile memory while disks may survive.

Media failure damages persistent storage and normally requires restoring a backup, then replaying available archived logs.

Human mistakes may require point-in-time restoration before the bad committed operation.

Restart recovery restores a valid database after a crash while preserving durable commits and removing incomplete effects.

Rollback handles failed/uncommitted work; redo preserves committed work missing from pages; checkpoints reduce log-search work; backups provide another stored copy.

These are cooperating mechanisms, not four mutually exclusive complete alternatives.

Buffer management is not durable backup.

Generic recovery-manager terminology should not be mistaken for an exact MySQL component specification.

A full backup copies the database to separate storage; scheduled backups plus archived logs support restoration to a chosen recoverable point.

Fundamental of Database Systems module

Advanced Database Systems module

ADB27 | WAL, undo and redo

Write-ahead logging requires the update log needed for recovery to reach stable storage before the corresponding changed data page reaches disk.

To acknowledge a durable commit, force the transaction's required log records, including its commit record, to stable storage.

A buffer can be changed before that log flush; WAL orders durable writes, not every CPU assignment.

Undo restores before-images of incomplete effects, usually traversing that transaction's updates backward.

Redo installs logged after-images when necessary, usually forward.

Setting X to its logged after-value is idempotent; blindly replaying X=X+10 twice is not.

In a simple undo/redo teaching model, undo losers and redo winners.

ARIES has a different redo phase that repeats history before undoing losers.

Fundamental of Database Systems module

Advanced Database Systems module

ADB28 | Steal and force determine recovery work

Steal permits a page with uncommitted changes to be written to disk; hence recovery may need undo.

No-steal prevents those uncommitted effects reaching the persistent database.

Force writes a transaction's updated data pages at commit; no-force permits committed dirty pages to remain buffered, hence recovery may need redo.

For the conventional page/log model: steal/no-force needs undo and redo; steal/force needs undo but no redo; no-steal/no-force needs redo but no undo; no-steal/force needs neither data-page undo nor redo after a system crash.

Logging/atomic commit machinery may still be necessary.

These policies are independent of whether an update overwrites an old page or uses a separate location.

Immediate update alone does not uniquely specify every recovery requirement.

Buffer policy	Force	No-force
Steal	UNDO	UNDO + REDO
No-steal	Neither data-page UNDO nor REDO	REDO

Fundamental of Database Systems module

Advanced Database Systems module

ADB29 | Deferred, immediate, in-place and shadow updates

Deferred update keeps uncommitted changes outside the persistent database; in the usual no-force setting it is NO-UNDO/REDO.

Redo-only and NO-UNDO/REDO mean the same thing, so those cannot be separate unique answers.

Immediate update allows persistent changes before commit and therefore may require undo; it need not flush every write_item immediately.

In-place update overwrites the original location.

Shadow/copy-on-write update writes a new location while preserving the committed version.

Shadow paging keeps an unchanged committed page map and a modified map; commit durably installs a new root/map pointer atomically after required pages are durable.

Before commit, a crash uses the old map.

It is an alternative to conventional log-based recovery, with allocation, fragmentation and concurrency costs; it is not primarily a special action done only during recovery.

Fundamental of Database Systems module

Advanced Database Systems module

ADB30 | Checkpoints and recovery boundaries

A checkpoint records enough durable recovery state to avoid scanning all old history.

In a simple quiescent/checkpoint exercise, specified flushing may make earlier committed work unnecessary to redo.

Active transactions can span the checkpoint, so their relevant earlier updates can still need undo.

Do not assume every transaction before a checkpoint needs no processing.

A fuzzy checkpoint permits ongoing transactions and records active-transaction and dirty-page information without forcing all pages simultaneously.

Recovery may start analysis at the checkpoint but redo from an earlier required log position.

A checkpoint is neither a full backup nor an automatic commit of active transactions.

Without checkpoints or another valid truncation boundary, long-running systems may need expensive log scans; deferred update does not avoid that problem.

Fundamental of Database Systems module

Advanced Database Systems module

ADB31 | ARIES and repeated crashes during recovery

ARIES uses WAL, page log sequence numbers (pageLSN), a transaction table and a dirty-page table.

Analysis reconstructs active/loser transactions and dirty-page information.

Redo repeats history, starting from the relevant earliest dirty-page update; pageLSNs help skip already-installed actions.

Undo reverses loser transactions.

Compensation log records (CLRs) describe undo actions and their continuation information.

They can be redone but are not themselves undone.

This makes recovery restartable: after a second crash, durable undo progress need not be undone again as if no progress occurred.

Absence of checkpoints alone does not imply identical undo/redo lists.

State the recovery algorithm, durable logs/page state and progress before predicting repeated-crash behavior.

Fundamental of Database Systems module

Advanced Database Systems module

Silberschatz, Korth and Sudarshan: Recovery System slides

ADB32 | Execute a small recovery problem

Assume steal/no-force, durable WAL and a simple undo/redo method.

Initially X=100,Y=50.

The durable log contains START T1; UPDATE T1 X 100→120; COMMIT T1; START T2; UPDATE T2 Y 50→70; crash.

T1 is a winner and T2 a loser.

Redo X=120 if needed; undo Y=50 if T2's change reached disk.

Under no-steal, Y never reached the persistent database, so no undo of it is needed.

To solve larger cases, list transactions and durable commit status, identify the checkpoint type and buffer policy, select the algorithm, apply redo/undo in its required order and recompute final values.

Transactions active at failure are not automatically restarted and committed.

Business-level retries require a new controlled attempt; idempotency prevents duplicate effects when the client is unsure whether a commit succeeded.

Fundamental of Database Systems module

Advanced Database Systems module

ADB33 | Distributed, homogeneous and heterogeneous systems

A distributed database contains logically related data across networked sites, managed as a coordinated system.

A homogeneous system uses compatible DBMS technology across sites; a heterogeneous system integrates differing DBMSs, data models, schemas or interfaces.

Different operating systems are possible but do not define the distinction.

Same DBMS does not imply identical local schemas, automatic replication or immediate synchronization.

Heterogeneous design must reconcile naming, types, units, identifiers, query capabilities and transaction semantics.

ODBC and JDBC provide connectivity interfaces; they do not automatically solve semantic differences or create one distributed transaction.

Locality, expansion and replicated availability are potential benefits.

Distribution adds communication, coordination and security costs; a site outage can still make needed data unavailable if no usable replica exists.

Fundamental of Database Systems module

Advanced Database Systems module

ADB34 | Architectures and local autonomy

In client/server architecture, clients submit requests while servers manage shared data, execute operations and enforce supported integrity/transaction rules.

In collaborating-server designs, servers cooperate on subqueries and transactions.

Middleware can translate requests, locate data, coordinate operations and combine results across independent servers.

These describe communication responsibilities, not fragmentation types.

A tightly integrated distributed system can present one global schema and shared control.

A federation integrates autonomous databases through mappings; a loosely coupled multidatabase system may preserve more independence without one complete global schema.

State how much schema, transaction and administrative autonomy local sites retain.

Calling a system heterogeneous does not require all sites to surrender autonomy or to copy every row everywhere.

Fundamental of Database Systems module

Advanced Database Systems module

ADB35 | Transparency: what exactly is hidden?

Location transparency hides which site holds a named object.

Fragmentation transparency hides the existence and reconstruction of fragments: users query the whole logical relation.

Replication transparency hides which copy is used and how copies are maintained.

The broad single-logical-database experience may involve all three.

If users explicitly name a fragment but need not specify its site, fragmentation is visible while location can remain transparent.

Distinguish whether fragment names, storage sites or both are hidden from the user.

Column subsets define vertical fragmentation, not a type of transparency.

Hiding this column split from queries is fragmentation transparency; the split itself is a data-design operation.

Fundamental of Database Systems module

Advanced Database Systems module

ADB36 | Fragmentation: correctness and reconstruction

Horizontal fragmentation selects row subsets by predicates; their union reconstructs the relation.

Check completeness so every original row is represented and disjointness if no overlap is intended.

Define an explicit fragment for NULL/other values when needed.

Vertical fragmentation projects column subsets and repeats a key or row identifier so joining on it reconstructs the original rows without spurious matches.

Hybrid fragmentation combines row and column splits in either suitable order.

Reconstruct by reversing the operations: key joins for corresponding vertical pieces and unions for horizontal pieces.

A full outer join is not universally required.

Partitioning improves locality and parallelism but alone creates no spare copy.

Losing the only site containing a fragment makes that fragment unavailable, even while other sites continue.

Fundamental of Database Systems module

Advanced Database Systems module

ADB37 | Replication policies and their costs

Full replication places the whole database at every relevant site; partial replication copies selected fragments to selected sites; nonredundant allocation stores each fragment once.

Copies improve read locality and can improve availability but require storage and update propagation.

They can also spread mistakes, so replication does not replace independent backups.

Eager/synchronous replication coordinates propagation before the specified commit/acknowledgment point, increasing update latency.

Lazy/asynchronous replication propagates later and may expose stale reads or need conflict resolution.

Snapshot replication transfers a copy at intervals; incremental replication transfers changes.

These dimensions can coexist, so commonly used does not identify one unique policy.

Full replication does not inherently mean every copy is updated simultaneously; its synchronization rule must be stated.

Fundamental of Database Systems module

Advanced Database Systems module

ADB38 | Distributed query plans and semijoin arithmetic

Distributed processing decomposes a logical query, localizes referenced relations to fragments/replicas, chooses global placement/order and optimizes local subplans.

Include bytes moved, message latency, bandwidth and local work.

Apply filters and projections before shipping when equivalent.

A semijoin retains only rows of one input that have a match in the other, returning that input's attributes.

Suppose R at A occupies 100 MB and S at B 20 MB; compute the result at A. Shipping all S costs 20 MB.

A semijoin can send 0.2 MB of distinct R join keys to B, then return 1 MB of matching S rows: 1.2 MB before other costs.

It wins on transferred bytes, but extra messages may dominate on high-latency links.

If nearly all S matches, key shipping may not help.

Fundamental of Database Systems module

Advanced Database Systems module

ADB39 | Distributed atomic commit and Oracle commit points

Two-phase commit (2PC) first asks participants to prepare and durably promise they can commit.

If all vote yes, the coordinator records commit durably and tells participants; otherwise it decides abort.

A prepared participant must retain enough state to obey the final decision.

Coordinator/communication failure can leave it in doubt and block progress until the decision is learned.

Timeout after preparation is not permission to invent a commit or abort.

2PC coordinates atomic outcome; locking/other concurrency protocols provide isolation.

Oracle's COMMIT_POINT_STRENGTH helps choose the commit point site in its distributed transaction mechanism, with higher eligible strength preferred.

This is an Oracle-specific 2PC role rule, not a lock phase or checkpoint setting.

The transaction coordinator coordinates decisions rather than physically storing every site's rows.

Fundamental of Database Systems module

Advanced Database Systems module

Oracle: COMMIT_POINT_STRENGTH

ADB40 | Construct homogeneous and heterogeneous designs

For a bank using one compatible DBMS at three branches, horizontally fragment branch accounts by BranchID; keep each account at its home branch and replicate selected read-mostly reference data.

For cross-branch transfers, use global identifiers, coordinated commit and a stated isolation policy.

Choose replica placement so the required service remains available after the specified failure.

Verify reconstruction, capacity, latency and recovery procedures.

For a merger retaining different branch DBMSs, add mappings for AccountID, currency, time zones and numeric precision, plus middleware/gateways for supported operations.

Decide whether all engines support atomic distributed commits; if not, define an explicit compensating workflow and its intermediate-state limits.

Compare retaining autonomy against coordination and availability requirements.

A good design answers who owns each fact, where copies live, what a failed link permits and how conflicting writes are resolved.

Fundamental of Database Systems module

Advanced Database Systems module

ADB41 | Warehouses, OLTP, OLAP and columnar storage

OLTP handles frequent operational transactions, such as account updates; OLAP scans and summarizes large datasets for analysis.

A warehouse integrates data for historical analysis, often through extraction, transformation and loading.

These are workload tendencies: OLTP can retain history and warehouses can be updated.

A star schema places measurements at a declared grain in a fact table linked to descriptive dimensions, such as time, product and store.

A snowflake normalizes dimension structure further.

Columnar storage groups values by column, reducing unnecessary column reads and aiding compression in scans; row-oriented storage groups each record's fields.

Neither physical layout alone determines whether the logical model is relational.

Indexes and denormalization choices depend on workload.

Fundamental of Database Systems module

Advanced Database Systems module

ADB42 | NoSQL families, wide columns and CAP

Key-value stores retrieve values by key; document stores hold structured documents; wide-column/column-family systems organize sparse rows and related column families; graph systems model nodes and edges for relationship traversal.

Wide-column systems such as Cassandra are not simply interchangeable with analytic columnar relational engines.

Flexible schema does not mean no schema rules, and NoSQL does not imply no transactions.

CAP states that under a network partition, a distributed service cannot guarantee both linearizable consistency and availability for every request at nonfailed nodes.

It is not a universal menu saying casually pick any two forever.

Systems choose failure behavior and may vary it by operation.

CAP consistency is a single-copy visibility/order property, distinct from ACID consistency's preservation of application invariants.

Gilbert and Lynch: Perspectives on the CAP Theorem

ADB43 | Data sources and association rules

Structured data follows explicit fields, such as a registrar's relational tables.

Unstructured data includes free text, images and audio; web/search collections can include these alongside structured content.

A DBMS can also store text or BLOBs, so a storage product does not prove all its contents are structured.

An association rule $X \Rightarrow Y$ describes co-occurrence.

Support=count(X and Y)/N; confidence=count(X and Y)/count(X)=P(Y|X).

If 75 of 100 shoe purchases include socks, confidence(shoes⇒socks)=0.75.

This does not prove causation or specify support without total transaction count.

Clustering groups similar cases without target labels; classification predicts categories and regression predicts numeric outcomes.

Purchasing co-occurrence is association mining even when its result informs store layout.

University of Tennessee: Association rules

ADB44 | Data-mining workflow and evaluation

CRISP-DM starts with business understanding: define the decision and success criteria.

Then understand data, prepare it, model, evaluate and deploy; the process is iterative.

Cleaning, integration and feature selection occur during preparation, not before the business problem exists.

Within modeling, split data appropriately, train on training data, tune using validation and evaluate once on a held-out test set before deployment.

Testing and evaluation are sometimes overlapping labels rather than universally distinct sequential phases.

If an MCQ asks training→testing→evaluation→deployment, state that convention; another valid workflow can evaluate during model selection before final testing.

Deployment includes monitoring and may trigger another preparation/modeling cycle.

Do not leak future or test outcomes into training.

IBM SPSS Modeler: CRISP-DM View

ADB45 | Compare database mechanisms under explicit assumptions

Database mechanisms can overlap: ARIES and deferred update can both be log-based; restart recovery can use checkpoints, rollback and redo; NO-UNDO/REDO equals redo-only.

Questions using best, important, common or faster need a workload and comparison criterion.

Read the literal scope rather than treating one familiar term as automatically unique.

State the minimum missing information: exact DBMS/version for vendor behavior; OID/type convention for object constructors; requester/holder ages for waiting; durable log, page state and policy for recovery; fragment definitions and network cost for distribution.

Evaluate each description separately when several database mechanisms apply.

Use the stated assumptions to explain which database mechanisms apply to each question.

Fundamental of Database Systems module

Advanced Database Systems module

ADB-S01 | Query representations and relational operators

A relation is represented by a table of tuples. Selection filters its rows by a predicate; projection retains chosen columns. A Cartesian product constructs every pair of input tuples. A matching join combines tuples satisfying its predicate, and can be represented as a selection over a product. Sorting orders results, while grouping collects rows sharing grouping values; these are different operations.

A query tree corresponds to a relational-algebra expression: base relations appear at leaves and operators at internal nodes. A query graph represents the predicate relationships of a relational-calculus expression without itself prescribing an operator execution order. An optimizer can transform the representation and choose access paths; the representation is not the query result.

Advanced Database Systems module · page 24

Advanced Database Systems module · page 34

PostgreSQL documentation - Table Expressions

PostgreSQL documentation - Sorting Rows

ADB-S02 | Transaction boundaries and savepoints

In SQL transaction-block notation, BEGIN starts a block, COMMIT completes it successfully, and ROLLBACK cancels its uncommitted changes. Committing requests a durable transaction outcome; it is distinct from merely finishing the last read or write operation.

SAVEPOINT marks an intermediate position inside the current transaction. ROLLBACK TO discards changes made after that position while retaining earlier work in the transaction. A savepoint is not an independent durable commit, and rolling back to it does not itself commit the retained changes.

PostgreSQL documentation - Transactions

ADB-S03 | Availability and evidence against false denial

Availability concerns keeping services and resources accessible to their authorized users. Non-repudiation concerns evidence that an event or action occurred and who originated it, so a participant cannot falsely deny it. Neither property specifies which particular user has permission to change a database row; that is the separate authorization decision.

Computer Security module · page 4

ADB-S04 | Elementary binary locks and shared-lock compatibility

An elementary binary lock has two states: free and held. A request on a free item sets the lock to held; a request on an already-held item waits until release. That simple model does not distinguish shared readers from a writer. Shared/exclusive locking adds modes so multiple shared readers can coexist while incompatible exclusive access is excluded.

Advanced Database Systems module · page 83

ADB-S05 | Classify a failure by its stated cause

A software failure originates in erroneous program behavior, such as a logical programming defect, invalid operation or operating-system fault. A hardware failure originates in malfunctioning physical equipment; loss of disk contents is media failure. An accidental user deletion is human error even if the software executes the requested command correctly. Physical catastrophes such as fire can also cause failure. The observed symptom of a crash does not alone identify which cause occurred.

Advanced Database Systems module · page 98

Object Oriented Programming

Complete Study Notes

OOP01 | Objects, classes and responsibilities

An object has identity, state and behavior.

State is stored in fields; behavior is provided by methods.

A Rectangle object may hold width and height and answer getArea() or getPerimeter().

Two objects can have equal fields and still be distinct objects.

A class defines their common structure and behavior; an object is a particular instance.

Choose classes around responsibilities: an account controls its own balance, a transaction records a payment, and a service coordinates them.

Objects send messages through method calls such as obj.add().

They can delegate work to other objects; saying that objects cannot manage their own state is misleading.

Procedural and object oriented designs both support reuse; global shared data is not compulsory in procedural programming.

Object Oriented Programming module

OOP02 | Four principles and the difference between hiding and abstraction

Encapsulation groups related state and operations and controls access.

Information hiding keeps implementation decisions behind a stable interface.

Abstraction presents the essential operations while omitting details irrelevant to the caller.

These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Inheritance forms a subtype relationship and may reuse implementation.

Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Prefer a precise mechanism over a vague slogan: wrapping fields and methods points to encapsulation; showing what an operation does while hiding how points to abstraction.

Object Oriented Programming module

OOP03 | Create objects and distinguish references from instances

Student s; declares a reference variable; it creates no Student object.

Student s=new Student(); allocates and initializes an instance.

Member access uses s.name or s.study(), subject to visibility.

Assignment t=s copies a reference, so both can refer to the same object; changing that object through t is observable through s.

Java passes all arguments by value.

Passing a reference copies it: a method may modify the shared object, but assigning its parameter to a new object does not reassign the caller's reference.

The runtime may reclaim unreachable objects; collection timing is unspecified.

Reachable object cycles can matter for retention, but an unreachable cycle can be collected.

Object Oriented Programming module

OOP04 | UML class and instance notation

A UML class box normally contains name, attributes and operations.

Visibility uses + public, - private and # protected.

A field can be written name:String and an operation getName():String.

A constructor Student(String,String,String,int) requires three strings and an integer, so an age argument "25" has the wrong type.

An instance is conventionally labeled with an underlined instanceName:ClassName, such as sara:Student.

Class notation describes a type; instance notation describes one particular object and may show its slot values.

Underlining static class features has a different purpose; read the whole notation rather than assuming every underline means the same thing.

Object Oriented Programming module

OOP05 | Variables, defaults, scope and initialization

The introductory three-way grouping is local, instance and class/static variables.

Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers.

Fields occur in a class outside method bodies; non-static fields belong to each instance, while static fields belong to the class.

Declaration location alone does not distinguish instance and static fields.

Java fields and array elements receive defaults: numeric zero, char '\u0000', boolean false and references null.

A local int t; followed by printing t is a compilation error.

Initializers such as int a=10,b=10; assign initial values.

A final local is permitted; final is not an access modifier.

Object Oriented Programming module

OOP06 | Primitive types, literals and conversions

Java has byte, short, int, long, float, double, char and boolean.

Signed integer widths are 8,16,32,64 bits respectively. byte ranges from -128 to 127; short from -32768 to 32767. char is an unsigned 16-bit UTF-16 code unit and can hold a digit, letter or symbol, not a whole String. enum and String are reference types.

A compatible widening conversion often needs no cast; narrowing usually needs an explicit cast and may lose information.

Widening an int to float can lose precision, so “widening never loses information” is false.

Arithmetic promotes char, byte and short to int in many expressions. boolean does not convert to or from numeric truth values; if(0) and for(...;0;...) are invalid.

Object Oriented Programming module

OOP07 | Static members, static blocks and final

static gives one field per loaded class, shared by its instances.

If count starts at zero and every constructor increments it, three creations leave count=3.

Prefer ClassName.count and ClassName.utility().

A static method has no current instance, but can still use instance members through an explicit object reference.

A static initializer runs during class initialization, before the class’s first relevant active use, not simply whenever a file is loaded. final prevents reassignment of a variable, overriding of a method or extension of a class.

A final reference may still refer to a mutable object.

A shared mathematical PI is a good static final candidate; a student name normally belongs to an instance.

A calculation using only parameters may be static.

Object Oriented Programming module

OOP08 | Access control and package boundaries

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Top-level classes/interfaces may be public or package-private, not protected/private.

Member types have additional visibility possibilities.

Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

A class’s own main method can access private fields of an instance of that class.

Object Oriented Programming module

JLS 21: protected access

OOP09 | Constructors and missing default constructors

A constructor has the class’s exact name and no return type, not even void.

It initializes the new object and can be overloaded or private.

A private constructor can support a factory or controlled singleton creation; it does not mean the class can never create an object internally.

If a class declares no constructor, Java supplies a default constructor.

If Circle(double r) is declared, new Circle() fails unless a no-argument constructor is also declared.

Constructors are not inherited or overridden.

They may contain return; but cannot return an expression.

Fields get their default values before ordinary constructor initialization, rather than receiving defaults only because a default constructor was generated.

Object Oriented Programming module

OOP10 | Constructor chaining and initialization order

Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor.

A constructor cannot explicitly invoke both.

If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Trace parent construction first, then subclass field initializers and instance blocks in order, then the relevant subclass constructor body.

With consistent class and constructor names and this(5) chaining in the FinalExam example, the parent sets num1=4, the subclass initializes num=3, this(5) sets num=25, and the no-arg body sets num1=num, leaving 25.

Java 25 permits restricted statements before explicit constructor invocation; the older first-statement rule must be version qualified.

Object Oriented Programming module

Java 25: flexible constructor bodies

OOP11 | Methods, getters and setters

A method has modifiers, return type, name, parameter list and body: public int getAge(){return age;}.

A void method returns no value.

Parameters name the inputs in the declaration; arguments supply values at a call.

An instance method can be invoked as object.method(), or with an implicit this inside suitable instance context.

An accessor/getter returns information; a mutator/setter changes state.

Encapsulation does not require blindly generating setters for every field: validate changes and expose only useful operations.

For a bank account, deposit must reject negative amounts.

A private field cannot normally be accessed directly by a separately declared subclass, even with this.field; use a permitted parent operation.

Object Oriented Programming module

OOP12 | Method overloading and compile-time selection

Overloaded methods share a name but have different parameter types, counts or type order.

Return type alone is insufficient in Java and C++.

Overloads can also arise through inherited methods; “same class only” is a teaching simplification. f(int), f(double) and f(int,int) are distinct, but changing a parameter name does not create an overload.

The compiler selects the applicable signature from the declared argument types and conversions. int arguments usually choose an exact int overload rather than a wider double overload.

A return type is not part of an ordinary Java method signature.

Distinguish a different signature from overriding an inherited signature.

Object Oriented Programming module

OOP13 | Inheritance and useful relationships

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

A class extends one direct class and can implement multiple interfaces.

Single, multilevel and hierarchical inheritance mean one parent, a chain, and several children of one parent.

Java does not permit class Dog extends Animal,Mammal.

Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal.

For shared Pet.name and speak(), put common state/contract in Pet, specialized fetch() in Dog, and appropriate speak() implementations in Dog and Cat.

Composition represents has-a, such as Car containing Engine.

Reuse alone does not justify an invalid is-a relationship.

Object Oriented Programming module

OOP14 | Overriding and restrictions

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly. @Override requests compiler checking; Java has no override keyword.

Constructors, final methods, static methods and private parent methods are not overridden.

A same-signature child method matching a private parent method is an independent declaration, not runtime overriding and not ordinary static-method hiding.

An abstract inherited method is implemented by an overriding method in a concrete subclass.

Merely reusing a name in two unrelated classes does not establish overriding.

Object Oriented Programming module

OOP15 | Dynamic dispatch: declared type versus actual type

For Animal a=new Dog(), Animal is the declared type and Dog the actual type.

The declared type determines which member operations compile.

If fetch() exists only in Dog, a.fetch() fails compilation.

For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

To solve dispatch questions: check the declaration and visibility, determine the compile-time signature, then identify the object's runtime class and its most specific override.

An array Animal[]{new Dog(),new Cat()} can call speak() on each using one loop.

New subclasses can supply behavior without rewriting this generic loop, which is the practical meaning of extensibility.

Object Oriented Programming module

OOP16 | this, super, fields and static hiding

Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation.

It does not create a second parent object.

An unqualified instance call may dispatch dynamically; an explicit super call is different.

Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override.

With Animal a=new Dog(), a.staticMethod() resolves using Animal's compile-time type.

Prefer Animal.staticMethod() for clarity.

Java this and super are unavailable as current-instance references in static context.

Object Oriented Programming module

OOP17 | C++ this and language-specific assumptions

In ordinary C++17 non-static member functions, this is a pointer to the object on which the function is called; this->field selects a member.

It is often explained as an implicit object argument to a member function call.

It is not an argument passed to member variables, and a static member function has no this pointer.

C++ runtime polymorphism requires a virtual function when calling through a base pointer/reference.

C++ may inherit multiple classes; Java permits multiple interface types instead.

C++ object-value assignment can copy an object, whereas assigning Java class variables copies references.

Never transfer Java dispatch or memory-management assumptions to a C++ question without checking the code.

C++ draft: this pointer

Object Oriented Programming module

OOP18 | Abstract classes and concrete implementations

An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods.

It need not contain an abstract method.

An abstract method has no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

If parent declares abstract String welcome(), a String welcome() implementation is required; int welcome() is incompatible, and welcome(int) merely adds an overload.

A valid class can contain both the required implementation and an unrelated overload.

Object Oriented Programming module

OOP19 | Interfaces: contracts and modern Java

interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem().

A class can extend a superclass and implement zero or more interfaces.

An interface can extend several interfaces.

No interface instance fields or constructors exist; fields are implicitly public static final.

A method declaration without private, default or static is implicitly public abstract.

Modern interfaces may also have default/static method bodies, and Java 9+ private helper methods.

Not every interface method must lack a body in modern Java.

Implementations must satisfy all inherited abstract contracts; they may inherit suitable public implementations.

Interfaces cannot be instantiated directly, but references may hold implementing objects, anonymous implementations or suitable functional-interface lambdas.

Object Oriented Programming module

JLS 21: interface methods and conflicts

OOP20 | Conflicting interface methods and abstract class choice

Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

A concrete inherited class method takes precedence over an interface default.

An inherited abstract class declaration can require a concrete subclass to implement the operation rather than silently adopting a default.

Unrelated interfaces providing conflicting defaults require an explicit resolution; interfaces do not remove every diamond ambiguity.

If two compatible interfaces merely declare the same abstract signature, one public implementation can satisfy both.

If return types or other contracts conflict, additional declarations may still be impossible.

The answer to “same method in abstract class and interface” depends on whether each is abstract or implemented and whether their signatures are compatible.

Object Oriented Programming module

JLS 21: interface methods and conflicts

OOP21 | Casts and instanceof

Upcasting a subclass reference to a superclass is implicit and does not change the object.

A downcast may pass compile-time type checks yet fail at runtime.

Car a=new Car(); Ferrari f=(Ferrari)a; throws ClassCastException before subsequent drive() or brake() calls.

It does not turn a plain Car into a Ferrari.

Use if(a instanceof Ferrari) before narrowing when the dynamic type is uncertain. null instanceof Ferrari is false; casting null yields null, although dereferencing it fails.

For Object obj="Hello", (String)obj succeeds and (Integer)obj throws ClassCastException.

A literal 'Hello' is invalid Java syntax; use double quotes for a string literal before tracing the program.

Object Oriented Programming module

OOP22 | Identity, equals, toString and immutable strings

For object references, == tests identity; equals invokes the class’s equality policy.

Object.equals uses identity unless overridden; String.equals compares character content.

Equal objects must have equal hashCode values when that contract is implemented. toString gives a textual representation, not an equality test.

Java String objects are immutable. text += " World" assigns a reference to a new concatenated result and can print Hello World without mutating the original String object. final class String prevents subclassing; final alone does not make arbitrary objects immutable. charAt(index) returns a char; length() counts UTF-16 units; substring(begin,end) excludes end.

Character.toUpperCase(ch) converts a character, and matches(String regex) takes a regular-expression parameter named regex.

Object Oriented Programming module

OOP23 | Expressions: numbers, bits and text output

Java evaluates operands left to right while respecting precedence.

With n=6,m=15,p=3, n-=m then n*=p leaves -27. 6-2+10%4+7 is 13.

In 8-2*5*(3+9/3)+6/3, the result is -50.

Integer 6/4 is 1; double 6.0/4 is 1.5.

Unlike C++, Java % also accepts floating-point operands.

For bitwise values, write binary columns: 11&9=9; 9^3=10; 10|12=14. && and || short circuit Boolean operands; & and | can also operate on Boolean values without short circuiting.

Printing Boolean true gives lowercase true, not 1 or uppercase TRUE.

Object Oriented Programming module

OOP24 | Increment order and the character-space trap

Starting x=5, printing x++ then ++x gives 5 then 7.

For int y=++x*x--, Java uses 6*6=36 and leaves x=5.

System.out.println(x + ' ' + y) then adds char space's numeric value 32, so it prints 73.

To print separated values, use x + " " + y instead.

Once a String operand is encountered, + concatenates in left-to-right groups: 1+2+"x" gives "3x", while "x"+1+2 gives "x12".

When x and y are numeric, x + ' ' + y performs numeric addition; x + " " + y concatenates a separating space.

Use the expression's operand types to determine the result.

JLS 21: expression order and string addition

OOP25 | Control flow traces and exact printing

while and for pre-test and may execute zero times; do-while post-tests and executes once. break exits the loop; continue skips its remaining body and then runs a for update.

Traditional colon-style switch labels fall through without break.

Choice 1 printing Hello then case 2 printing you and breaking is followed by Goodbye outside the switch; default is skipped.

Count executions independently from printing.

For i=0..4, if(i%2==0) print(i+" "); followed unconditionally by print(i+1); produces "0 122 344 5".

A loop summing j=1..10 yields 55.

In if(a)... else if(a&&b)..., the second condition can never be true on a path reaching it because a is already false.

Object Oriented Programming module

OOP26 | Arrays: creation, bounds and ragged rows

int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer. int a[],b; makes only a an array; int[] c,d; makes both arrays.

Array objects have fixed length after creation, although the reference may be reassigned.

Elements have a declared common component type, which may be a superclass/interface.

Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Java two-dimensional arrays are arrays of arrays and may be ragged: {{12,-9,8},{7,14},{-32,-1,0}} has row lengths 3,2,3.

Java does not expose a portable physical contiguity guarantee, so a language-level answer should not rely on raw addresses.

Object Oriented Programming module

OOP27 | Array algorithms, sort and copy APIs

For int[] a=new int[4], setting a[i]=i*3 and accumulating yields 0+3+6+9=18.

To average an array, sum every element and divide by its length using floating arithmetic; handle zero length separately.

Floating representations and printing can differ from a rounded mathematical result.

For the six-value example, the mathematical average is 98.8/6, approximately 16.46666667; the exact printed string also depends on floating representation and formatting.

java.util.Arrays.sort(a) sorts in place.

System.arraycopy(src,srcPos,dst,dstPos,length) copies into an existing array; its exact spelling uses lowercase c. Arrays.copyOf(a,n) creates a copy. a=b merely aliases the same array.

Random access is the reliable array advantage; size is bounded by implementation/resources, not unlimited.

Object Oriented Programming module

OOP28 | Packages, imports and command-line tools

package p; precedes imports; imports precede top-level type declarations. import p.*; imports accessible types from p as needed, not types in p.sub automatically.

Packages organize names and control access; hierarchical spelling does not grant special access between a package and its subpackages. java.lang is implicitly imported; java.util supplies utilities; java.io supplies I/O; java.awt includes Button and Checkbox.

Ordinary compilation uses javac Test.java, producing bytecode, then java Test launches the class.

Modern Java can also launch a source file using java Test.java; distinguish source-file launch from running a compiled class.

For ordinary public top-level types, filename and type name match.

Capitalized class names and lowercase method names are conventions, not universal syntax requirements.

Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Object Oriented Programming module

OOP29 | Exception hierarchy and checked obligations

Throwable is the root of throwable values, with Exception and Error branches.

RuntimeException and Error subclasses are unchecked; other throwable classes are checked.

IOException, FileNotFoundException and SQLException are standard checked examples.

ArithmeticException, NullPointerException, ClassCastException and array bounds exceptions are unchecked.

A missing semicolon or undeclared variable is a compile-time source error, not a checked exception.

Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime.

A custom exception can extend Exception or RuntimeException.

Error normally signals serious conditions for which routine recovery is inappropriate; it is not technically impossible to catch an Error.

Object Oriented Programming module

JLS 21: checked and unchecked exceptions

OOP30 | throw, throws and catch selection

throw new IOException("message") actively throws a value; throws IOException in a signature declares a possible checked exception.

A caller must handle or propagate checked obligations. try encloses guarded operations; a matching catch handles the thrown type or a subtype.

Execution skips the remainder of the try when a throw occurs.

Catch blocks are considered in source order.

Place FileNotFoundException before IOException and IOException before Exception; a general catch before a subtype catch makes that later handler unreachable and prevents compilation.

If no catch matches, the exception propagates outward and an uncaught exception terminates its thread.

Multiple catch blocks are allowed; catch(IOException | SQLException e) handles alternatives, which must not be related by subclassing.

Object Oriented Programming module

JLS 21: checked and unchecked exceptions

OOP31 | finally, cleanup and diagnostic information

finally normally executes after try and any selected catch, including when leaving by return or propagating an exception.

It is not guaranteed if the process/JVM exits or crashes, or control never leaves the try.

A return expression is evaluated before finally; a finally return overrides the pending return and can suppress an exception, so avoid that pattern.

try-with-resources closes AutoCloseable resources even on exceptional exit. getMessage returns the message, toString normally includes type and message, and printStackTrace shows exception information and recorded call frames for diagnosis.

Tests should cover expected success, caught failures and propagation.

Exception handling does not replace validation or correct an algorithm’s logic automatically.

Object Oriented Programming module

JLS 21: try and finally

OOP32 | Streams, GUI events and JDBC practice

InputStream/OutputStream read and write bytes; Reader/Writer handle text characters through an encoding.

Close resources deterministically.

A read returning -1 signals end-of-stream.

A GUI event source notifies registered listeners; a button can call an ActionListener’s actionPerformed method.

In Swing, update components on the event dispatch thread and keep handlers short so the interface remains responsive.

Object Oriented Programming module

Oracle: event dispatch thread

OOP33 | JDBC flow from user action to database

A JDBC operation obtains a Connection, prepares SQL, binds values, executes and processes results, then closes resources.

For PreparedStatement ps=con.prepareStatement("SELECT name FROM Student WHERE id=?"), ps.setInt(1,id) binds the first parameter. executeQuery returns a ResultSet; rs.next() moves to each row before reading columns. executeUpdate returns an affected-row count for updates.

Parameter binding keeps values separate from SQL syntax.

Catch or declare SQLException.

When several writes form one operation, disable auto-commit, commit on success and roll back on failure.

A GUI handler should delegate a slow database operation away from its UI thread, then display the result through the appropriate UI mechanism.

Oracle: prepared JDBC statements

OOP34 | Legacy applets and lifecycle questions

In the historical applet model, Java code was embedded in an HTML document and executed by a compatible JVM host/plugin or applet viewer. init initializes once; start follows initialization and resumes activity; stop suspends activity when the host deactivates it; destroy performs final cleanup before unloading. paint draws content and is not a constructor.

Study applets as historical Java concepts rather than modern browser deployment tools. destroy is not defined exclusively as “browser shuts down”: the host may destroy an applet on unloading, and abrupt process termination does not guarantee cleanup.

A JVM alone is also not the complete browser embedding environment.

Oracle: applet lifecycle callbacks

OOP35 | Android Activity, Intent, manifest and layouts

An Android Activity is a component providing an entry point for user interaction, often corresponding to a screen.

An Intent describes a requested action or explicit target; extras carry values between activities.

For example, putExtra("id",17) attaches an ID for the destination to read.

A content provider shares structured data, while a broadcast receiver handles broadcast messages; neither is the ordinary mechanism for starting an activity with parameters.

Android: Intents and extras

Android: Activities

OOP36 | Android app declarations and packaging

AndroidManifest.xml declares essential app metadata such as components, permissions and intent filters.

It is not executable code and does not contain all source or layout details.

Traditional view-based layout containers such as LinearLayout and RelativeLayout derive from android.view.ViewGroup, which contains child views.

An .apk is an Android package archive used for installation and runtime contents. “Android Package Kit” is a common exam expansion; Android’s documentation calls it an Android package.

A manifest describes the package, while layout resources describe views.

Use these distinctions to compare Android application structure with Java language concepts.

Android: manifest

Android: ViewGroup

Android: packages

OOP37 | Validate Java syntax before tracing

Check exact Java spelling and syntax: extends differs from Extends, println differs from printIn/printin, constructor names must match class names, and braces and operators such as <= and == affect validity and behavior.

Java is case sensitive: an invalid program may not compile even when a possible trace seems clear.

For a valid Java loop with i=2,4,6,8,10 that prints i*i only when i%4==0, the printed values are 16 and 64.

A valid implementation can define public String welcome() and additionally overload welcome(int).

Identify an implementation by its complete declarations and method bodies.

Check the required signature before selecting an implementation.

A rounded mathematical mean is not necessarily the exact Java printed output.

Object Oriented Programming module

OOP38 | Apply Java rules under explicit assumptions

Several Java statements can be valid at once: a static constant and a parameter-only utility may both be suitable static members; multiple array declarations/initializers can be legal; an abstract class may contain a complete interface implementation.

Encapsulation and abstraction overlap, so choose only when the requested emphasis is specific.

An interface supports multiple type inheritance and abstraction. “All methods are public abstract” applies only under a restricted abstract-declaration or legacy interpretation. Java protected also grants access within the same package, so it cannot enforce “only subclasses.”

No keyword implements overriding: it is recognized from compatible declarations, and @Override is an annotation.

For C++ this, ordinary non-static member functions have an implicit object; variables do not receive a hidden argument.

Object Oriented Programming module

OOP39 | Language classifications and portability

Under the conventional introductory generation taxonomy, general-purpose high-level languages such as C++, Java, Python and Perl are third generation.

The taxonomy is separate from object orientation and from compilation versus interpretation.

Java source normally compiles to JVM bytecode, and a compatible JVM plus libraries supports portability across systems.

Portability still depends on compatible APIs, versions and environment assumptions; it is not an unconditional guarantee for any program on any device.

Java supports objects and classes but also primitive values. “Difficult to learn” is a subjective judgment rather than a technical language property.

Syntax conventions, bytecode portability and runtime implementation must be distinguished.

Object Oriented Programming module

University of Rhode Island: conventional language generation taxonomy

OOP-S01 | Identifiers, literals and source comments

An identifier is a programmer-chosen name for an entity such as a class, method or variable. A reserved keyword has a language-defined role and cannot be used as an ordinary identifier. A literal writes a value directly, such as an integer, floating-point number, quoted character, quoted string or Boolean value.

A single-line Java comment begins with // and continues to the end of that line. A block comment begins with /* and ends with */. A documentation comment begins with /** and ends with */ and can be processed by Javadoc. Comment text does not execute as program statements.

For ordinary ASCII Java identifiers, the first character is a letter, underscore or dollar sign; later characters may also be digits. A reserved keyword is not an ordinary identifier, and a lone underscore is not a valid identifier in Java 17.

Object Oriented Programming module · page 4

Java SE 17 JLS - Identifiers

OOP-S02 | Java 17 floating-point evaluation and printed doubles

Java 17 float uses IEEE 754 binary32, and double uses binary64. Successive multiplications group from left to right. Each floating-point operation rounds to its representable result, using nearest-value rounding with ties to even; regrouping a product can change its result.

PrintStream.print(double) converts the stored value through String.valueOf and Double.toString. For finite nonzero magnitudes from 0.001 inclusive to 10000000 exclusive, the standard decimal form uses at least one fractional digit and only the further digits needed to distinguish the value from adjacent doubles. println(double) also adds a line terminator. This is not fixed-decimal-place formatting.

A print call emits its value without inserting a separator or ending the line. A println call emits its value followed by a line terminator. Spaces must come from the supplied text or explicit output operations; adjacent calls do not automatically separate their values.

Java SE 17 JLS - Floating-point expressions and multiplication

Java SE 17 - Double.toString(double)

Java SE 17 - PrintStream.print(double)

OOP-S03 | Expression results, assignment and control headers

Java relational comparisons (<, <=, >, >=) and equality tests (==, !=) produce primitive boolean truth values. Their result is not an integer truth code or a value of the numeric operand type.

The conditional expression condition ? whenTrue : whenFalse requires a boolean or Boolean condition. It evaluates only the selected alternative. An integer or floating-point first operand is a compile-time error.

Assignments group right to left: a=b=c assigns c to b, then the resulting value to a. A later assignment overwrites an earlier value. Compound assignments update the left variable using the indicated operation.

Prefix ++ and -- update a variable before supplying its value; postfix forms supply the old value before the update. Multiplication, division and remainder bind before addition and subtraction; same-level binary arithmetic groups left to right. Integer division truncates toward zero. Unary plus preserves its numeric operand’s value.

A while header is while(condition). A basic for header is for(initialization; condition; update). A do-while statement places its condition after the body. An if selects its true branch or, when present, its else branch; only the selected branch executes.

For Boolean values, ! reverses true and false, && is true only when both operands are true, and || is true when at least one is true. Short-circuit evaluation skips an unnecessary right operand. For each binary column, integral & produces 1 only for two 1 bits, | produces 1 for either 1 bit, and ^ produces 1 for differing bits.

An unlabeled break leaves the innermost enclosing loop or switch. continue moves to the next iteration step, while return leaves the current method and may supply a return value. These transfers have different destinations; return is not merely a loop exit.

Java SE 17 JLS - Operators and expressions

Java SE 17 JLS - Statements and loop headers

Object Oriented Programming module · page 7

Oracle Java tutorial · Boolean and conditional operators

Oracle Java tutorial · Bitwise operators

Oracle Java tutorial · Branching statements

OOP-S04 | Core package membership and text values

The base Exception class is java.lang.Exception. It does not belong to java.io merely because some subclasses represent I/O failures. Package membership and position in an inheritance hierarchy are separate facts.

Calendar and Date are classes in java.util; Collection is an interface in that package. java.util also contains Arrays and Scanner. java.net supplies networking facilities; java.awt supplies the Button and Checkbox UI controls described in the package overview.

A String represents a sequence of UTF-16 code units. A single char represents one code unit rather than an entire String. String objects are immutable; changing a reference to another String does not modify the original text value.

A double-quoted string literal denotes a String object. new String(existingString) creates a String with the same text; new String(charArray) copies the array's characters into an immutable String. A single-quoted literal denotes one char, so several characters enclosed together in single quotes cannot supply a String.

Character.toUpperCase(ch) supplies the uppercase equivalent where one exists; Character.toLowerCase(ch) supplies the lowercase equivalent. Character.isDigit(ch) tests whether a character is a digit and returns boolean. Character.toString(ch) supplies a one-character String rather than converting letter case.

Java SE 17 - Exception

Java SE 17 - java.util package

Java SE 17 - java.net package

Java SE 17 - String

Oracle Java 17 Character API

OOP-S05 | Inheritance boundaries and whole-part relationships

In Java's class-declaration terminology, constructors are not members and are not inherited. A subclass can invoke superclass construction; this does not turn the superclass constructor into an inherited subclass method.

Aggregation models one instance grouping other instances as a whole with parts. It is an object-relationship concept, not a method-dispatch rule. Inheritance instead relates a specialized class to a general class; overridden operations can then provide polymorphic behavior.

Object Oriented Programming module · page 16

Object Oriented Programming module · page 15

OMG UML 2.5.1 - Property aggregation semantics

OOP-S06 | Introductory Java characteristics

Java is commonly characterized as object oriented, platform independent and simple. Its architecture-neutral bytecode is executed by a compatible platform-specific JVM. “Simple” is a teaching characterization relative to language design, not a promise that every learner finds every Java program easy.

Object Oriented Programming module · page 2

Computer Organization and Architecture

Complete Study Notes

COA01 | Architecture, organization and system structure

Architecture describes programmer-visible behavior: instructions, registers, data types, address spaces and I/O conventions.

Organization is the implementation: control signals, internal buses, pipelines and memory technology.

Different organizations can implement the same architecture; neither definition imposes a maximum lifetime.

Microarchitecture names the internal processor design.

CPU, main memory, I/O and their interconnection are the major structural parts.

The CPU contains an ALU, control unit and registers; a microcontroller is a complete integrated control system, not another mandatory CPU subunit.

A motherboard physically connects components.

Gates are fundamental logic building blocks; distinguish that level from CPU, memory, disk and motherboard components.

Computer Organization and Architecture module

COA02 | Stored programs and software roles

In the textbook Von Neumann model, instructions and data share an addressed memory system; the CPU normally advances sequentially unless control flow changes.

Harvard separates instruction and data memories or paths.

Modern systems may combine a unified address-space view with separate instruction/data caches, so “architecture used nowadays” is not an exclusive choice.

Separate program/data regions within one memory do not make a Harvard machine.

An operating system manages resources and provides services.

Applications solve user tasks.

A compiler translates source into target code; assembly is a symbolic form of machine instructions.

JIT compilation happens during program execution, often translating intermediate code.

Translation before execution alone does not distinguish JIT from an ahead-of-time compiler.

Machine-language programming depends on the register set, supported types and memory/addressing structure.

Computer Organization and Architecture module

COA03 | Gates, truth tables and symbols

AND is 1 only when all inputs are 1; OR is 1 if any input is 1.

NOT complements a bit.

NAND complements AND; NOR complements OR.

For two inputs XOR is 1 when inputs differ, and XNOR when they agree.

An enabled buffer copies its input.

A bubble at a gate output indicates inversion: an AND-shaped symbol with a bubble is NAND.

With n binary inputs, enumerate 2^n truth-table rows.

Evaluate complements, products and then sums.

A plus sign means Boolean OR, not integer addition.

Example

Worked microexample

- F=x+y'z at x=0,y=1,z=1: y'=0, so F=0 OR (0 AND 1)=0. Four independent inputs give 16 rows.

Computer Organization and Architecture module

COA04 | Boolean reduction and universal gates

Useful identities are A+0=A, A·1=A, A+A=A, AA=A, A+A'=1, AA'=0, A+AB=A and A(A+B)=A. De Morgan gives (AB)'=A'+B' and (A+B)'=A'B'.

Check a proposed simplification with a truth table if uncertain.

NAND and NOR are each universal: feeding the same input to both inputs constructs NOT; combine complemented gates to construct AND or OR.

For Karnaugh maps, use Gray-code order, group adjacent 1s in rectangles of powers of two, allow wraparound and overlap, cover every required 1, and remove variables changing within a group.

Do not group diagonally or include a required 0.

Example

Worked microexample

- AB+AB'=A(B+B')=A. A NAND A=A'; NAND followed by inversion gives AND.

Computer Organization and Architecture module

COA05 | Combinational versus sequential circuits

A combinational output depends on present inputs after propagation delay.

Adders, multiplexers, demultiplexers, encoders and decoders have no stored logical state.

A sequential circuit depends on stored state as well as inputs; registers, counters and flip-flops are examples.

This is the defensible criterion when comparing a shift register against decoder, encoder and multiplexer.

A flip-flop stores one bit while powered.

State retention during operation does not mean nonvolatile storage.

Ordinary flip-flops and SRAM lose state when power is removed.

Clocked sequential circuits update at designated clock events; combinational logic does not inherently require a clock.

Computer Organization and Architecture module

COA06 | Flip-flop and counter state traces

For an active-high SR latch, S=1,R=0 sets, S=0,R=1 resets, and S=R=0 holds; S=R=1 is forbidden for the NOR-latch model.

Active-low NAND-latch conventions differ.

A D flip-flop copies D at its active edge.

JK holds for 00, resets for 01, sets for 10 and toggles for 11.

T toggles when T=1 and holds when T=0.

Trace a sequential circuit one clock at a time from the given initial state.

A k-bit binary counter has 2^k states and wraps modulo 2^k unless extra control changes the sequence.

Ripple counters propagate changes asynchronously; synchronous counters share a clock.

Example

Worked microexample

- A 3-bit counter at 110 advances to 111, then 000. A T flip-flop starting at 0 with T inputs 1,0,1 gives states 1,1,0.

Computer Organization and Architecture module

COA07 | Adders, selectors and code converters

A half-adder has sum A XOR B and carry AB.

A full-adder includes carry-in: sum=A XOR B XOR Cin and carry=AB+Cin(A XOR B).

Cascading full-adders propagates carry across a word.

A 2^k-to-1 multiplexer uses k select bits to choose one data input.

A demultiplexer routes one input to a selected output.

A k-to-2^k decoder asserts the selected output, subject to enable and active-level conventions.

An ordinary encoder assumes one active input; a priority encoder defines which wins if several are active.

Registers are storage, not combinational code converters.

Example

Worked microexample

- For a full-adder with A=B=Cin=1, binary total is 11: sum=1 and carry=1. An 8-to-1 mux needs 3 select bits.

Computer Organization and Architecture module

COA08 | Base conversion and binary codes

In base b, a digit at position i has value digit×b^i. Binary uses 0-1, octal 0-7 and hexadecimal 0-9,A-F; F=15.

Convert a positive integer to binary by repeated division by 2 and read remainders upward.

For fractions, repeatedly multiply by 2 and read integer parts downward.

Group binary bits in threes for octal and fours for hexadecimal, outward from the radix point.

BCD encodes each decimal digit separately in four bits; BCD 12 is 0001 0010, while ordinary binary 12 is 1100.

ASCII is a character code, not a physical I/O interface.

Example

Worked microexample

- 1011_2=8+2+1=11. $2F_{16}$=47. 101.101_2=5.625. Decimal 13 gives remainders 1,0,1,1 and hence 1101_2.

Computer Organization and Architecture module

COA09 | Signed arithmetic, subtraction and overflow

For n-bit unsigned values the range is 0 through 2^n−1.

Two’s complement ranges from −2^(n−1) through 2^(n−1)−1.

Negate by inverting all n bits and adding one.

Compute A−B as A+(two’s complement of B), retaining n result bits.

Signed magnitude and one’s complement have two zero encodings; two’s complement has one.

Unsigned carry and signed overflow are different.

Signed addition overflows when equal-sign operands produce an opposite-sign result.

Keep the width fixed throughout the calculation.

Example

Worked microexample

- In 4 bits, 7−3 = 0111+1101 = 1 0100; discard carry to get 4. But 0111+0001=1000 is signed overflow, since +8 is outside −8…+7.

Computer Organization and Architecture module

COA10 | Floating point and arithmetic pipelines

A normalized binary floating-point value is sign×significand×2^exponent.

For ordinary IEEE binary32 normal values, value=(−1)^s(1.f)2^(e−127); binary64 uses 11 exponent bits and bias 1023.

Special exponent patterns represent zero/subnormal values or infinities/NaNs, so the implicit leading 1 is not universal.

For floating addition, compare exponents, shift the smaller significand to align, add or subtract, normalize and round.

Alignment may lose low-order bits.

Overflow concerns exponent range; rounding error concerns finite precision.

An arithmetic pipeline overlaps these steps for different operand pairs; it need not perform all steps in one cycle.

Example

Worked microexample

- $1.10_2 \times 2^3 + 1.00_2 \times 2^1$ becomes $(1.10_2 + 0.01_2) \times 2^3 = 1.11_2 \times 2^3 = 14$.

Computer Organization and Architecture module

COA11 | Register transfers, buses and memory cycles

R2←R1 copies R1 into R2; it does not erase R1.

A control condition P:R2←R1 performs the transfer only when P is true at the relevant clock event.

Simultaneous transfers read old register values; sequential statements can use a newly written value.

A common bus selects one source at a time, while destination load signals select receivers.

A read places an address in MAR/AR, asserts read, waits for memory and captures data in MDR/MBR/DR. A write supplies both address and data and asserts write.

The ALU computes; the control unit sequences and enables the transfers.

A microoperation is one elementary operation on register contents.

Example

Worked microexample

- With R1=3,R2=7, simultaneous R1←R2,R2←R1 swaps to 7,3. Executed sequentially, both end at 7.

Computer Organization and Architecture module

COA12 | Address, data and control buses

Address signals select a location; data signals carry values; control signals coordinate read/write, timing, requests and acknowledgements.

In the simple CPU-master model, addresses flow from CPU to memory, while data travels in either direction.

A DMA controller can also become bus master.

Aggregate control signals include both CPU outputs and device inputs; both data and control buses can therefore be bidirectional under that scope.

With a address bits there are 2^a addressable locations.

Multiply by bytes per location before calling this a byte capacity.

Word width, data-bus width, instruction width and address width need not match.

Example

Worked microexample

- 32 address bits and byte addressing give 2^32 bytes=4 GiB (often called 4 GB in introductory questions). A 16-bit data bus does not itself imply a 16-bit address space.

Computer Organization and Architecture module

COA13 | Logic masks and shifts

To set selected bits use A OR mask; to clear them use A AND NOT(mask); to toggle use A XOR mask; to retain only selected bits use A AND mask.

A XOR A clears the entire register.

Clearing with XOR is valid only with the appropriate operand, not any arbitrary mask.

Logical shifts insert zero.

Arithmetic right shift replicates the sign bit.

Left shift inserts zero and may overflow; it does not preserve the sign in every case.

Rotate returns a shifted-out bit at the opposite end.

SISO shift registers delay serial streams; SIPO collects a word and PISO serializes a parallel word.

Example

Worked microexample

- A=1010, mask=1100: set→1110, selective clear→0010, toggle→0110. Arithmetic-right 1100 gives 1110 (−4 to −2 in two's complement); logical-right gives 0110.

Computer Organization and Architecture module

COA14 | CPU registers and control-word construction

PC identifies the next instruction to fetch.

IR holds the fetched instruction; MAR holds an address used for the current memory access; MDR holds transferred data.

The accumulator holds operands/results, SP tracks the stack, and status flags report conditions. "Address of the instruction" without a phase can describe PC or MAR.

Registers and cache both store temporary information, but registers are explicit execution operands and cache holds copies of memory blocks.

A control word chooses ALU sources, destination and operation.

Use the machine's encoding table rather than guessing opcodes.

In this 14-bit control-word example, SELA, SELB and SELD each have 3 bits, followed by a 5-bit operation field.

Example

Worked microexample

- For R1←R2−R3 in this encoding, select 010,011,001 and subtraction 00101: 010 011 001 00101. Source and destination order matters.

Computer Organization and Architecture module

COA15 | Fetch, decode, execute and next-PC choice

A conceptual instruction cycle fetches an instruction, decodes opcode/addressing fields, obtains operands, executes, writes results where required and checks permitted interrupts.

A five-stage implementation may name stages IF, ID, EX, MEM and WB; these stages are not mandatory for every architecture.

A sample fetch is MAR←PC; MDR←M[MAR]; IR←MDR; PC←PC+instruction_length.

A branch can replace the sequential PC later.

Thus "which phase determines the next instruction?" needs to distinguish reading the current PC from calculating a branch target.

The control unit coordinates fetching and data flow; the ALU performs arithmetic and logic.

A write-back/result-storage phase is not another fetch.

Computer Organization and Architecture module

COA16 | Opcode fields and address-count formats

An opcode specifies the operation; operand fields locate data or registers, and a mode field explains interpretation.

A 4-bit opcode permits at most 16 encodings, not necessarily 16 implemented instructions.

A 6-bit operand-reference field has 64 encodings; what they designate depends on the machine.

Three-address ADD X,Y,Z means X←Y+Z under destination-first notation.

Two-address ADD X,Y overwrites X with X+Y. One-address accumulator ADD X means AC←AC+M[X].

Zero-address stack ADD uses two implicit top-of-stack operands.

Count explicit operand fields; an instruction such as INC X does not become an accumulator instruction merely because it has one written operand.

Computer Organization and Architecture module

COA17 | Stacks and evaluation order

A stack is last-in, first-out.

PUSH adds an item and POP removes the logical top.

A register stack uses registers; a memory stack uses RAM and a stack pointer.

Check whether SP points to the top or the next free location, whether addresses increase or decrease, and whether the update occurs before or after access.

Overflow means pushing a full stack; underflow means popping an empty one.

Zero-address arithmetic typically pops the second operand first, then the first operand; subtraction therefore needs careful order.

PUSH and POP may still contain an explicit address even on a stack machine.

Example

Worked microexample

- For (A+B)×(C+D): PUSH A; PUSH B; ADD; PUSH C; PUSH D; ADD; MUL. For a downward-growing full stack, PUSH commonly decrements SP before storing and POP reads before incrementing.

Computer Organization and Architecture module

COA18 | Effective addresses and addressing modes

Let A be an instruction field, R a register value and M[x] the contents at x. Immediate uses value A; direct uses EA=A and operand M[A]; indirect uses EA=M[A] then operand M[EA].

Register uses operand R; register-indirect uses EA=R. Indexed/base displacement uses EA=R+A; PC-relative uses the architecture-defined PC plus a signed displacement.

Auto-increment/decrement also updates the register by operand size.

A transfer between a register and memory does not uniquely specify addressing mode; either direct, indexed or indirect addressing may produce it.

Example

Worked microexample

- A=100,R=20,M[100]=300,M[300]=9: direct operand is 300; indirect operand is 9; indexed EA is 120. An immediate field 100 is the value 100, not M[100].

Computer Organization and Architecture module

COA19 | Instruction classes and control flow

Data-transfer instructions copy values.

Arithmetic/logic instructions compute values.

TEST/CMP examine values and set condition information, often without retaining the arithmetic result.

Branch instructions choose a target, conditionally or unconditionally.

CALL saves a return point; RETURN restores it.

SKIP, where supported, bypasses the next instruction when its condition holds. “Jump to the next instruction” does not clearly mean skip.

External interrupts arise from devices or timers; synchronous internal exceptions arise from the executing instruction, such as divide errors, illegal opcodes or access violations.

A software interrupt is deliberately requested by an instruction.

Arithmetic overflow trapping depends on architecture and instruction; ordinary overflow flags need not trigger an interrupt.

Computer Organization and Architecture module

COA20 | RISC, CISC and control implementation

Textbook RISC emphasizes simple operations, regular encodings, load/store memory access, plentiful registers and pipeline-friendly control.

CISC commonly supplies more complex operations, richer addressing modes and variable lengths.

Hardwired control directly generates control signals; microprogrammed control sequences stored control words.

These are tendencies, not universal guarantees.

One instruction completing per pipeline cycle is throughput, not proof that every instruction has one-cycle latency.

Modern RISC instructions can take multiple cycles and modern CISC processors can translate instructions into simpler internal operations.

Neither label guarantees lower power, cost or execution time.

The semantic gap is the distance between high-level constructs and machine operations; reducing it is associated with richer instructions, not a universal shared objective of both design approaches.

Computer Organization and Architecture module

COA21 | x86 register names and classification

In Intel x86 notation, AX/BX/CX/DX, BP/SP and SI/DI are 16-bit general-purpose registers, with conventional special roles.

AL is AX's low byte and AH its high byte; analogous byte names apply to BX/CX/DX.

EAX is 32-bit and RAX 64-bit.

FLAGS/EFLAGS/RFLAGS report status; IP/EIP/RIP identifies instruction position.

CS and SS are code/stack segment registers.

DX is programmer-visible; IR/MDR are conceptual internal registers.

SP's stack role does not remove it from Intel's general-purpose register classification.

EAX/EIP existed with the 80386, before Pentium; R8-R15 belong to the 64-bit extension, not original Pentium.

State the textbook convention when using a narrower "multipurpose" register category.

Intel Software Developer Manual, Volume 1

COA22 | x86 instruction legality and implicit operands

Assume Intel destination-first syntax and ordinary legacy x86 forms.

MOV AL,BP is invalid because operand widths differ (8 versus 16 bits).

INC CX and DEC CX are valid word operations.

Segment-register MOV has register/memory forms, not a direct immediate-to-segment form: load an immediate into AX, then MOV DS,AX.

Unsigned MUL with a byte operand multiplies AL by that byte and writes AX.

With a word operand it multiplies AX and writes DX:AX.

Thus AL=8 and BH=10 gives AX=80 after MUL BH; CL is irrelevant.

MOVS and LODS are string transfer operations, XCHG exchanges operands, and INC is arithmetic. “Different” requires the transfer-versus-arithmetic criterion.

Intel Software Developer Manual, Volume 2B

COA23 | Memory hierarchy and timing measures

A usual speed/cost-per-bit order is registers, L1/L2/L3 cache, DRAM, SSD, HDD and archival tape.

Faster levels are generally smaller and more expensive per bit.

Access time measures latency of a read/write; cycle time is the minimum interval before another operation and may include recovery.

Transfer rate measures data per time, not individual-request delay.

Temporal locality reuses recent data; spatial locality uses nearby locations.

Cache copies memory blocks near the CPU.

Hit ratio=hits/accesses; miss ratio=1−hit ratio.

Distinguish capacity, latency, bandwidth and hit ratio before comparing alternatives.

Example

Worked microexample

- For 90 hits in 100 accesses, hit ratio=0.9. If hit time is 2 ns and an additional miss penalty is 50 ns, AMAT=2+0.1×50=7 ns.

Computer Organization and Architecture module

COA24 | RAM, firmware and memory-chip capacity

SRAM uses powered bistable storage: fast and costly per bit, without periodic refresh.

DRAM stores charge and requires refresh: denser and commonly used for main memory.

Both are ordinarily volatile.

ROM and flash retain contents without power; erasable/programmed variants mean “ROM can never be modified” is too broad.

Firmware is low-level system code, commonly in nonvolatile storage.

For a chip described as 2^a×w, a is the number of address inputs and w the data width.

Combine chips in parallel to increase word width and in banks to increase the number of words.

Distinguish address decoding from data selection.

Example

Worked microexample

- Eight 1K×1 chips in parallel make 1K×8 storage: 10 shared address lines and 8 data bits. Four such banks make 4K×8 and require 12 address bits overall.

Computer Organization and Architecture module

COA25 | Cache placement, index bits and conflicts

Let B be block size in bytes and L the number of cache lines.

Block number=floor(address/B); offset=address mod B. Direct mapping uses line=block mod L. Fully associative placement permits any line.

A W-way set-associative cache has S=L/W sets and set=block mod S; any of W ways in that set may hold the block.

For power-of-two sizes, offset bits=log2 B, index bits=log2 S and remaining address bits form the tag.

Compare tags and valid bits to identify a hit.

Two blocks with the same direct-mapped index cannot reside together.

Associations reduce placement conflicts but require more tag comparisons.

Example

Worked microexample

- 16-byte blocks and four direct-mapped lines: addresses 0 and 64 have block numbers 0 and 4, both index 0, so alternating them causes conflict misses.

Computer Organization and Architecture module

COA26 | Cache replacement and write-policy traces

FIFO evicts the earliest inserted line; LRU evicts the least recently accessed line.

On each access, record hit/miss, the set involved and policy state.

Hits update LRU order but do not reset FIFO insertion order.

Direct mapping has no replacement choice within a line.

Write-through updates the lower level on each write; write-back modifies cache and marks a dirty bit, writing to the lower level on eviction.

Write-allocate fetches a missed block before writing; no-write-allocate writes below without installing it.

L1/L2 describe hierarchy levels; split cache means separate instruction/data caches and unified means combined.

Internal/external location is another classification, not synonymous with split/unified.

Example

Worked microexample

- Two fully associative lines, references A,B,A,C: all except the second A are misses. LRU evicts B for C; FIFO evicts A.

Computer Organization and Architecture module

COA27 | Virtual addresses, MMU and page faults

Virtual memory gives processes translated address spaces and protection; backing storage can hold nonresident pages.

The MMU translates virtual/logical addresses to physical addresses, using page tables and often a TLB of recent translations.

Virtual memory is not just extra disk disguised as RAM.

With page size P, page=floor(virtual_address/P), offset=virtual_address mod P and physical_address=frame×P+offset.

A valid nonresident page access faults and may require loading data; invalid or forbidden accesses can also fault.

A cache miss, TLB miss and page fault are different events; not every page fault reads a disk, since zero-fill and already-resident shared pages exist.

Example

Worked microexample

- With 1024-byte pages, address 2053 is page 2, offset 5. If page 2 maps to frame 7, physical address=7173.

Computer Organization and Architecture module

COA28 | Persistent storage and access methods

An HDD stores magnetic patterns on tracks divided into sectors; reads/writes access addressed blocks while platters rotate.

A head normally stays over a track while reading but moves radially for seeks.

It does not need continuous power to preserve stored bits.

SSDs use nonvolatile semiconductor storage without mechanical seek.

Optical media use lasers; tape is suited to sequential streaming and archives.

An Ethernet cable is a communication medium, not a disk or RAM.

In textbook access taxonomy, tape is sequential, disk is direct access with positioning delay, RAM is random access with broadly address-independent timing, and associative access matches

content. “Random-access disk” is also common informal usage; specify the taxonomy when comparing direct and random access.

Optical access ranges such as 100-300 ms are historical examples, not universal device specifications.

Computer Organization and Architecture module

COA29 | I/O interfaces, registers and commands

A peripheral performs input, output or storage outside the CPU’s execution core; CPU itself is not a peripheral.

Input units encode external information and output units present processed results.

A device controller handles device protocol/timing; its interface commonly exposes data, status and control registers.

Buffers absorb speed differences and error/status bits report conditions.

Control commands start actions, test/status commands inspect readiness or errors, read commands bring device data toward the system, and write/output commands send system data to the device. “Issue” describes sending a command, not a standard fourth command category.

A controller is hardware; a driver is software that operates it.

Computer Organization and Architecture module

COA30 | Polling, interrupts and DMA

Programmed/polled I/O repeatedly checks status and lets the CPU move each item.

Interrupt-driven I/O lets the CPU do other work until an event requests service; the handler still consumes CPU time to service the device.

DMA lets a controller move a block directly between device and memory after CPU setup.

Completion or errors can interrupt the CPU.

DMA does not eliminate CPU involvement in configuration, ownership, address mapping or completion.

Burst mode can hold the bus for a block; cycle stealing takes individual transfer opportunities, and transparent DMA uses idle opportunities.

DMA reduces per-word CPU work but competes for memory bandwidth.

Choose it for substantial blocks; mapping devices to addresses is a separate design dimension.

Example

Worked microexample

- For 1000 words, polling or interrupt transfer may require CPU work per word. DMA replaces much of that work with setup and completion, although the actual memory transfers still take time.

Computer Organization and Architecture module

COA31 | I/O address spaces, priorities and handshaking

Memory-mapped I/O assigns device registers addresses in the memory address space and uses suitable load/store operations.

Isolated/port-mapped I/O has a separate port space and special instructions such as IN/OUT.

MMIO must respect device access widths, ordering and side effects; a status read is not necessarily a harmless RAM read.

Asynchronous handshaking uses request/ready and acknowledgement signals to tolerate unequal speeds: sender presents data and requests, receiver accepts and acknowledges, then each withdraws its signal.

A single strobe lacks this feedback.

Interrupt priority selects among simultaneous requests, using schemes such as polling, daisy chains or priority hardware.

Serial sends bits over time; parallel transfers multiple bits on multiple lines, without guaranteeing higher real-world speed.

Computer Organization and Architecture module

COA32 | Named interfaces: choose by connection role

USB is a serial peripheral interconnect used by keyboards and external storage; RS-232 specifies a traditional point-to-point serial electrical/interface standard.

Both RS-232 and USB use serial communication; distinguish their interface roles and specifications.

HDMI carries digital video and audio.

Ethernet connects network devices; it is not memory.

SATA is a storage interface; legacy IDE/PATA and SCSI represent other storage connections, while iSCSI carries SCSI commands over IP.

PCI is a peripheral expansion interconnect, not a disk protocol; connecting unlike bus segments requires suitable bridge/controller logic. “Extends processor-bus connectivity” without topology does not uniquely identify PCI or Multibus.

POSIX is a software interface standard with operations such as open/read/write/close; ASCII is a character encoding.

An unqualified “standard PC disk interface” needs an era or device requirement.

USB Implementers Forum, mass storage transport specification

Texas Instruments AN-216

HDMI Licensing Administrator specifications overview

Linux libATA Developer Guide

COA33 | Analog interfaces and component identification

An ADC measures an analog input and produces a digital code; a DAC produces an analog output from a digital code.

ADC080x parts are 8-bit successive-approximation analog-to-digital converters; DAC0830 is a digital-to-analog converter, while 82C55 denotes a programmable parallel I/O interface, not an ADC.

Component names are not interchangeable with the direction of conversion.

For an ideal n-bit converter spanning a voltage range V, one code step is approximately V/2^n; actual endpoint conventions and specifications must be stated.

Quantization rounds an analog range into discrete codes; sample rate and resolution measure different properties.

Example

Worked microexample

- An ideal 8-bit ADC over a 5.12 V span has 256 codes and a 0.02 V step. Greater bit depth reduces the quantization step for the same span.

Texas Instruments ADC0804-N

COA34 | Execution-time calculations and bottlenecks

CPU time=instruction_count×CPI×clock_period=instruction_count×CPI/clock_rate.

CPI may be a weighted average across instruction classes; IPC is its reciprocal only under a consistent whole-program definition.

Raising clock frequency does not guarantee a faster program when instruction count or stalls increase.

Throughput counts completed work per unit time; latency measures one job’s delay.

When a fraction f of execution is sped up by factor s, overall speedup=1/((1−f)+f/s).

The unaffected fraction limits gains.

Compare the same workload and include memory and branch penalties instead of equating clock speed with performance.

Example

Worked microexample

- 10^6 instructions, CPI=2 and 1 GHz take 2 ms. If half the time becomes twice as fast, overall speedup=1/(0.5+0.25)=4/3, not 2.

Computer Organization and Architecture module

COA35 | Pipelines: overlap and timing

A pipeline overlaps stages of different instructions; it primarily improves throughput.

In an ideal k-stage single-issue pipeline, n instructions take k+n−1 cycles when stages are balanced and there are no stalls.

Without overlap, comparable work takes nk stage cycles.

The ideal steady-state completion rate is one instruction per cycle, while an individual instruction still spans multiple stages.

Choose the pipeline clock using the slowest stage plus register overhead.

Fill/drain overhead matters for short streams. “Fetch/decode one instruction while executing another” describes pipelining, whereas issuing several instructions in one cycle requires multiple-issue resources.

Example

Worked microexample

- A five-stage pipeline processing 10 instructions takes 14 ideal cycles versus 50 without overlap: speedup=50/14≈3.57. It does not reach the limiting factor of five for this short stream.

Computer Organization and Architecture module

COA36 | Pipeline hazards and instruction-level parallelism

Structural hazards compete for one resource.

Data hazards arise when an instruction needs a result not yet available; forwarding can supply results early, but some dependencies still stall.

Control hazards arise when the next PC depends on an unresolved branch.

Branch prediction guesses direction/target and continues speculatively; wrong guesses discard work.

Superscalar execution issues multiple instructions to multiple functional units in a cycle when dependencies permit.

Out-of-order execution schedules ready operations before stalled older ones while preserving architectural results and exception behavior.

Dynamic resource allocation assigns available internal entries/units at runtime.

These mechanisms exploit instruction-level parallelism; wider fetch improves instruction supply, not necessarily individual memory latency.

Computer Organization and Architecture module

INFLIBNET Computer Architecture, Multiple Issue Processors I

COA37 | Multiprocessing, vectors and Flynn classification

Flynn classifies instruction and data streams: SISD has one of each; SIMD applies one instruction stream to multiple data elements; MIMD uses multiple instruction and data streams; MISD has multiple instruction streams for a common data stream and is uncommon.

A vector instruction describes an operation over a sequence of elements; physical lanes may process them in batches, not all in one instant.

SMP processors share memory/I/O and have peer roles under an OS.

Distributed-memory systems communicate between separate address spaces.

More cores increase available parallel work, but serial dependencies and coordination limit speedup. “Core i7” is a product family rather than one core count; a model number is essential.

Computer Organization and Architecture module

COA38 | Historical conventions and processor-width caveats

The familiar generation convention associates first with vacuum tubes, second with transistors, third with integrated circuits and fourth with microprocessor/VLSI-era personal computers.

Stored-program machines arose in the first era.

High-level languages became prominent around the transition into the second; introduction dates do not fit every generation boundary.

Magnetic-core memory spanned more than one era, so “witnessed use” cannot identify exactly one.

Several generations improved performance.

Integration generally reduced individual component dimensions while capacity, density and processor performance grew; these are trends rather than guarantees for every model.

Intel’s 80386 introduced 32-bit x86 execution in 1985, before 80486/Pentium; the 80286 was a 16-bit predecessor.

An n-bit processor label does not force its PC, IR or address register to be n bits.

Columbia University Computing History, Computer Generations

Intel 1987 Annual Report

Computer Organization and Architecture module

COA39 | Identify architecture assumptions before answering

Main memory, registers and cache can all provide temporary storage; identify the scope and role. Memory can also include nonvolatile ROM/flash, so distinguish RAM when comparing power-off data retention with disk storage.

Architecture lifetime, power efficiency and generational improvements require assumptions; definitions alone do not make these exclusive.

For a TEST-instruction question, identify the instruction and the complete statements being compared before drawing a conclusion.

Identify a logic gate from its actual graphical symbol.

An opcode can be identified from an explicit description of its operation-field role even without a figure.

Check that the question includes the phrase, choices or diagram needed to apply the relevant architecture rule.

Computer Organization and Architecture module

COA-S01 | Model-specific core and thread counts

Intel’s desktop comparison chart lists the Core i7-920 with four physical cores and eight hardware threads. Core count and thread count are distinct specification fields: use the exact processor model and the field requested rather than inferring a count from the Core i7 family name.

Intel Core i7 Desktop Processor Comparison Chart

COA-S02 | Buses as shared communication paths

A bus is a set of common signal lines used to transfer binary information among connected units. In a parallel register bus, one line carries each bit position; control signals select a source and enable the intended destination. Storage registers and memory retain information, while the bus provides a transfer path.

Computer Organization and Architecture module · page 40

Computer Organization and Architecture module · page 6

COA-S03 | The basic single-accumulator memory instruction

In the textbook single-accumulator organization, a memory-reference instruction has an operation part and an address part. The address locates a memory operand; the operation combines that operand with the accumulator as required. The address field is not the operand value itself, and this simple format does not describe every addressing mode or instruction set.

Computer Organization and Architecture module · page 47

COA-S04 | Textbook instruction-set and control tendencies

RISC means Reduced Instruction Set Computer; CISC means Complex Instruction Set Computer. The textbook RISC design pairs a relatively large register set, load/store access and easily decoded instructions with hardwired rather than microprogrammed control and an efficient pipeline. These are design tendencies, not guarantees of one-cycle latency or lower power for every implementation.

Computer Organization and Architecture module · page 77

Computer Organization and Architecture module · page 78

COA-S05 | Instruction and stack pointers in 64-bit x86

In Intel 64-bit mode, RIP holds the offset of the next instruction to execute. RSP is the 64-bit stack pointer, extending the ESP/SP stack role. These pointers are distinct from CS and SS, which are code and stack segment registers.

Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1 · §§3.4-3.5 and 6.1

COA-S06 | Exclusive-OR notation

The circled plus symbol ⊕ denotes exclusive OR. For two Boolean inputs, A⊕B is 1 exactly when the inputs differ; an overbar or output inversion would complement this result.

Texas Instruments SNx4HC86 Quadruple 2-Input XOR Gates · description

COA-S07 | Bistable one-bit state

A flip-flop is bistable: it has two stable states representing 0 and 1. While powered, it can retain one bit until an input or designated clock event changes its state. Multiple flip-flops are required to represent a multi-bit counter state.

Computer Organization and Architecture module · page 16

Data Communication and Computer Networking

Complete Study Notes

NET01 | Communication components and quality

Data are raw representations; information is data interpreted for a purpose.

Communication requires a message, sender, receiver, medium and agreed protocol.

A source creates data; a transmitter converts it to a signal; a receiver reconstructs data.

Distance affects engineering but is not a sixth component.

Delivery means reaching the intended recipient; accuracy means preserving content; timeliness means meeting a deadline; jitter means variation in arrival delay.

A call can have acceptable average delay yet sound broken because of jitter.

Throughput measures achieved transfer rate, reliability measures dependable operation, and security concerns unauthorized actions.

Scalability means accommodating growth without unacceptable service degradation.

Data Communication and Computer Networking module

NET02 | Network size, access and architecture

PAN connects personal devices, such as Bluetooth accessories.

LAN covers a local building or campus; MAN spans a metropolitan region; WAN connects distant regions.

Device count alone does not determine the category.

The Internet is a global network of networks, commonly the largest example of a WAN.

An intranet serves an organization internally; an extranet exposes selected services to approved outsiders.

Neither term alone specifies geographic size.

A client requests a service; a server supplies it.

Peer systems may perform both roles.

Dedicated servers concentrate on service workloads but can still be clients of DNS, backup or update services.

A ten-user cutoff is a teaching guideline, not a technical law.

Centralization simplifies administration but creates dependencies; peer networks still need security.

Data Communication and Computer Networking module

NET03 | Protocols, standards and layers

Protocol syntax defines message structure, semantics defines field meanings, and timing defines when/how fast exchanges occur.

A standard supports interoperability.

De facto means established through use; de jure means formally approved.

ISO develops international standards; IEEE develops LAN standards such as 802.3 and 802.11; ITU-T develops telecommunications standards.

ANSI coordinates US standards, EIA is associated with interface standards, and FCC is a regulator.

A reference model divides responsibilities and interfaces; it is not itself executable networking software.

OSI is a seven-layer reference model; TCP/IP describes an implemented protocol suite, usually taught with four layers.

A five-layer teaching model separates physical and link functions.

State the model before counting layers.

Data Communication and Computer Networking module

NET04 | Signals, periods and bandwidth

Analog signals vary continuously; digital signals use discrete symbol states.

Data and signal types are separate: digital bits can modulate an analog carrier.

A sine wave has amplitude, frequency and phase.

Frequency f counts cycles/second, period T=1/f, and wavelength λ=v/f depends on propagation speed v. Phase specifies displacement within a cycle.

Composite signals combine frequencies.

Frequency bandwidth is highest minus lowest frequency, measured in hertz.

A low-pass channel includes zero frequency; a band-pass channel occupies a nonzero frequency range.

Baseband sends the signal without shifting it to a carrier; passband transmission modulates a carrier.

Do not confuse hertz with bit/s, even though both are informally called bandwidth.

At 2 kHz, one period is 0.5 ms.

Data Communication and Computer Networking module

NET05 | Bit rate, baud and channel capacity

Bit rate R is bits/second; a bit interval is 1/R. Baud is symbols/second, not necessarily the number of voltage transitions.

With M equally encoded symbol states, each symbol can carry log2(M) bits; R=baud×log2(M), before coding overhead.

Binary Manchester uses extra transitions, so binary does not universally mean baud equals bit rate.

For an ideal noiseless channel, Nyquist gives R≤2B log2(M).

For Gaussian noise, Shannon gives C=B log2(1+S/N), with a linear power ratio; SNR(dB)=10 log10(S/N).

Bandwidth, signal levels and noise appear explicitly.

Distance affects attenuation, delay and often achievable rate, but is not a separate variable in these simplified capacity formulas.

Example

Worked example

- B=3000 Hz and M=4 give a 12,000 bit/s noiseless limit. If S/N=15, Shannon also gives 3000×log2(16)=12,000 bit/s. Do not insert 15 dB as a linear ratio.

University of Florida, R. Newman-Wolfe: Transmission

NET06 | Delay, impairments and transmission modes

Transmission delay is packet bits divided by link bit rate; propagation delay is distance divided by signal speed.

Processing and queueing add delay.

A 1000-byte packet takes 0.8 ms to serialize at 10 Mbit/s; increasing line rate does not reduce propagation time.

Attenuation reduces signal strength.

Distortion changes shape, often because frequencies have different delays.

Noise adds unwanted energy: thermal noise comes from random thermal motion, crosstalk leaks from nearby channels, and impulse noise is a short disturbance.

Amplifiers amplify noise too; digital repeaters regenerate decisions but cannot reconstruct every corrupted bit.

Simplex is one way, half duplex alternates directions, and full duplex permits simultaneous sending/receiving.

A walkie-talkie is half duplex; a telephone conversation uses full duplex.

Serial sends sequential bits; parallel uses multiple conductors and may suffer skew.

Character-oriented asynchronous serial adds start/stop bits; synchronous framing uses ongoing clock recovery.

Neither is synonymous with browser asynchronous requests.

Data Communication and Computer Networking module

NET07 | Cables, wireless and connectors

Twisted pairs reduce coupled interference; UTP has no added shield and STP adds shielding.

Coax has an inner conductor and outer shield sharing an axis.

Fiber guides light and resists electromagnetic interference; it supports high capacity and long reach, but requires suitable optics and installation.

Exact speed, cost and reach depend on the specification, not the material alone.

Fiber can still be tapped.

Radio, microwave and infrared are unguided media.

Ground waves follow Earth, sky waves can reflect from ionized layers, and line-of-sight paths require suitable visibility.

Microwaves are radio waves in the broader physical classification.

Infrared generally does not penetrate walls.

Directionality depends on antennas: omnidirectional means broad azimuth coverage, not equal radiation in every 3-D direction.

Legacy Ethernet straight-through cables join unlike MDI/MDI-X roles; crossover joins like roles, such as PC-PC or switch-switch.

Auto-MDI-X can remove that requirement.

Data Communication and Computer Networking module

NET08 | Topology and failure reasoning

Physical topology describes link layout; logical topology describes communication/access relationships.

Point-to-point links join two endpoints; multipoint links share a medium.

A full mesh consists of point-to-point links.

Bus uses a shared backbone; an unterminated break can disrupt the segment.

Star connects endpoints through a central device: one edge failure isolates that endpoint, whereas central failure affects the star.

Ring forms a loop; basic single rings are vulnerable to a break, while protected dual rings differ.

Token passing is associated with Token Ring but is an access method, not a necessary property of every ring.

Tree is hierarchical; hybrid combines arrangements.

A full undirected mesh of n nodes needs n(n-1)/2 links and n-1 interfaces per node.

For 50 nodes: 50×49/2=1225 links.

Redundant paths improve fault tolerance at greater cable and interface cost.

Data Communication and Computer Networking module

NET09 | NICs, repeaters and hubs

A NIC supplies a computer's network interface and commonly implements both physical and link functions.

Calling a NIC a physical interface does not mean all its functions are layer 1.

Physical specifications include connectors, signaling, voltage, timing and physical coding/modulation; the PHY interface to MAC handles hardware signaling.

A repeater regenerates layer-1 signals without inspecting destination addresses.

An active hub is a multiport repeater: it repeats incoming signals onto other ports, placing half-duplex endpoints in one collision domain.

A passive hub is a connector.

A hub neither learns MAC addresses nor eliminates collisions.

An endpoint such as a network printer consumes services; routers, hubs and access points are intermediary devices.

In the legacy IEEE 802.3 architecture with an attachment unit interface (AUI) and medium attachment unit (MAU), the MAC interfaces to the Physical Signaling (PLS) sublayer.

For this legacy MAC interface, the relevant sublayer is PLS.

This name is architecture-specific: other PHY designs use reconciliation (RS), physical coding (PCS), physical medium attachment (PMA) and related sublayers.

Do not present PLS as a universal OSI sublayer name.

IEEE 802.3 10GBASE-ER architectures, David Law

NET10 | Bridges, switches and access points

A bridge or ordinary Ethernet switch operates at layer 2.

It learns source-MAC-to-port mappings, looks up the destination, forwards a known unicast to its port, filters it if the destination is on the incoming segment, and floods an unknown unicast or broadcast within its VLAN.

A switch does not always send only to one destination port.

Wireless access points normally bridge wireless and wired link-layer traffic.

Each switch port separates collision domains; full-duplex point-to-point Ethernet has no collisions and does not use CSMA/CD.

Switching permits independent conversations simultaneously.

Broadcasts still reach the VLAN unless another control filters them.

A layer-3 switch also routes, so device marketing names do not override the function being examined.

Data Communication and Computer Networking module

NET11 | Routers, gateways and traffic controls

Routers forward IP packets between networks using destination prefixes and next hops.

They separate broadcast domains and normally do not forward local link broadcasts.

A default gateway is usually a router, while a protocol gateway can translate application formats; therefore a gateway is not inherently a device that implements every OSI layer.

A firewall allows, rejects or inspects traffic under a security policy; it does not necessarily encrypt.

A VPN, Virtual Private Network, provides a logical private connection over another network; a secure VPN authenticates/encrypts its tunnel.

A modem modulates outgoing and demodulates incoming carrier signals.

These functions may coexist in one home router but remain conceptually distinct.

Data Communication and Computer Networking module

NET12 | OSI layers and encapsulation

Sending proceeds from application data to transport segment/datagram, network packet, link frame and physical bits/symbols; receiving reverses the order.

Each layer serves the layer above.

A router removes a received link frame, inspects IP, and creates a new link frame for the next hop.

MAC addresses usually change across routed hops; end-to-end IP addresses ordinarily remain until translation.

Layer	Main duty	Typical unit/address
7 Application	Network services for applications	Message/data
6 Presentation	Representation, compression, encryption roles	Data
5 Session	Dialog setup, checkpoints, recovery	Data
4 Transport	Process delivery, segmentation; protocol-dependent reliability	TCP segment or UDP datagram; port
3 Network	Logical addressing and routing	Packet; IP
2 Data link	Hop delivery, framing, MAC access and error detection	Frame; MAC
1 Physical	Signaling and bit/symbol transport	Bits/symbols

Data Communication and Computer Networking module

NET13 | Link control, framing and error detection

Link-layer LLC identifies or multiplexes upper-layer protocols and may supply link control; MAC handles medium-dependent addressing/access and frame delivery.

Network addressing belongs to layer 3.

Ethernet, PPP and HDLC are link protocols.

Frame headers may carry addresses and control/type fields; synchronization and exact framing depend on the protocol.

Detection adds redundancy but does not guarantee correction.

Even parity adds a bit so the total number of ones is even: 1011 needs parity 1.

It detects every odd number of bit flips but misses even counts.

CRC treats bits as a polynomial and appends the XOR-division remainder; the receiver repeats the division.

Internet checksums add words with end-around carry and complement the sum.

A burst spans from the first corrupted bit through the last, including unchanged interior bits.

Retransmission or sufficient forward-error-correcting redundancy is needed for recovery.

Data Communication and Computer Networking module

NET14 | Access control and reliable-link procedures

Random access allows contention: ALOHA sends opportunistically; CSMA senses before sending.

In legacy half-duplex Ethernet, CSMA/CD detects collision, sends a jam indication, stops, and retries after random backoff.

The jam signals a collision, not its clearance.

Wi-Fi 802.11 uses CSMA/CA with waiting/backoff and acknowledgments; sensing does not eliminate hidden-terminal collisions.

Controlled access uses polling, reservations or a token to grant turns; correctly coordinated access avoids contention collisions.

Channelization divides frequency, time or code resources.

Stop-and-wait sends one frame and waits for ACK, retransmitting on timeout; sequence bits distinguish duplicates.

Sliding windows permit multiple unacknowledged frames.

Link reliability/flow control is hop-local; transport reliability/flow control is end-to-end.

Ethernet detects corruption but does not itself guarantee reliable delivery.

Data Communication and Computer Networking module

NET15 | Line coding, modulation and sampling

Line coding maps digital bits to digital signal patterns.

NRZ uses levels without a forced mid-bit return; Manchester supplies a mid-bit transition for clocking, with bit convention specified.

Differential Manchester encodes using transitions relative to previous state.

AMI alternates the polarity of successive ones; pseudoternary alternates zero pulses.

Block coding adds code redundancy; scrambling replaces problematic patterns without necessarily increasing bit count.

Multilevel schemes include 2B1Q and 8B6T; MLT-3 uses a sequence of voltage transitions.

ASK varies carrier amplitude, FSK frequency, PSK phase; QAM combines amplitude and phase.

AM/FM/PM can carry analog data.

PCM digitizes analog input by sampling, quantizing and encoding.

Sampling must exceed twice the highest retained baseband frequency in practical designs; b-bit samples have 2^b quantization levels and uncompressed rate fs×b. PAM/PWM are pulse modulation methods; pulse amplitude or width alone is not a binary PCM code.

Data Communication and Computer Networking module

NET16 | Network delivery and IPv4 header

IP supplies connectionless best-effort packet delivery, not end-to-end ordering or retransmission.

IPv4 addresses are 32 bits.

Its minimum header is 20 bytes and can grow to 60 bytes; it is not a 32-bit header.

IHL counts 32-bit words; total length includes header and payload.

TTL limits lifetime across routers; the protocol field identifies TCP, UDP or another upper-layer protocol.

The IPv4 checksum protects its header; IP does not lack error checking altogether.

Identification, flags and offset support fragmentation.

IPv4 sources and routers may fragment when allowed; reassembly occurs at the destination.

If DF is set and the next MTU is too small, forwarding fails and an ICMP diagnostic may be returned.

IETF RFC 791

NET17 | Transport choice, ports and connections

TCP offers a reliable ordered byte stream using sequence numbers, acknowledgments, retransmission and windows.

Flow control protects the receiver; congestion control protects network capacity.

UDP sends independent datagrams with ports and a checksum but no built-in setup, delivery guarantee, retransmission or ordering.

Live interactive traffic may choose UDP when late data is less useful than lost data.

Application protocols can add reliability over UDP.

Ports identify transport endpoints; the application can choose a transport and request socket buffer sizes, subject to OS limits.

A TCP connection is identified by source/destination IP and source/destination port, with TCP understood.

IPv4 protocol numbers identify upper protocols; 80 is a usual HTTP port, not an IP protocol number.

Clients often use ephemeral ports.

SCTP is a reliable message-oriented transport with multiple streams.

IETF RFC 9293

NET18 | Trace the TCP handshake

TCP establishes state with SYN, SYN+ACK, then ACK.

SYN consumes one sequence position even with no application bytes.

An acknowledgment states the next expected sequence number.

For client ISN=395 and server ISN=260: client sends SYN seq395; server sends SYN+ACK seq260 ack396; client sends ACK seq396 ack261.

Later byte counts advance sequence numbers.

FIN also consumes one sequence position.

TCP setup creates a logical transport connection, not a reserved physical circuit.

Ports allow multiplexing separate applications over IP; they do not combine all applications into one TCP byte stream.

Do not assign TCP reliability to all layer-4 protocols merely because the reference layer lists reliability as a possible service.

IETF RFC 9293

NET19 | Upper layers and the TCP/IP correspondence

OSI session roles establish/manage dialogs and can coordinate checkpoints so transfer resumes after interruption.

Presentation roles translate encodings and handle compression/encryption.

Compression reduces representation size; encryption hides content and need not shrink it.

Application protocols provide services directly to applications.

Practical implementations distribute these functions differently; TLS is not proof of a separate physical presentation-layer device.

The four-layer TCP/IP model folds OSI layers 5-7 into application, retains transport, calls layer 3 Internet, and combines link/physical as network access.

A model is a guide to functions, not a guarantee that every application gets a reliable network.

Distinguish end-to-end application service from its underlying process-to-process transport.

Data Communication and Computer Networking module

NET20 | DNS, DHCP and address resolution

DNS resolves names to records, including IPv4 A and IPv6 AAAA addresses; it can use UDP or TCP port 53.

DHCP dynamically leases IPv4 settings, commonly address, mask, gateway and DNS server.

Its basic exchange is discover, offer, request, acknowledge.

IP itself does not assign addresses.

ARP resolves an on-link IPv4 next hop to a link address, using a broadcast request and normally unicast reply.

For an off-subnet target, ARP resolves the gateway, not the remote target's MAC.

It bridges the network/link addressing boundary and is often classified at the link layer; calling ARP simply a network protocol uses a loose textbook convention.

RARP reverses hardware-to-IP lookup; BOOTP/DHCP support boot configuration.

ICMP reports errors and diagnostics; IGMP manages IPv4 multicast membership.

These are separate companion protocols, not four pieces whose combination is IP.

IETF RFC 826

NET21 | Web, file, mail and remote-login protocols

HTTP defines web request/response semantics; it is stateless, and a message body may be absent.

HTTPS adds TLS protection.

FTP transfers files using control and separate data connections; it is not the same protocol as SFTP.

SMTP submits and relays mail; POP3 retrieves messages and IMAP accesses/synchronizes server mailboxes.

Saying SMTP handles both sending and end-user mailbox retrieval is inaccurate.

Usual service ports: FTP control 21, SSH 22, Telnet 23, SMTP relay 25, DNS 53, HTTP 80, POP3 110, IMAP 143, HTTPS 443.

Telnet provides unencrypted remote terminals; SSH protects remote access.

Servers can be configured on other ports.

A browser's HTTP exchange does not determine the physical duplex mode of its network interface.

Data Communication and Computer Networking module

NET22 | IPv4 notation and classful defaults

Write IPv4 as four decimal octets 0-255, or as 32 binary bits.

Use weights 128,64,32,16,8,4,2,1 within each octet.

Thus 11000001 10000011 00011011 11111111 is 193.131.27.255.

A value such as 257 is impossible.

Syntactic validity differs from assignability: an address ending .0 or .255 can be a host under some prefixes.

Historical classes use leading patterns: A 0 with /8, B 10 with /16, C 110 with /24; D 1110 is multicast and E 1111 reserved.

Ordinary class-A first octets are 1-126 because 0 and 127 have special uses.

B spans 128-191 and C 192-223.

Default usable counts are 2^24-2, 2^16-2 and 254.

Modern CIDR requires the actual prefix and overrides class defaults.

Data Communication and Computer Networking module

NET23 | Special ranges, private addresses and IPv6

RFC1918 private blocks are 10.0.0.0/8, 172.16.0.0/12 (through 172.31.255.255), and 192.168.0.0/16.

They can be reused internally but are not globally routed as public destinations.

A LAN may also use legitimately assigned public addresses; private is common, not mandatory. 127.0.0.0/8 is IPv4 loopback, commonly 127.0.0.1, for local protocol access without an external link. 255.255.255.255 is limited broadcast; 0.0.0.0 can identify an unconfigured source.

Link-local/APIPA uses 169.254.0.0/16, with edge /24 blocks excluded from normal automatic selection.

IPv6 has 128-bit hexadecimal addresses, flexible prefixes and no IPv4-style broadcast.

Sources, not routers, fragment IPv6 packets.

IPv6 does not automatically encrypt traffic or make authentication unnecessary.

Compare IPv6’s larger 128-bit address space and encryption support separately; specify the property being evaluated.

IETF RFC 1918

IETF RFC 8200

NET24 | Masks, prefixes and binary AND

A /p IPv4 prefix has p leading one bits followed by 32-p zeros.

Partial octet masks are 128,192,224,240,248,252,254. /19 is 255.255.224.0; /20 is 255.255.240.0; /21 is 255.255.248.0, correcting the study guide’s 255.248.0.0 mistake.

Compute network=address AND mask.

Host bits zero identify the block start; host bits one give broadcast.

With 172.16.28.252/20, the third octet is grouped in blocks of 16, so the network is 172.16.16.0 and broadcast 172.16.31.255.

A written host/prefix such as 172.105.180.0/21 need not be an aligned block label: masking yields 172.105.176.0/21.

Data Communication and Computer Networking module

NET25 | Find subnet bounds and usable hosts

For ordinary broadcast LANs let h=32-p. Total addresses=2^h; usable hosts=2^h-2, reserving network and directed broadcast.

Find the changing octet's increment as 256-mask octet, round the address down to a multiple, and extend through the next boundary minus one.

192.168.2.64/26 spans .64-.127; usable .65-.126 gives 62 hosts. 192.168.10.25/28 spans .16-.31, so its network is .16. 172.17.128.0/21 spans 172.17.128.0-172.17.135.255; .128.255 and .135.0 are valid interior hosts, but .135.255 is broadcast. /27 gives 32 total and 30 usable, not 32 usable.

Data Communication and Computer Networking module

NET26 | Borrow bits and choose a mask

To split parent /a into equal /p blocks, borrow b=p-a bits and obtain 2^b subnets.

Do not subtract two subnets under modern practice.

For a Class B /16 parent split to /19, b=3 gives eight subnets, each with 8192 addresses and 8190 conventional hosts.

A /19 alone does not state its parent.

For H hosts, choose the smallest h satisfying 2^h-2≥H, then p=32-h. For 24 hosts use /27; 52 or 57 use /26; 100 use /25; 500 use /23.

For at least six equal subnets choose b=3.

A Class C /24 split to /27 yields eight subnets.

Always check both the number of subnets and the host capacity.

Data Communication and Computer Networking module

NET27 | Design and implement a VLSM address plan

VLSM assigns different masks to different LAN sizes.

Sort demands largest first, round each to a power-of-two address block including reservations, allocate aligned nonoverlapping blocks within the parent, and leave the remaining space explicit.

For a /24 with LAN demands 100,60,29: allocate .0/25 (hosts .1-.126), .128/26 (.129-.190), and .192/27 (.193-.222); .224/27 remains.

Two remote LANs of 100 each fit .0/25 and .128/25 within a separate /24.

Count router interfaces as hosts if not already included in the demand.

Implementation assigns unique usable IPs/masks, puts each default gateway in its local subnet, configures router interfaces/routes and DHCP exclusions, then tests local reachability, gateway reachability, remote routing and DNS separately.

Check duplicate addresses, overlap and return routes before blaming an application.

Data Communication and Computer Networking module

NET28 | Point-to-point links and route aggregation

A conventional /30 has four addresses and two usable endpoints; check the link and addressing assumptions when selecting a WAN prefix.

RFC3021 permits /31 on supported point-to-point links: both addresses are endpoints, so the subtract-two rule does not apply. /32 identifies one address/host route.

Do not claim /30 is universally the smallest valid link prefix.

CIDR, Classless Inter-Domain Routing, replaces rigid class allocation and supports aggregation.

Combine contiguous aligned blocks sharing leading bits; unrelated adjacent-looking labels may not aggregate cleanly.

Four consecutive /26 blocks covering a /24 summarize to /24.

More-specific routes must remain available where the aggregate would otherwise send traffic to the wrong next hop.

IETF RFC 3021

IETF RFC 4632

NET29 | NAT and port translation

Static NAT gives a fixed address mapping; dynamic NAT allocates mappings from a pool.

PAT/NAPT maps address-plus-port combinations so multiple private clients share a public address.

It is translation, not ARP resolution and not encryption.

Return traffic must match the translation state or a configured inbound mapping.

A private host such as 10.0.0.1 commonly reaches the public Internet through NAT at a gateway, together with valid routing and a public external address.

NAT alone does not supply a physical connection or DNS.

A public address does not automatically grant permission to use it; both public and private unicast addresses can be used on a LAN under the correct allocation and subnet.

Data Communication and Computer Networking module

NET30 | Routing algorithms and destination sets

Routing computes paths; forwarding chooses a next hop for an individual packet.

Select the matching route with the longest prefix, then apply the routing system's tie-breaking criteria.

A default /0 matches when nothing more specific does.

Static routes are configured; dynamic protocols exchange reachability.

Distance vector uses neighbor estimates, while link state distributes link information and computes shortest paths.

Interior protocols operate within an autonomous system; BGP exchanges routes between systems.

Flooding sends copies on eligible outgoing links with duplicate/lifetime controls; random walk picks a neighbor probabilistically.

Unicast targets one destination, broadcast all in a scope, multicast a subscribed group and anycast one member sharing an address, normally nearest under routing metrics.

Multi-destination packets can list multiple targets; this is not necessarily identical to broadcast.

Group multicast trees use mechanisms such as reverse-path checks; Ethernet spanning tree is a different layer-2 mechanism.

Data Communication and Computer Networking module

NET31 | Circuit, message and packet switching

Circuit switching sets up a path with reserved resources, transfers data, then releases it.

Predictable allocated bandwidth trades against setup/blocking and wasted reservation during silence.

A traditional telephone call is the standard example.

Packet switching divides data into packets that share links statistically; queueing can add variable delay and loss.

This does not make modern voice-over-packet impossible: buffering, scheduling and capacity planning manage it.

Datagram packets are forwarded independently.

Virtual circuits establish forwarding state, but resource reservation is optional, not a necessary property of every virtual circuit.

Message switching stores a complete message before forwarding, without a dedicated end-to-end path.

Packet store-and-forward waits for the full packet/frame; cut-through starts after enough header arrives; fragment-free historically waits through the initial collision-sensitive portion.

No routing algorithm guarantees that every packet always takes the geometrically shortest route.

Data Communication and Computer Networking module

NET32 | Multiplexing and channelization

Multiplexing combines multiple input streams onto a shared link; demultiplexing separates them at the receiver.

It differs from broadcasting one message to many destinations.

FDM assigns distinct frequency bands with suitable guard spacing; WDM assigns different optical wavelengths.

TDM allocates time slots: synchronous TDM reserves recurrent slots even if an input is idle, while statistical TDM allocates according to demand and needs identification.

FDMA and TDMA allocate frequencies or time among users; CDMA separates coded signals.

Physical multiplexing is different from transport port multiplexing.

Combining signals onto a shared link is multiplexing; distinguish it from point-to-point connectivity, broadcast, synchronization and multicast.

Data Communication and Computer Networking module

NET33 | Identify networking layers and addressing assumptions

First identify the exact layer, protocol and scope.

Flow/error control can exist at both link and transport layers; MAC addresses can be locally administered; bridges and switches both operate at layer 2.

A LAN can use public or private unicast space.

UDP and IP are both connectionless.

Session dialogs and TCP connections are distinct senses of connection.

Distinguish the protocol, layer and scope before treating these descriptions as exclusive.

A host address without a prefix does not determine capacity unless the question explicitly assumes historical classful addressing.

A subnet count needs a parent block.

Excluding subnet zero and the all-ones subnet is an obsolete convention; apply it only when explicitly stated.

Choose a networking concept by its definition rather than by repeated distractors.

Read complete, unambiguous option labels before matching them to networking concepts.

Data Communication and Computer Networking module

NET-S01 | Independent routing classifications

Adaptive or dynamic routing changes routes in response to updated network information; nonadaptive or static routing uses configured choices. Distributed routing means routers perform computations using exchanged information instead of requiring one central route computation. These properties do not imply random next-hop selection: the path-selection rule is a separate classification.

Data Communication and Computer Networking module · page 19

RFC 1058 - Distance-vector computation and topology changes

NET-S02 | Management, traffic service and application transport bindings

SNMP means Simple Network Management Protocol. Its management applications monitor and control managed devices, such as hosts and routers, by exchanging management information with their agents. This device-management role differs from transferring email, resolving names or mapping an IP address to a MAC address.

In the conventional service bindings, FTP control, SMTP mail transfer and Telnet remote terminals use TCP. DNS supports both UDP and TCP on port 53; it is not exclusively a UDP service. The application protocol's service role is distinct from the transport carrying its messages.

Quality of service (QoS) controls traffic treatment to meet objectives such as throughput, delay, jitter or loss. Classification, queue scheduling and allocation of buffer or bandwidth resources implement such treatment. QoS concerns delivered service behavior; scalability concerns accommodating growth, and security concerns unauthorized actions.

RFC 3411 - SNMP architecture, section 1.2

Data Communication and Computer Networking module · page 28

RFC 2475 - Differentiated-services architecture

NET-S03 | Why IPv4 conservation and IPv6 matter

The finite IPv4 address space has faced allocation scarcity. Private addressing with NAT conserves public IPv4 addresses, while IPv6 provides a much larger address space.

Data Communication and Computer Networking module · page 19

NET-S04 | Computer networks and network interfaces

A computer network connects computers or other devices so they can exchange information and share resources such as files, storage and printers. The interconnection uses both hardware and software; neither hardware alone nor software alone is the definition of the whole network.

NIC expands to Network Interface Card. It is the host's network interface, providing the connection through which the host sends and receives network signals. A cable connector attaches the medium to that interface; it is distinct from an intermediary switch or router.

CRC expands to Cyclic Redundancy Check.

Data Communication and Computer Networking module · page 22

Data Communication and Computer Networking module · page 23

Data Communication and Computer Networking module · page 13

NET-S05 | Network-access technologies and geographic use

Ethernet and Token Ring are traditional LAN technologies. X.25 and Frame Relay are traditional WAN technologies. TCP/IP can run over different network-access technologies; its Internet-layer packet service does not require one particular link medium or frame format.

Data Communication and Computer Networking module · page 26

NET-S06 | Interface addresses and transport-number namespaces

An IP address identifies a network interface rather than limiting a whole computer to one address. A computer or router can have several interfaces and addresses. Under a supplied IPv4 prefix, leading bits identify the network and the remaining bits identify an address within that network; a MAC address is a separate link-layer identifier.

TCP and UDP port fields are 16 bits, allowing numbers from 0 through 65535. Well-known ports are 0-1023, registered ports are 1024-49151, and dynamic/private ports are 49152-65535. Client endpoints can use dynamically selected ports instead of every application requiring a well-known assignment. These port numbers are distinct from the IP version and upper-protocol fields.

A bridge or switch's MAC address table is its learned forwarding database: it associates MAC addresses with ports for frame forwarding. It differs from an ARP cache, which associates an IPv4 next hop with a link address, and from a routing table, which selects next hops for IP prefixes.

A common IPv6 unicast subnet uses a 64-bit subnet prefix and a 64-bit interface identifier. This split does not make the complete address 64 bits: it remains 128 bits. Special uses and explicitly different prefix lengths must be considered separately.

Data Communication and Computer Networking module · page 18

Data Communication and Computer Networking module · page 28

Cisco - Configure MAC address learning and forwarding tables

RFC 7421 - Analysis of the 64-bit boundary in IPv6 addressing

NET-S07 | Physical signaling and legacy Ethernet cable roles

Fiber-optic links can use near-infrared light. The selected optical wavelength is part of the link's physical signaling specification, just as connectors and electrical signal levels are physical-interface specifications. This does not require every fiber link to use the same wavelength.

For legacy fixed Ethernet port roles, PCs and router Ethernet endpoints use the same endpoint role, while ordinary switch ports use the opposite role. Thus a direct router-to-PC link uses crossover wiring; PC-to-switch or router-to-switch uses straight-through wiring. Automatic MDI-X can remove the need to choose the crossover manually.

Ethernet switches can have fiber-optic interfaces with suitable optical transceivers. A forwarding role therefore does not impose a copper-only restriction: distinguish what the device does with frames from the physical medium used by its ports.

Intermodulation distortion occurs when multiple signals interact through a nonlinear component and generate unwanted combination-frequency products. Its mechanism differs from random thermal motion, adjacent-channel leakage and an isolated impulse disturbance.

Cisco - Introduction to Optical Fibers, dB, Attenuation and Measurements

Data Communication and Computer Networking module · page 14

Cisco Press - Switch Speeds and Forwarding Methods

Cisco - Configuring interface characteristics and auto-MDI-X

Cisco - Troubleshoot fiber links on Catalyst 9000 series switches

National Instruments - Nonlinearity testing using multitone generation

Data Structure and Algorithms

Complete Study Notes

DSA01 | Start with the data and the required operations

An abstract data type (ADT) defines values and operations; representation supplies implementation.

Arrays or linked nodes can implement the same LIFO stack.

Abstraction keeps relevant details: patient urgency and arrival order matter for service; hair colour does not.

Arrays, linked lists, stacks and queues are linear.

Trees model hierarchies; graphs model general connections.

A record grouping integer, character and floating fields is heterogeneous; an ordinary array has one element type.

Data Structure and Algorithms module

DSA02 | Arrays: indexed access, capacity and shifting

For equal-size elements, address(A[i]) = base + i × element_size under zero-based indexing.

This arithmetic gives constant-time positional access in the usual RAM cost model.

A row-major rectangular matrix uses base + (row × columns + column) × element_size.

A fixed array has a chosen capacity; the choice need not occur at compile time.

A resizable array allocates a larger block and copies when needed.

Inserting in the middle shifts later elements, so it costs linear time.

Appending into spare capacity is constant; a resizing append can be linear, though repeated geometric growth gives amortized constant append cost.

Example

Work it through

- Insert 9 at index 1 of [2,4,6] with a spare slot: shift 6 then 4 right, giving [2,9,4,6].

Data Structure and Algorithms module

DSA03 | Operations, records and memory ownership

Traversal processes every item; searching locates a target; insertion and deletion change membership; updating changes a stored value.

Sorting reorders items, while merging combines sequences.

To merge sorted [1,4] and [2,3], repeatedly emit the smaller front, producing [1,2,3,4].

Destroying a structure releases owned storage.

A C struct groups named fields, including arrays and pointers.

A singly linked node needs payload plus a same-type link.

C fields cannot have initializers inside the type declaration; modern C++ permits default member initializers.

In C++, match new with delete and new[] with delete[].

Repair links before freeing a node; never dereference it afterwards.

Data Structure and Algorithms module

DSA04 | Algorithm properties are separate obligations

Finiteness means termination after finitely many steps for every valid input.

Definiteness means each instruction has one interpretation; effectiveness means each step is executable.

Correctness means meeting the stated result for every valid input.

A finite program can still return the wrong answer.

A problem specifies inputs, outputs and assumptions.

Language independence concerns the procedure, not its implementation.

Efficiency needs a resource model.

An unfamiliar characteristic is not false merely because a teaching list omits it.

Data Structure and Algorithms module

DSA05 | Measure costs before naming their growth

Time counts chosen operations; space counts memory, including live recursive frames.

Distributed algorithms may also measure communication volume.

A single assignment is one unit in the elementary model, not literally one second.

Sequential blocks add costs; independent nested loops multiply their iteration counts.

Conditional worst-case cost includes the test and the more expensive branch.

Best, average and worst compare equal-size inputs; average needs a distribution.

For m operations each with mean cost f(n), expected total is m f(n), without requiring independence.

Exact coefficients depend on which elementary operations count.

Data Structure and Algorithms module

DSA06 | Upper, lower and tight bounds

For eventually nonnegative functions, f(n)=O(g(n)) means f(n)≤c g(n) for some positive c beyond a threshold k. Omega reverses the inequality; Theta requires both.

Little-o means f/g tends to zero; little-omega means it grows without bound.

These relations can describe best-case, average-case, worst-case or space functions.

For n≥1, $3n^2+4n+1 \leq 8n^2$: c=8,k=1 witness $O(n^2)$.

Constants and lower-order terms disappear from tight polynomial growth.

Transitivity multiplies witnesses: f≤c g≤cd h beyond both thresholds.

Example

Work it through

- Increasing representative rates: 1, log n, n, n log n, n^2, n^3, 2^n, 3^n, n!, n^n. Logarithm bases differ only by a constant factor.

Exam tip

Exam trap

- A linear function is also $O(n^2)$. MCQs usually intend the tightest listed order; mixed O and Theta options may overlap.

Data Structure and Algorithms module

DSA07 | Count loops and account for hidden access costs

A loop i=1; i<n; i*=2 takes about $\log_2 n$ iterations.

Two independent n-iteration loops nested around constant work take Theta(n^2).

An n-level ordinary recursion with constant live storage per level needs Theta(n) stack space.

Indexed access is not constant for every List implementation.

Java LinkedList.get(i) walks from the nearer endpoint.

Summing get(0), get(2), get(4), … over a length-N list still gives Theta(N^2), because a linear number of middle indices each require linear traversal.

An iterator scans once in linear time.

An exact executable runtime requires valid code and an explicit cost convention.

Oracle Java SE 21: LinkedList

DSA08 | Understand the linked-node chain

A singly linked node contains data and next.

Head points to the first node; null ends a noncircular chain.

Empty means head==null.

Nodes allocated as needed grow dynamically, with pointer overhead and no direct index arithmetic.

To display or count nodes, follow next from head until null, processing each node.

Counting is Theta(n) unless size is maintained.

An array-linked list still follows links unless separately position-indexed.

Data Structure and Algorithms module

DSA09 | Update links without losing the rest of the list

Prepend with new.next=head; head=new.

To insert after a known predecessor p, set new.next=p.next before p.next=new.

Appending with only head requires a tail traversal; maintaining tail avoids it.

Delete the head by saving old=head, advancing head=head.next and releasing old in C++.

Deleting the tail of a singly linked list requires its predecessor, even when tail itself is known.

Handle empty and single-node lists separately; after removing the only node, set both head and tail to null.

To remove a known successor, bypass it before deallocation.

Clearing an owned n-node chain visits and releases every node.

Reverse a chain with prev=null, cur=head: save next=cur.next; set cur.next=prev; advance prev=cur and cur=next; finish with head=prev.

This is linear time and constant extra pointers.

Example

Work it through

- For 3→7→9, insert 5 after 3: first point 5 at 7, then 3 at 5. Reversing those assignments can lose access to 7→9.

Data Structure and Algorithms module

DSA10 | Doubly linked, circular and practical list choices

Doubly linked nodes add prev; head and tail enable constant-time operations at both ends.

Singly linked tail deletion still needs its predecessor.

A circular tail links back to head; stop traversal on returning to the start, handling empty separately.

Lists suit changing sequences when references to insertion positions are already available.

Finding a position is still linear.

Lists can hold polynomial terms (coefficient, exponent), directory or block links, child lists in nonbinary trees, and hash chains.

Browser history can use a doubly linked cursor.

A playlist may be circular; airline seat lookup requires a specified seat-record model, not an automatic linked-list answer.

Operation	Array	Singly linked list
Access index i	O(1)	O(n) worst
Insert at front	O(n) shifts	O(1)
Append	O(1) with spare capacity	O(1) with tail; O(n) head only
Delete tail	O(1)	O(n), predecessor needed

Data Structure and Algorithms module

DSA11 | Implement a stack with one consistent top convention

Stack policy is LIFO, equivalently FILO.

Push inserts, pop removes/returns top, and peek only observes.

With zero-based occupied slots, top=-1 is empty and top=C-1 is full.

Check before A[++top]=x or returning A[top--].

A count-based alternative uses size=0 and A[size++]=x. Do not combine its empty/full tests with the top-index convention.

Fixed-array push/pop are constant time.

Linked stacks achieve constant time by inserting and deleting at the head; using a singly linked tail for pop would require a predecessor traversal.

Exam tip

Exam trap

- Overflow is an attempted insertion into full storage; underflow is an attempted removal from empty storage. Merely checking isEmpty does not underflow.

Data Structure and Algorithms module

DSA12 | Trace stack operations and reversible actions

Write the stack bottom-to-top after every instruction and record popped values separately.

Push 1, push 2, pop emits 2 and leaves [1].

Push 1, push 2, pop, pop, pop then emits 2,1,1; push 2, pop emits 2.

The complete output is 2,2,1,1,2.

Calls nest on a runtime stack; the latest call returns first.

Bracket matching pushes opening brackets and matches each closing bracket against top.

Reversal pushes input then pops it.

Undo moves the latest action from undo to redo; redo moves it back.

A new edit after undo normally clears redo.

DFS and backtracking use the same latest-pending-choice discipline.

Data Structure and Algorithms module

DSA13 | Infix, postfix and prefix expressions

Infix places an operator between operands and uses precedence and parentheses.

Prefix places it first; postfix places it last.

Their fixed operator arity determines grouping.

To convert infix to postfix, output operands immediately and hold operators on a stack.

Pop higher-precedence operators before pushing a new operator; also pop equal precedence for left-associative operators.

Parentheses temporarily override this order.

To evaluate postfix, push operands.

At a binary operator, pop right operand first, then left, and push left op right.

Build an expression tree for either prefix (preorder) or postfix (postorder); a stack is useful but is not the only possible conversion representation.

Example

Work it through

- A+B*C-D/E becomes ABC*+DE/-. For 8 3 -, pop 3 then 8, producing 5. (A+B)*C becomes AB+C* and prefix *+ABC.

Data Structure and Algorithms module

University of Manchester: Infix, Postfix and Prefix

DSA14 | Queues preserve arrival order

An ordinary queue enqueues at rear, dequeues at front, and peeks at front.

This is FIFO or first-come, first-served.

A linked implementation with head and tail supports both operations in constant time: append at tail, remove at head.

The first insertion sets both pointers; removing the last item clears both.

With head alone, insertion must traverse to the tail.

FIFO suits equal-priority printer jobs, shared-resource waiting, asynchronous buffers and level-order exploration.

A play sequence can also use a queue.

Distinguish airline seat records from waiting requests; an application name alone does not force one representation.

Example

Work it through

- Enqueue 4,8; peek is 4, so a conditional removal only if peek>5 does nothing. Enqueue 1,6 then dequeue: [8,1,6].

Data Structure and Algorithms module

DSA15 | Circular buffers: logical order differs from physical slots

A simple noncircular array queue may report overflow when rear reaches the final index despite released slots at the front.

A circular queue reuses slots with modulo arithmetic.

Use a count convention: front points to the next removal, size counts items, and insertion slot is (front+size)%C. Empty means size==0; full means size==C.

Dequeue reads A[front], sets front=(front+1)%C and decreases size.

An alternative reserves one unused slot and uses next(rear)==front for full; its empty test and capacity differ.

State the convention before tracing.

Example

Work it through

- Full capacity-six [11,13,15,17,19,21], front=0: remove 11, then enqueue 9 into slot 0. Physical array: [9,13,15,17,19,21]; logical front-to-rear queue: [13,15,17,19,21,9]. A noncircular version can instead overflow.

Data Structure and Algorithms module

DSA16 | Deques, merging and splitting queues

A deque permits both-end insertion/removal without guaranteeing middle access.

Endpoint-linked operations are constant but middle indexing is linear.

A circular-array deque may support indexed access.

Dequeue names removal; deque names the structure.

Merging FIFO queues needs an explicit policy: append one sequence after another or alternate their fronts.

Splitting by a predicate preserves each subgroup's original order if each removed item is appended to its matching output queue.

A fixed-capacity buffer also needs a full-policy choice: reject, block or overwrite.

Capacity alone does not choose a policy.

Data Structure and Algorithms module

DSA17 | Priority queues serve by priority, with an explicit tie rule

A priority queue returns the smallest or largest priority key under its convention.

Urgent patients may precede earlier arrivals.

Insertions and extractions can interleave.

Binary heaps support constant peek and logarithmic insert/extract.

Arrays and lists can also implement priority queues: an unordered sequence gives cheap insertion but linear extremum removal; an ordered sequence reverses that tradeoff.

Equal priorities have no inherent FIFO guarantee.

If stable service is required, compare (priority, arrival_counter).

Choose the priority direction and tie rule before tracing.

Exam tip

Exam trap

- FIFO ordering among equal priorities requires an explicit stability policy. That is an additional policy, not a universal priority-queue rule.

Sedgewick and Wayne: Priority Queues

DSA18 | Tree vocabulary and the different binary shapes

A rooted tree has one parentless root and one parent per other node.

It is connected and acyclic, has n-1 edges, and one path between any pair.

Depth counts root-to-node edges; height is maximum depth.

A single-node tree therefore has height zero.

Leaves have no children.

Binary means at most two children.

Full means each node has zero or two children.

Complete means all earlier levels are filled and the last fills from left.

Perfect means all leaves share a level and every internal node has two children.

Balanced means height stays logarithmic under a stated rule; it does not require a perfect shape.

Data Structure and Algorithms module

DSA19 | BST search follows a global ordering invariant

With distinct keys, every key in a node's left subtree is smaller and every key in its right subtree is larger.

Immediate-child comparisons alone are insufficient: root 10, left child 5, and 5's right child 12 violate the root's restriction.

Duplicates require a separate count or placement policy.

Search compares at each node and follows one child until equality or null.

Insertion uses that same path to attach a new leaf.

Operations cost O(height): logarithmic for balanced trees, linear for a chain.

Best-case successful search is Theta(1), a root hit.

An optimal BST minimizes expected search cost using access probabilities; constructing it is a different problem from searching it.

Standard optimal-BST dynamic programming tries each possible root for each key interval: $O(n^2)$ intervals times O(n) roots gives $O(n^3)$ construction.

It minimizes probability-weighted search depth, not necessarily worst-case height.

Example

Work it through

- Insert 40,20,50,10,30,60: root children are 20 and 50; 20 has 10,30; 50 has right child 60. Height is two edges. Sorted insertions 1,2,3,4,5 instead form a right chain.

Data Structure and Algorithms module

University of Freiburg: Optimal Binary Search Trees

DSA20 | Tree traversal is a rule you can execute

Preorder visits node,left,right; inorder left,node,right; postorder left,right,node.

Apply recursively; null children do nothing.

Inorder returns sorted keys only for a BST.

Preorder supports copying, postorder supports deleting children before parents, and level order uses a FIFO queue.

For root 4, children 2 and 6, and leaves 1,3,5,7: preorder is 4,2,1,3,6,5,7; inorder is 1,2,3,4,5,6,7; postorder is 1,3,2,5,7,6,4.

Level order is 4,2,6,1,3,5,7.

Randomized visitation is possible; it is simply not one of these standard named depth-first orders.

Data Structure and Algorithms module

DSA21 | BST deletion preserves every subtree restriction

A leaf can be detached by clearing its parent link.

For one child, make the parent bypass the node and point to the child.

For two children, copy the inorder predecessor (left-subtree maximum) or successor (right-subtree minimum), then delete that replacement from its original location.

That second deletion has at most one child.

Deletion by merging is another approach.

Promote the right subtree and attach the entire old left subtree to the minimum node in the promoted right subtree; or use the symmetric left promotion.

Update root itself when deleting root.

Copying values and reconnecting subtrees can yield different valid shapes, so a trace must name its deletion method.

Data Structure and Algorithms module

DSA22 | Balanced trees prevent the sorted-input chain

AVL balance factor is height(left)-height(right); every node must have factor -1,0 or 1.

After insertion or deletion, update heights upward and rotate where necessary.

LL imbalance needs a right rotation; RR needs left; LR rotates the left child left then the node right; RL is symmetric.

Rotations preserve inorder key order.

Inserting 30,20,10 triggers LL repair: 20 becomes root with children 10,30.

Red-black colouring rules provide another logarithmic-height discipline.

Both prevent linear worst-case search; balanced trees also support ordered ranges unlike ordinary hashing.

Sedgewick and Wayne: Balanced Search Trees

DSA23 | A heap combines completeness with a weaker ordering

A binary max-heap is complete with each parent at least its children; a min-heap reverses this order.

Unlike BSTs, heaps do not order entire subtrees.

The root is maximum.

If all keys are distinct, the second largest is a root child.

With duplicate maxima, the second largest distinct value may be deeper: [9,9,9,8,7,6,5] is a valid max-heap and that value is 8 at index 3.

The minimum may be any leaf, requiring linear worst-case scanning.

Zero-based array representation: root index 0, left=2i+1, right=2i+2, parent=floor((i-1)/2) for i>0.

One-based representation uses children 2i,2i+1 and parent floor(i/2).

A child exists only if its index is inside the current heap size.

Exam tip

Exam trap

- The right-child formula 2i+1 assumes one-based indexing. For zero-based indexing with the root at index 0, the right-child formula is 2i+2.

Data Structure and Algorithms module

DSA24 | Restore a heap after insertion or removal

Insert at the next complete-tree position and swap upward while parent order is violated.

To remove the root, move the final element to root, reduce heap size, then swap downward with the more dominant child until order holds.

Each path has logarithmic length.

Building bottom-up by sinking internal nodes takes linear total time; repeated independent insertions give an O(n log n) construction bound.

Example

Work it through

- Max-heap [10,8,6,2,1,4,5]: remove 10, move 5 to root and sink, giving [8,5,6,2,1,4]. Remove 8, move 4 up then exchange with 6: [6,5,4,2,1].

Data Structure and Algorithms module

Sedgewick and Wayne: Priority Queues

DSA25 | Hash addresses, uniformity and random function choice

Direct addressing reserves a position for every possible key in a finite universe U, costing Theta(|U|) slots.

A hash table uses h(key) in 0..m-1 and stores only actual entries plus bucket capacity.

With h(k)=k mod 6, key 36 has home address 0; collisions may move its final storage position.

Determinism means repeated equal keys get the same address for the chosen function.

Uniformity means expected keys spread evenly.

Universal hashing chooses one function randomly from a suitable family for the table, independently of a fixed key set; it does not choose a fresh unrelated function for each lookup.

Random choice controls collision probability, not a promise of zero collisions.

MIT 6.006: Recitation4 Hashing

DSA26 | Resolve collisions and watch the load factor

Separate chaining stores colliding entries in a bucket list.

Load alpha=n/m is average chain length; expected operations are O(1+alpha) under suitable distribution assumptions, with O(n) worst cases.

Open addressing keeps entries inside the array and probes until a key or an available slot is found.

Linear probing tries (h(k)+j)%m. For size 7 and keys 10,17,24, all home addresses are 3 and occupied positions become 3,4,5.

Quadratic probing changes the step; double hashing uses h1(k)+j h2(k), modulo m, with a step relatively prime to m to cover all slots.

Neither method guarantees no collisions.

Deletion needs tombstones or reinsertion so searches do not stop prematurely.

At h(k)=3 in a seven-slot linear-probing table, addresses wrap 3,4,5,6,0,1,2.

A home address is not guaranteed to remain the final slot.

Sedgewick and Wayne: Hash Tables

DSA27 | Maps, sets and caches specify different contracts

A map associates unique keys with values; different keys may share values, and entries can be traversed.

A set forbids duplicate elements.

Null and iteration-order policies depend on implementation.

Lists preserve positions and generally allow duplicates; Java LinkedList also permits null.

A dictionary or compiler symbol table can use hashing for expected constant lookup, or a balanced tree for logarithmic lookup with ordering.

A bounded cache also needs eviction rules: FIFO uses a queue; LRU can combine hash lookup with a doubly linked recency list.

Move a hit to the recent end and evict the least-recent end when full.

No storage structure is universally best without operation and policy requirements.

Oracle Java SE 21: Map

Oracle Java SE 21: Set

Oracle Java SE 21: LinkedList

Oracle Java SE 21: LinkedHashMap

DSA28 | Linear search: a scan with an early stop

Compare the target with successive elements until equal or exhausted.

Unsorted data is allowed; sorted data is allowed too.

A first-element match costs Theta(1).

An absent target or last-position match needs n comparisons, giving Theta(n) worst time.

Uniformly likely successful positions average (n+1)/2 comparisons, still linear.

For a single unsorted query, sorting first is rarely justified.

Small lists may favour scanning because setup matters.

Sorting does not avoid examining all values for mean, variance or arbitrary parity counts.

Data Structure and Algorithms module

DSA29 | Binary search: maintain a shrinking ordered interval

On an ascending random-access sequence, set lo=0, hi=n-1.

Compute mid=lo+(hi-lo)//2.

Equality finishes; if target<A[mid], set hi=mid-1; otherwise set lo=mid+1.

Stop unsuccessfully when lo>hi.

Reverse comparisons for descending order.

Comparable keys and valid finite bounds matter; fixed allocation before population is not an extra algorithmic requirement.

Worst probes are floor($\log_2 n$)+1 for n>0: seven for 100 keys, about twenty for one million.

Iteration uses constant space; recursion uses logarithmic stack depth.

Linked midpoint traversal loses logarithmic total time.

Boundary variants retain a candidate rather than stopping at equality.

Example

Work it through

- Find 19 in [11,13,15,17,19,21]: mid=2 gives 15; search indices 3..5; mid=4 gives 19. Binary debugging needs an ordered good/bad boundary, not arbitrary fault locations.

Data Structure and Algorithms module

DSA30 | Search variants need their own assumptions

Interpolation search estimates the index from numeric values: lo+(hi-lo)×(key-A[lo])/(A[hi]-A[lo]), rounded to a valid index.

Guard equal endpoints and out-of-range targets.

On sorted, suitably uniform data its expected probes can be O(log log n); pathological distributions can require linear time.

Binary search gives a logarithmic worst bound without that distribution assumption.

Sorted-array ternary search compares two split points and keeps the relevant third, still logarithmic.

Jump search probes block boundaries then scans within a block.

Regular intervals could mean unit-step scanning or larger jumps.

Interpolation also needs sortedness, so prerequisite questions can overlap binary search.

University of Waterloo, Douglas Harder: Interpolation Search

University of Massachusetts: Running Time lecture

University of Waterloo, Douglas Harder: Jump Search

DSA31 | Compare sorting rules, not just algorithm names

Stable sorting preserves the input order of equal keys.

An in-place element algorithm uses a fixed amount of extra element storage; a recursive call stack may still matter when total auxiliary space is requested.

There is no single algorithm identified by stable or in-place alone.

State the implementation, input case and resource being compared.

Method	Best / average / worst time	Extra storage; usual stability
Bubble with early stop	n / n^2 / n^2	constant; stable
Selection	n^2 / n^2 / n^2	constant; unstable

Method	Best / average / worst time	Extra storage; usual stability
Insertion	$n / n^2 / n^2$	constant; stable
Quicksort	$n \log n / n \log n / n^2$	stack usually log n average, n worst; unstable
Ordinary array mergesort	n log n in all cases	linear; stable with left-first ties
Heapsort	O(n log n) bound in all cases	constant with iterative sink; unstable

Exam tip

Exam trap

- Big-O bounds can overlap. $n \log(n^2)=2n \log n$, so it has the same growth as n log n, unlike $n(\log n)^2$.

Data Structure and Algorithms module

DSA32 | Bubble sort: complete adjacent passes

For ascending order, compare each adjacent pair and swap only when left>right.

The largest remaining value moves to the final unsorted position after one left-to-right pass.

Shrink the next pass by one position.

Quadratic worst/average work comes from up to n(n-1)/2 pair comparisons.

Early stopping makes a sorted input linear: one no-swap pass finishes.

A fixed pass schedule stays quadratic.

Strict swaps preserve equal-key order.

Cocktail sort runs bubble passes in both directions and is stable with strict swaps.

Example

Work it through

- [4,1,3,2] → [1,4,3,2] → [1,3,4,2] → [1,3,2,4] after the first pass.

Data Structure and Algorithms module

DSA33 | Selection sort: minimize exchanges, not comparisons

At position i, scan the remaining suffix for its minimum, then exchange it with A[i].

There are n-1 meaningful placements and n(n-1)/2 comparisons regardless of initial order.

At most n-1 exchanges of distinct positions are needed.

A loop that also swaps the last item with itself can count n swap calls.

Selection suits expensive record moves with cheap key comparisons, not every large dataset.

It is unstable: exchanging first 2a with 1 in [2a,2b,1] gives [1,2b,2a].

Constant extra element storage is shared by several sorts.

Data Structure and Algorithms module

DSA34 | Insertion sort: extend a sorted prefix

Begin with the first element already sorted.

For each remaining item, save its value, shift larger preceding values right, and insert it into the gap.

That gives n-1 insertion passes.

On sorted input each pass needs one ordering check and no shifts: linear time.

Reverse order causes 1+2+…+(n-1) shifts: quadratic time.

Few inversions mean few shifts, favouring insertion for nearly sorted or small arrays.

Shift only strictly larger keys for stability.

A linked implementation saves array shifts but still must locate each insertion position.

Example

Work it through

- Insert 2 into prefix [1,4,7]: save 2, move 7 then 4 right, and write 2 after 1. The sorted prefix becomes [1,2,4,7].

Data Structure and Algorithms module

DSA35 | Shell sort: insertion across decreasing gaps

Choose a sequence of positive gaps ending at 1.

For each gap h, insertion-sort each subsequence of positions h apart.

The final h=1 pass finishes ordinary sorting after earlier long moves have reduced disorder.

An h-sorted array satisfies A[j]≤A[j+h] wherever both indices exist.

The O(n^(3/2)) bound depends on gaps; Knuth increments 1,4,13,… have that worst order.

Shell sort is generally unstable because distant moves can reorder ties; extra element storage is constant.

Example

Work it through

- For [5,4,3,2,1], gap 3 sorts positions (0,3) and (1,4), giving [2,1,3,5,4]. Gap 1 then produces [1,2,3,4,5].

Data Structure and Algorithms module

Sedgewick and Wayne: Shell.java

DSA36 | Quicksort: separate partition guarantees from exact traces

Choose a pivot, partition, then recursively sort the pieces.

Balanced partitions give n log n work; repeated extreme pivots sum n+(n-1)+... to quadratic work.

Expected n log n needs an input/random-pivot model.

Random and median-of-three choices have different tradeoffs; neither is universally best.

In the ten-distinct-value example, a final pivot 5 has five smaller values before it and four larger after it.

Internal orders depend on the partition algorithm; several candidate arrays can satisfy the invariant.

State the algorithm before predicting one exact array.

Quicksort normally rearranges in the same array but recursive storage can be linear in the worst case.

Data Structure and Algorithms module

DSA37 | Merge sort and internal versus external sorting

Split into halves, sort them, then merge by advancing the smaller current front.

With left-first tie handling, equal keys retain order.

Ordinary array merging needs a linear auxiliary buffer.

There are logarithmically many levels with linear merge work per level, giving n log n time even on sorted input unless an optimization skips merges.

Top-down mergesort recurses; bottom-up merges runs of widths 1,2,4,... iteratively.

In-place merging variants exist with different tradeoffs; linear merge names an operation, not a universally standardized sort variant.

Internal sorting fits the working dataset in memory.

External sorting builds sorted runs and merges buffered streams from disk, aiming to reduce I/O. Reverse order alone does not characterize maximum merge comparisons.

Data Structure and Algorithms module

Virginia Tech OpenDSA: External Sorting

DSA38 | Heapsort: repeatedly place the extreme at the end

For ascending in-place heapsort, build a max-heap over the array.

Exchange root with the final active element, shrink the active heap, and sink the new root.

The removed maximum now belongs in the sorted suffix.

Repeat until one active item remains.

Building with bottom-up sinking is linear; repeated extraction gives O(n log n) worst time.

Iterative sinking needs constant auxiliary storage.

The method is generally unstable because long root-to-end exchanges can reorder equal keys.

It is not uniquely the method for sorting by priorities: any comparison sort can compare a priority field.

Heap extraction is its distinctive mechanism.

Data Structure and Algorithms module

DSA39 | Radix sorting uses key digits, not arbitrary magic

LSD radix sorting handles fixed-width digit keys by stably sorting least-significant digit first, then moving left.

Stable intermediate passes preserve the ordering already established by less-significant digits.

With d digits and radix r, counting-based passes take O(d(n+r)) time and O(n+r) auxiliary storage.

MSD begins at leading digits and subdivides equal-prefix groups.

Strings are lexicographically comparable.

Radix methods require accessible digits and length/signed-value conventions, not arbitrary unstructured keys.

Large radix, long keys or small n may favour comparison sorting.

Sedgewick and Wayne: LSD.java

DSA40 | Graphs: represent the relationships you actually need

A graph has vertices V and edges E. Directed edges distinguish u→v from v→u; undirected edges represent two-way relationships.

Following accounts is directed; a mutual friendship relation is undirected.

Weighted edges carry costs; unweighted shortest-path questions count edges.

DAG expands to directed acyclic graph.

Trees are connected acyclic graphs, although graph data structures may also contain cycles.

A matrix uses Theta(V^2) space and constant edge lookup.

Adjacency lists use Theta(V+E) space and enumerate neighbours.

Dense simple graphs approach quadratic edge counts.

Examining every edge costs more on denser graphs; vertex-only operations need not inspect every edge.

Data Structure and Algorithms module

DSA41 | BFS: discover one distance layer at a time

Mark the source when enqueuing it.

Repeatedly dequeue a vertex, examine its neighbours, and enqueue each unvisited neighbour while marking it.

Store predecessor[v] when discovering v if a path must be reconstructed.

FIFO ensures vertices at distance d are explored before later distance layers.

BFS finds minimum-edge paths in unweighted graphs, not arbitrary minimum-weight paths.

With adjacency lists it costs O(V+E); scanning a matrix costs $O(V^2)$.

To traverse a disconnected graph, restart from each unvisited vertex.

Neighbour order affects tied traversal order but not minimum distances.

Example

Work it through

- Edges A-B, A-C, B-D, C-D; alphabetical neighbours: queue [A] → [B,C] → [C,D] → [D] → []. Discovery order A,B,C,D; D has distance two.

Sedgewick and Wayne: Undirected Graphs

DSA42 | DFS: finish a branch before returning

Mark a vertex, then recursively visit each unmarked neighbour before continuing with later neighbours.

An explicit stack can store pending work instead.

Both implement latest-pending-work first; a queue would change that discipline.

Marking prevents infinite repetition around graph cycles.

Using the same graph and alphabetical neighbours, recursive DFS can discover A,B,D,C. Its path is a valid reachability witness but not necessarily shortest.

Adjacency-list cost is O(V+E), with O(V) visited state and possible stack depth.

DFS supports cycle and connectivity analysis; a directed cycle test distinguishes active recursion vertices from already completed ones.

Data Structure and Algorithms module

DSA43 | Shortest paths and spanning trees solve different objectives

For nonnegative weights, standard Dijkstra initializes source distance zero and others infinity, repeatedly finalizes the unsettled vertex with smallest distance, and relaxes each outgoing edge: dist[v]=min(dist[v],dist[u]+w(u,v)).

A min-priority queue supports the next choice.

BFS is enough when every edge has equal cost; negative edges require another justified method, such as Bellman-Ford.

An MST minimizes the total cost of connecting all vertices of a connected undirected weighted graph.

Prim adds the cheapest edge crossing from its current tree to an outside vertex.

Kruskal processes edges by increasing weight and rejects those joining vertices already connected.

An MST need not give shortest source-to-vertex routes, and equal weights can permit several MSTs.

Example

Work it through

- Triangle weights AB=1, BC=2, AC=10: Prim or Kruskal selects AB,BC, total 3. Dijkstra from A gets B=1 and improves C from 10 to 3 through B.

Sedgewick and Wayne: Shortest Paths

Sedgewick and Wayne: Minimum Spanning Trees

DSA44 | Recognize two other tested algorithms through their rules

Kadane’s algorithm finds a contiguous maximum-sum subarray.

For a required nonempty answer, initialize ending=best=A[0]; update ending=max(x,ending+x), then best=max(best,ending).

It is linear time with constant working storage.

Initializing best=0 instead silently allows an empty answer and fails the nonempty all-negative case.

Knuth-Morris-Pratt (KMP) finds a pattern inside text.

Preprocess the pattern’s longest proper prefix that is also a suffix.

After mismatch, fall back using those prefix lengths rather than restarting comparisons from the beginning; the text pointer need not move backward.

Pattern preprocessing plus scanning takes O(pattern length+text length).

Example

Work it through

- For [-2,3,-1,4,-5], Kadane ending values are -2,3,2,6,1, so best is 6 from [3,-1,4]. Pattern ABAB has prefix lengths [0,0,1,2].

University of Alberta: Advanced Algorithms Lecture2

Jayadev Misra, University of Texas: Knuth-Morris-Pratt Algorithm

DSA45 | State trace data and representation assumptions

A trace needs the complete operation sequence, initial state, capacity and output orientation.

State these assumptions before calculating.

Evaluate all choices against the rules; a selected answer does not make other valid choices false.

Specify whether a question asks which structure can implement something or which is best under stated costs.

ADTs do not force one representation, and serial-access memory is not inherently LIFO.

A heap file is an unordered database record file, unrelated to a binary heap: insertion uses available space, scanning can find records, and ordered output requires sorting or an index.

It need not insert at the beginning.

Data Structure and Algorithms module

UC Berkeley CS186: Disk and Files

DSA-S01 | Best, average and worst compare equal-size inputs

Fix the input size before comparing cases. Best-case cost is the minimum resource use over legal inputs of that size; worst-case cost is the maximum. Average-case cost averages costs using a stated input distribution. These three input-case functions can each be described using upper, lower or tight asymptotic bounds; the case name does not select the bound notation.

For successful binary search with all stored positions equally likely, average probes have order Theta(log n). The probe decisions form a binary tree: only a bounded number of positions occur in shallow levels, while a constant fraction require logarithmic depth. Interval halving also gives the logarithmic upper bound. A distribution concentrating on an immediate midpoint hit would give a different average.

NIST Dictionary of Algorithms and Data Structures - Best case

NIST Dictionary of Algorithms and Data Structures - Worst case

NIST Dictionary of Algorithms and Data Structures - Average case

Sedgewick and Wayne - Algorithms, binary-search symbol-table average costs

DSA-S02 | Queues for deferred interrupt work

An interrupt service routine may need to record work quickly and defer its completion. Operating systems can queue that deferred work for later processing; Windows deferred procedure calls are one concrete example. This is a queue application, but it does not imply that every interrupt system uses a single FIFO service policy.

Microsoft - Introduction to DPC objects

DSA-S03 | Linked nodes and multiway tree branching

A linked binary-tree node stores a payload and two child pointers, one for the left child and one for the right child. A null child pointer means that child is absent; a root pointer gives access to the whole tree. In a dynamically allocated version, insertion can allocate a node and attach it through the appropriate child link.

A B-tree is a balanced multiway search tree. Its allowed branching factor can exceed two, so a general B-tree is not the binary-tree family. Balanced describes a height discipline; it does not by itself restrict every node to two children.

Data Structure and Algorithms module · page 9

Data Structure and Algorithms module · page 5

NIST Dictionary of Algorithms and Data Structures - B-tree

DSA-S04 | Recognize recursive divide-and-conquer structure

Divide and conquer reduces a problem to smaller instances of the same kind, solves the required smaller instances and obtains the original result from them. Mergesort recursively sorts two halves and merges; quicksort partitions and recursively sorts the resulting ranges. This course also classifies binary search as divide and conquer: an ordered comparison selects the only half that can contain the target. This is different from simply repeating adjacent swaps, choosing successive minima, or making stable LSD digit passes.

Data Structure and Algorithms module · page 11

Data Structure and Algorithms module · page 3

Web Programming

Complete Study Notes

WEB01 | Browser, server and technology roles

The Internet connects networks; the Web is a resource system commonly using HTTP.

A browser/user agent requests resources and renders HTML/CSS, then runs permitted JavaScript.

A web/HTTP server receives requests and returns representations, sometimes invoking application code.

Client and server are roles and can run on the same machine.

HTML supplies structure/meaning; CSS controls presentation; JavaScript supplies programmable interactivity.

PHP, Python, JSP/Java and ASP.NET applications generate server responses and access databases; C# is commonly used with ASP.NET.

Node.js is a JavaScript runtime, not a separate language.

JavaScript can run on either side.

Browser scripts are sandboxed, but may access selected files/devices through permission-based APIs.

They are not inherently limited to trivial programs, nor can they freely access arbitrary client hardware.

Web Programming module

WEB02 | Hosting, URLs and static versus dynamic content

Hosting supplies an environment where a site is served; domain registration reserves a name, and DNS maps names to network records.

An ISP provides connectivity.

A URL identifies scheme, host, optional port, path, query and fragment; the host identifies the machine/service name, while a fragment selects a local document location.

A basic static server returns stored files; dynamic processing constructs a representation from logic, requests or data.

Static pages can still run interactive browser scripts, and dynamic generation need not change visible text every time.

A website groups resources; a page is one document/resource.

Open standards do not guarantee identical support for every feature in every browser.

Historical editor-name items refer to Microsoft FrontPage and Macromedia HomeSite5, not languages or required modern tools.

Web Programming module

Adobe/Macromedia HomeSite support archive

Microsoft IE5 Corporate Deployment Guide

WEB03 | Read an HTTP/1.1 exchange

An HTTP/1.1 request begins with method, request-target and version, followed by header fields, a blank line, and an optional body.

For example, GET /courses?id=7 HTTP/1.1 followed by Host: example.edu and a terminating blank line can have no body.

A response begins with version, status code and optional reason phrase, then headers, blank line and permitted content.

Request lines are not response status lines.

Content-Type describes representation format; Content-Length or transfer framing establishes message boundaries when applicable.

HTTP is stateless in its request semantics, but applications maintain state through sessions/databases.

HTTP2/3 encode messages differently; a textual request-line description applies to HTTP/1.x.

Asynchronous use is a client programming choice, not proof that HTTP has no responses.

IETF RFC 9112

WEB04 | Methods, status and GET/POST reasoning

GET retrieves a representation with safe semantics; do not implement an intentional state-changing action as a GET link.

POST asks the target to process supplied content.

PUT creates/replaces the identified resource representation; DELETE requests removal of its association.

Safe describes intended state effects; idempotent means repeating the same request has the same intended effect, not necessarily the same response.

GET/PUT/DELETE are idempotent; arbitrary POST is not.

200 indicates success, 201 creation, 204 success without content, 301 permanent redirection, 400 malformed request, 401 missing/invalid authentication, 403 refusal, 404 unavailable/not found, and 500 an unexpected server failure.

GET forms usually put name/value pairs in the URL query; POST forms put them in the body.

Neither method encrypts data.

POST bodies are visible to developer tools and subject to server/configured size limits.

IETF RFC 9110

WEB05 | HTTPS and resource-oriented APIs

HTTPS uses HTTP over TLS, protecting channel confidentiality/integrity and authenticating the server under certificate checks.

It does not make unvalidated input safe or guarantee a trustworthy merchant.

A RESTful API follows constraints such as client/server separation, stateless requests, cacheability and a uniform resource interface.

Resources have identifiers and representations; JSON is common but not required.

Merely returning JSON does not prove REST.

For a resource /students/7, GET reads it, PUT can replace it, and DELETE requests removal, subject to authentication/authorization and validation.

Represent errors consistently and evaluate retry safety before repeating a request.

Store durable application facts server-side; stateless API requests can still operate on persistent database state.

Web Programming module

WEB06 | HTML document, text and metadata

HTML means HyperText Markup Language. <!DOCTYPE html> selects modern standards mode; <html> is the document root, <head> holds metadata, and <body> contains document content. <title> supplies the page/tab title.

Meta charset declares encoding; description summarizes the page; viewport controls mobile viewport behavior.

Heading levels h1-h6 express hierarchy, with h6 conventionally smallest in default styling, not a mandatory CSS size.

<p> marks paragraphs;
 is a line break and <hr> a thematic break. <pre> preserves source whitespace; <code> marks code. indicates importance and stress; <b>/<i> have other text semantics and are not identical to importance/emphasis. <sub>/<sup> mark sub/superscripts. <!-- comment --> is not displayed but remains visible in source.

Entities include ©, <, > and &.

Tags mark boundaries; an element includes content and attributes, or is a void element such as img.

Web Programming module

WEB07 | Links, images and accessible meaning

Use Read for a link; href is the destination, not url/link.

A fragment such as href="#details" links to an element with id="details". target="_self" uses the current context, _blank a new context, _parent its parent and _top the top-level context.

Browser preferences determine tab versus window.

<img src="photo.jpg" alt="Students in a lab"> supplies source and alternative text.

Use empty alt for purely decorative images; informative images need meaningful alternatives.

Width/height reserve dimensions; title is not a replacement for alt.

An anchor may wrap an image.

Give controls programmatic labels, clear link text and keyboard access; visual styling alone does not establish accessibility.

Attributes belong in opening tags, with unique IDs when used for document identification.

Web Programming module

WEB08 | Lists, tables and navigation

Unordered <ul> and ordered <ol> contain <li> items.

Ordered lists can specify start and numbering type; use CSS list-style-type for presentation.

A definition list uses <dl>, <dt> terms and <dd> descriptions.

Navigation can be a semantic <nav> containing a list of links.

<table> contains rows <tr>, header cells <th> and data cells <td>; a td is not the table itself. colspan spans columns; rowspan spans rows.

Account for occupied cells before placing the next cell, and use header associations for accessible data.

CSS handles borders, background, cell padding and collapsed borders; legacy bgcolor/bordercolor are not preferred styling.

Resetting nav ul with list-style-type:none; margin:0; padding:0 removes bullets and default spacing only; it does not itself arrange links horizontally or create a dropdown.

Web Programming module

WEB09 | Form controls and successful submission

A form’s action identifies its processing URL, method its submission method, and enctype its encoding.

The default method is GET.

Named enabled successful controls contribute values: unchecked checkboxes/radios are usually absent.

A label’s for matches the control’s id; name identifies its submitted field.

Same-name radio buttons form a single-choice group; checkboxes permit independent selections. <select> contains <option>; textarea accepts multiple lines.

For a text input, size indicates approximate visible width, whereas maxlength limits entered text length.

Neither means a server security bound. type="password" masks display but does not encrypt. submit submits; button has no submission action by itself. type=email/required give browser validation feedback; arbitrary clients can bypass it.

No GET/POST method universally means “parse one element at a time.”

Web Programming module

WHATWG HTML input standard

WEB10 | Frames, media and graphics

Legacy frameset/frame divides a window into separate documents; frameset replaced body in old frameset documents.

Named targets controlled which frame changed; independent scrolling could leave other frames stationary.

These elements are obsolete in modern HTML.

An iframe embeds a document within an ordinary page and needs appropriate permissions/sandboxing for untrusted content.

Modern audio/video elements embed supported media, often with controls and fallback content; embed/object can also embed supported resources.

Several tags can embed supported video resources; distinguish their semantics and browser support.

Old bgsound or Flash examples do not imply current browser support.

Canvas provides script-driven bitmap drawing via a context; SVG describes scalable shapes in the DOM.

For a simple canvas task, obtain getContext("2d"), then use fillRect(x,y,w,h); line/path methods require beginPath/moveTo/lineTo/stroke.

Preserve a text alternative where graphics convey information.

Web Programming module

WEB11 | CSS attachment and selector meaning

An external sheet uses <link rel="stylesheet" href="styles.css"> in head; an internal sheet uses style; an inline declaration uses an element's style attribute.

External CSS promotes reuse, consistent design and caching; it does not guarantee that every page becomes smaller or accessible.

A rule is selector { property: value; }.

Comments use /* ... */.

Type p selects paragraph elements, .notice selects a class, #logo an ID, h1,h2 groups selectors, nav ul selects descendants and nav>ul direct children.

Attribute selectors match properties such as [type="email"], pseudo-classes states such as :hover/:focus, and pseudo-elements parts such as ::before.

A selector chooses where declarations apply; it is not itself the property that sets a color.

Web Programming module

WEB12 | Resolve cascade, specificity and inheritance

First compare applicable declarations in their cascade origin, importance and layer context; then compare specificity and order within the relevant context.

In an ordinary same-origin/same-layer author example, an ID outweighs class/attribute/pseudo-class selectors, which outweigh type/pseudo-element selectors.

Compare counts lexicographically, not as an unlimited base-ten total.

Equally specific applicable rules use later source order.

An important declaration outranks a competing normal declaration in that context.

For <p id="intro" class="note">, #intro{color:blue} beats .note{color:red}.

A competing p{color:green!important} beats those normal declarations.

Inherited color/fonts apply when the element lacks a stronger specified value; margin/padding/borders do not normally inherit. :not() contributes the specificity of its argument rather than universally zero.

Web Programming module

Mozilla MDN specificity

WEB13 | Style properties and box calculations

color sets text color; background-color sets background; font-size size, font-family face, font-style italic, font-weight weight, text-align inline-content alignment, and text-decoration effects such as underline.

Margin lies outside the border; padding lies inside; border surrounds content/padding.

With content-box width200px, horizontal padding10px and borders2px, border-box width is224px, excluding margins.

With border-box width200px, content width is176px.

Shorthand values follow top/right/bottom/left: one applies everywhere, two vertical/horizontal, three top/horizontal/bottom, four each side.

Adjacent vertical block margins can collapse rather than add; padding does not collapse.

Non-replaced inline elements do not obey width/height like blocks.

A 1rem length uses root font size; font-size in em uses parent font size, whereas many other em lengths use the element's computed font size.

In historical markup, the body text attribute controlled default text color, while bgcolor controlled background.

Current CSS uses color and background-color instead.

Web Programming module

WEB14 | Layout, alignment and responsive design

Normal flow lays out blocks/inlines.

Flexbox arranges direct children on a main axis; justify-content follows that axis, align-items the cross axis.

With row direction in ordinary horizontal writing, cross-axis alignment is vertical.

Grid organizes rows and columns with explicit tracks, such as grid-template-columns:1fr 2fr. vertical-align applies to inline/table-cell contexts and does not center every block. align-content distributes tracks/lines when there is applicable extra space, not arbitrary single-child centering.

Relative positioning offsets a box while retaining its flow space; absolute removes it and uses its containing block, often established by a positioned ancestor; fixed commonly uses the viewport.

Responsive design combines flexible layout, images, viewport metadata and media queries, for example @media(max-width:600px){.cards{grid-template-columns:1fr}}.

A min-width768px query targets wider viewports, not mobile alone.

Browser compatibility requires feature testing and fallbacks; responsive size handling is a different issue.

Web Programming module

WEB15 | JavaScript syntax, inclusion and declarations

Use <script src="app.js"></script> for an external .js file; script content has no HTML closing-tag syntax inside its JavaScript source.

A deferred script runs after parsing and before DOMContentLoaded.

JavaScript originated at Netscape; it is distinct from Java and supports sophisticated applications through prototype-based objects and first-class functions.

Identifiers are case-sensitive and may start with letters, underscore or $, not digits or spaces. var has function/global scope; let/const are block-scoped. const prevents rebinding but not mutation of an object's contents. // comments end at line end; /* */ span a block.

Semicolons delimit statements, although automatic insertion exists.

Whitespace usually separates tokens but newlines can affect parsing, especially after return. "use strict" is a directive string, not a reserved statement keyword.

Parse syntax before computing outputs.

Web Programming module

WEB16 | Types, conversions and missing values

JavaScript primitives are undefined, null, Boolean, Number, BigInt, String and Symbol; objects include arrays, functions and dates.

Float is not a separate Number primitive type.

Arrays are objects, even though informal texts call them an array data type. typeof new Date() is "object"; typeof null is historically "object".

An unassigned declared variable or missing ordinary object property evaluates to undefined; directly reading an undeclared variable usually throws ReferenceError.

Strings use either quote style, with appropriate escaping. + can concatenate when a string is involved; arithmetic / converts ordinary numeric strings. "abc"/"def" produces NaN; nonzero Number divided by zero produces signed Infinity, while 0/0 produces NaN. parseInt("12.8",10) reads12, stopping at the dot; this is parsing, not rounding.

Global isNaN converts first; Number.isNaN does not.

Web Programming module

Mozilla MDN JavaScript data structures

WEB17 | Trace assignments, comparisons and postfix updates

Evaluate operators in their specified order and write down both value and type.

JavaScript has ==/!= with coercion and ===/!== without coercion; strict operators do not replace or remove the loose ones.

A condition ? x : y chooses one branch.

var a="20"; var b=a=30; assigns Number 30 to both, so a+b is 60, not 50. "1 plus 1 is "+1+1 concatenates left to right to "1 plus 1 is 11".

Postfix a++ yields the old value and then increments a. If a=10, a==a++ compares10 with10, then the body sees11.

With a=1, printing a-- then a prints10.

These JS evaluation rules are not portable to C/C++ expressions with unsequenced modifications.

Web Programming module

Mozilla MDN equality comparisons

Mozilla MDN increment operator

WEB18 | Math, string and array API microtraces

Math.abs(-9)=9, Math.floor(5.9)=5, and Math.sqrt(36)=6.

A JS string’s substring(start,end) includes start and excludes end: "exitexam".substring(2,6) is "itex".

Indexes start at zero.

Array push/pop add/remove at the end; unshift/shift act at the beginning. slice returns a selected shallow copy; splice(start,deleteCount,...items) mutates by removal/insertion. reverse reverses in place. reduce combines sequentially from left to right; reduceRight combines sequentially from right to left with an accumulator, not simultaneous parallel execution.

For [1,2,3].reduceRight((a,v)=>a-v,0), accumulator values are -3,-5,-6.

An absent initial value uses the last present item first; an empty array without one throws.

Method spelling and case matter.

Web Programming module

Mozilla MDN reduceRight

Mozilla MDN Math

Mozilla MDN substring

WEB19 | Control flow, functions and errors

for(initialization;test;update) initializes once, checks before each iteration, then updates after each body. while checks before its body; do...while executes at least once. break exits and continue advances to the next iteration.

A loop from n=1 while n<7 prints a message only at n=5 when testing n%5==0.

function f(x){return x*2;} has a parameter and returns a value; functions can be passed/stored like other values.

Arrow syntax is compact but differs in this/constructor behavior.

Runtime argument limits mean “no limit whatsoever” is too absolute. try/catch handles thrown errors during execution; throw signals one; finally runs cleanup.

Parse-time syntax errors and logical errors are different. bool ans=true is not JavaScript declaration syntax; use let ans=true.

A literal type of expression is not the typeof operator.

Web Programming module

WEB20 | DOM selection, creation and safe updates

The DOM is a hierarchy of nodes representing the document. document.getElementById("x") returns the matching element or null; querySelector(".item") returns the first match, and querySelectorAll returns a static NodeList.

Other plural lookup methods can return live collections.

Guard missing elements and run code after the relevant markup exists.

To add a div: const el=document.createElement("div"); el.textContent="Hello"; document.body.appendChild(el);.

Creation alone does not insert it. createTextNode makes text; appendChild attaches an existing node. innerHTML parses/replaces descendants, so += reparses/replaces the prior descendant content too and can destroy listeners/state.

Use textContent for untrusted plain text. element.style.backgroundColor sets a style property; classList.add changes a class. getElementByld with a lowercase l replacing I is a different, nonexistent method.

Web Programming module

WEB21 | Events and validated submission

An event handler runs when an event occurs: click, input, keydown, mouseover, submit and load are examples. addEventListener("click",handler) registers a function without calling it immediately.

Multiple listeners may coexist; once removes one after use, capture selects an earlier propagation phase and passive promises no default cancellation.

Event bubbling is distinct from the browser's default action.

For submission, form.addEventListener("submit",e=>{if(!valid()){e.preventDefault();}}) stops the default only on failure.

Returning false from an addEventListener callback does not cancel it; return false works in an appropriate inline/onsubmit property handler whose caller uses that return.

For cancelable keyboard defaults, call preventDefault rather than assuming a different key event itself disappears.

Browser checks improve feedback; repeat validation and authorization on the server.

Web Programming module

Mozilla MDN addEventListener

WEB22 | Window, dialogs and browser properties

In a normal browser page, window exposes the browsing context and document is its document property. alert displays information, confirm returns a Boolean, and prompt requests text or returns null on cancellation. console.log logs diagnostics. location.reload() reloads the current page. screen.availHeight, screen.availWidth and screen.colorDepth have exact camelCase spelling; ColorsDepth is not the property.

window.onload is a handler property for the load event, typically after dependent resources finish.

Global let/const are not window properties like classic top-level var bindings. document.write is a method/function property; writing after document load can replace the document. document.bgColor is a legacy property; Document.bgcolor has wrong object/case.

Use document.body.style.backgroundColor for current styling.

Browser workers support concurrent computation, but DOM access normally remains on the page's main thread.

Web Programming module

Mozilla MDN Screen

WEB23 | PHP syntax, names and types

PHP means PHP: Hypertext Preprocessor.

Use <?php ... ?> in a .php file; a closing tag can be omitted at file end.

It can emit ordinary text and HTML.

PHP8 no longer accepts historical <script language="PHP"> delimiters; short <? depends on configuration, so use the full form.

A stray >> is not a closing PHP tag.

Variables begin with $, then a letter/underscore and subsequent letters/digits/underscores; $AGE18 works but $18AGE does not.

Names are case-sensitive.

Dynamic typing permits values of different types without a preceding type declaration, though optional declarations can constrain them.

Scalars include int,float,string,bool; arrays/objects are structured values and null means no value. var_dump displays type and value.

Constants cannot normally be redefined during execution.

Web Programming module

WEB24 | PHP output, strings and comparisons

echo is a language construct with no return value and can emit comma-separated expressions. print takes one expression and returns1.

They are two basic output constructs, not the only output mechanisms; printf, print_r and others exist.

Neither automatically inserts spaces/newlines.

Two prints of5 emit55; "I"."am" emitsIam.

Dot concatenates; + is arithmetic.

Double-quoted strings interpolate variables and escapes such as \n; single quotes normally preserve dollar names literally.

To emit literal $lfc use echo '$lfc'; or escape the dollar in double quotes.

String offsets start at zero: "maroon"[2] is r when the surrounding PHP is valid. strlen counts bytes; strpos can return integer0 or false, requiring ===false for failure. strrev reverses bytes and str_replace replaces matching content.

PHP 8==8.0 is true, but 8===8.0 is false because types differ; echo false emits an empty string, echo true emits1.

PHP sends markup as response characters; a browser then interprets it. echo "A
B" emits a tag in raw output but normally displays two text lines.

The invalid </br> markup depends on HTML error recovery, so specify raw versus rendered output.

Web Programming module

PHP manual: strings

PHP manual: echo

WEB25 | PHP scope, parameters and persistent locals

A variable outside a function has global scope.

A function’s locals and by-value parameters are separate bindings; use global $x or $GLOBALS["x"] to refer to a global explicitly.

Superglobals are available across scopes.

A static local retains its value across calls during the execution lifecycle; static storage is not simply another visibility scope.

An uninitialized static local begins as null, so echo emits no characters.

If global $i=10 and sample($i) increments/prints its by-value parameter, the caller’s $i stays10.

Printing before, during and after gives10,11,10 when commas are written by the source.

A global statement outside the function does not turn its value parameter into a reference.

Common “three scopes” teaching counts local, global and static, but state the taxonomy.

A static counter initialized0 then incremented on each call emits1 then2.

Web Programming module

PHP manual: variable scope

WEB26 | PHP arrays, loops and callable behavior

PHP arrays map integer/string keys to values.

Numeric arrays start at index0 unless keys are supplied: [20,12,23,19,18] has19 at index3.

Associative syntax uses a contiguous => operator: ["top"=>"black"] stores black under key"top".

Splitting it as = > is a parse error, not valid array syntax.

foreach($items as $key=>$value) iterates entries; for and while use tests/updates much like the corresponding JS forms. $sub-=$sub is $sub=$sub-$sub, not decrement.

For $sum=0; foreach([2,4] as $v){$sum+=$v;} the result is6.

A function can receive arguments, return a value, or produce output; classes/objects/variables also support server processing, so no single one of these mechanisms alone defines “dynamic PHP.” Use braces to make nesting explicit.

Web Programming module

WEB27 | Request data, server validation and injection

$_GET contains query parameters; $_POST contains supported form POST data; $_FILES upload metadata; $_SERVER request/environment details; $_COOKIE cookie values; $_SESSION session data after startup. $GLOBALS is the actual global-symbol array; $_GLOBAL is not the built-in name. $_REQUEST may combine GET/POST/cookie data according to configuration; it is not a database interface and has ambiguous precedence, so prefer the intended source. isset tests existence and non-null, not truth or valid syntax.

Read input, check required fields/type/length/range, authenticate and authorize, then process.

Bind SQL parameters to keep values separate from commands.

Encode output for HTML/attribute/JS context; textContent handles plain browser text.

A security test can detect SQL injection or XSS but does not itself fix either.

Client validation, encryption and a database connection do not replace these controls.

Web Programming module

WEB28 | Cookies and browser storage

Cookies are browser-held name/value data automatically attached to matching requests subject to domain, path, expiry, Secure and SameSite rules.

HttpOnly restricts script access; Secure requires secure transport.

They are not attached to every site indiscriminately.

PHP setcookie sends a response header, so set it before output unless buffering correctly defers output; $_COOKIE reflects received cookies.

localStorage persists origin-specific string values across sessions until removed/cleared; sessionStorage is scoped to a page session/tab.

Neither is automatically transmitted like matching cookies, but JavaScript can send stored data explicitly.

IndexedDB supplies structured local storage.

Thus all four are client storage techniques, and localStorage can participate in token-based session designs, with script-access risks.

Storage size limits vary.

Being logged in after reopening may use a persistent session/remember token or browser restoration; storage alone does not prove server acceptance.

Web Programming module

Mozilla MDN localStorage

WEB29 | Server sessions and the authorization trust path

session_start() starts/resumes PHP session handling and makes $_SESSION available.

Normally the browser holds an opaque session identifier while data resides in server-controlled storage.

A counter initializes missing $_SESSION["views"] to1 and increments on later requests. unset removes one entry; destroying stored session data is distinct from clearing the browser cookie/current variables, so logout must handle the full lifecycle.

A client cookie logged_in=1 is attacker-controlled.

If the server trusts it alone, authorization can be bypassed.

If the server checks a valid session ID and its server-side user/role on every protected action, editing that unrelated flag does not grant rights.

HTTPS does not make client-generated claims trustworthy.

Determine which server-side check is performed before judging whether a session-handling design is secure or vulnerable.

Protect session IDs, regenerate after login, expire appropriately and avoid putting sensitive tokens in URLs.

Web Programming module

WEB30 | Uploads, files and storage safety

Uploads ordinarily use method="post", enctype="multipart/form-data" and a named file input. accept hints to the browser which file types to offer; it does not enforce the server's policy.

Validate server-side size/content, inspect upload errors, use the temporary upload path, choose a safe generated destination and use move_uploaded_file.

Avoid executable storage and trusting a client filename/extension.

file_exists tests files or directories; is_file narrows to regular files. fopen returns a stream or false: r reads an existing file, r+ reads/writes without truncation, w truncates/creates for writing, w+ adds reading, a appends/creates, and x exclusively creates. fclose releases it. readfile emits content and returns a byte count, so echo readfile(...) also prints that count.

Check failures before reading/writing and constrain paths to permitted locations.

Web Programming module

WHATWG HTML input standard

WEB31 | Database connections, TLS and actual APIs

MySQL is a relational DBMS; a database persists structured application data and supports concurrent clients.

PHP8 uses MySQLi or PDO; the old mysql_* extension was removed, so migrate older snippets to a supported API. mysqli_connect(host,user,password,database,port,socket) opens a connection; selecting a database can be deferred, and defaults make several arguments technically optional.

Do not reduce the full signature to exactly one optional field.

Configure MySQLi TLS options through mysqli_init(), mysqli_ssl_set(...) and mysqli_real_connect(...) with the required verification configuration.

mysqli_ssl_connect(), pdo_connect_ssl(), mysql_ssl_connect() and secure_connect() are not substitutes for the MySQLi TLS configuration sequence above.

PDO uses an appropriate driver DSN/options instead.

MySQLi procedural functions, object methods and PDO methods have distinct signatures; affected_rows() alone is not a generic PHP function.

Web Programming module

PHP manual: mysqli::ssl_set

PHP manual: affected_rows

WEB32 | SQL operations, result rows and safe rendering

SELECT retrieves rows; INSERT adds; UPDATE ... SET changes values; DELETE removes rows.

WHERE limits affected/matching records; omitting it from DELETE ordinarily targets every row, subject to constraints, permissions and errors.

CREATE DATABASE/TABLE defines storage; a primary key identifies rows.

AUTO_INCREMENT generates numeric values but gaps and explicit values are possible; do not promise gapless consecutive business numbering.

mysqli_query returns a result object for successful result-producing queries, true for other successful statements, or failure/exception depending on settings.

Fetch associative rows with mysqli_fetch_assoc, positional rows with mysqli_fetch_row, or objects with mysqli_fetch_object. mysqli_num_rows counts a result's rows; mysqli_affected_rows($conn) counts changed/inserted/deleted rows under its options; mysqli_insert_id($conn) reports a generated AUTO_INCREMENT value from the connection.

Escape rendered cell text; use prepared statements for submitted values; mysqli_close closes the connection.

Web Programming module

PHP manual: affected_rows

WEB33 | PHP includes, objects and dates

include evaluates another file in the current scope and warns on failure; require fails critically.

In PHP8 a failed require raises Error, which can be caught; uncaught it stops normal execution.

The source's E_COMPILE_ERROR description belongs to older PHP. include_once/require_once avoid repeated inclusion.

Do not include a user-selected arbitrary path.

A class defines properties/methods; new creates an object, $obj->field accesses a property, and $obj->method() calls a method. public is externally accessible, private is class-restricted, and a constructor initializes an instance.

Use $this inside instance methods.

DateTime/DateTimeZone represent time and zone; diff returns a DateInterval and DatePeriod iterates intervals.

Time calculations require explicit zones and calendar assumptions, not an assumed server-local timezone.

Web Programming module

PHP manual: require

WEB34 | AJAX and response handling

AJAX means Asynchronous JavaScript and XML, but responses can be JSON, text, HTML or XML.

It is a technique, not a language, standalone framework or mandatory XML protocol.

Configure an XHR with open("GET",url,true), register response/error handlers and call send(). onreadystatechange may run several times; readyState4 means completion, after which status and data must be checked.

A callback is a function invoked later by the runtime/API.

The browser invokes its response handler when the response is available; the remote server does not directly execute browser code.

With fetch, await fetch(url) obtains a response, then check response.ok and parse await response.json().

Handle network/parse errors.

Starting an asynchronous operation does not make its result synchronously available to the next statement.

Avoid overwriting a newer search result with an older response that arrives late.

Web Programming module

WEB35 | Polling, WebSockets and data formats

Short polling repeatedly requests updates.

Long polling holds one client request until an event or timeout, returns a response, then the client reconnects; it is not continuous unsolicited streaming on that same finished response.

WebSocket establishes a persistent bidirectional message channel after its handshake.

Either HTTP-based AJAX or WebSockets can support nonblocking user interaction and carry JSON, so generic questions may be nonexclusive.

JSON represents objects, arrays, strings, numbers, Booleans and null; keys/strings use double quotes, and functions/undefined are not JSON values.

XML is another valid interchange format.

A REST API can return either.

Select transport by latency, direction, connection cost and recovery requirements, not merely the occurrence of “asynchronous.” Validate incoming messages and enforce authorization per operation.

Web Programming module

WEB36 | Node server, event loop and scaling

A minimal CommonJS Node server can use const http=require("node:http"); http.createServer((req,res)=>{res.writeHead(200,{"Content-Type":"text/plain"});res.end("Hello");}).listen(8000);.

The request listener runs for incoming requests; end completes the response.

Define every variable used in startup logging, including hostname, before using it.

Node schedules JavaScript callbacks through an event loop and uses OS/worker facilities for asynchronous work.

Concurrency means overlapping progress; parallelism means execution at the same instant.

One event-loop thread can handle many waiting requests, but long CPU loops or synchronous I/O block its callbacks.

Workers/processes and a load balancer distribute work across server instances; they do not automatically coordinate shared data or session state.

Multiple clients, or one client with overlapping requests, can both create concurrency.

Web Programming module

Node.js: do not block the event loop

WEB37 | Races, locks and transaction invariants

Suppose two requests read a stock count1, both decide it is available and both write0: two sales consumed one item.

Correctness depends on interleaving, even if each callback has only a short synchronous body.

A mutex protects one critical section; semaphores can coordinate bounded access; spinlocks busy-wait and are unsuitable for blocking a browser/event-loop thread.

Several listed mechanisms can synchronize, so mutex is not the only possible answer.

In a relational database use an atomic conditional UPDATE stock SET qty=qty-1 WHERE id=?

AND qty>0, check affected rows, and combine related writes in an appropriate transaction.

A row lock or serializable transaction can protect multi-step logic; handle rollback/retry on conflicts.

A process-local lock does not cover all load-balanced servers.

Idempotency tokens with unique storage constraints prevent a retried payment from becoming a duplicate transaction.

PostgreSQL transaction isolation

WEB38 | Frameworks and version-aware setup

A framework supplies conventions and reusable structure; a library supplies callable capabilities. jQuery simplifies DOM operations; React builds component interfaces; Angular is a component platform, distinct from older AngularJS; Vue also builds interactive interfaces.

Each can contribute to a responsive interface, so those generic descriptions do not identify one unique framework.

React props pass inputs; state records changing component data; JSX needs transformation rather than being ordinary browser JavaScript syntax.

Bootstrap supplies layout/components.

The jQuery→Popper→Bootstrap dependency order is version4-era guidance; Bootstrap5 drops required jQuery.

Angular CLI commands include ng new, ng serve, ng build, ng generate and ng test, with e2e support depending on configuration. Use ng serve to start the development server.

Create React App is deprecated; recognize it as historical tooling rather than a universal current recommendation.

Web Programming module

Bootstrap5 migration

React installation

React: passing props

WEB39 | Manage content, performance and revisions

A CMS gives editors content creation, publishing, roles and media management without editing every source file.

A database stores persistent facts; it is not itself the whole server application.

A VCS records source history, branches and merges to support review/collaboration; deployment publishes a chosen build and is a separate operation.

SEO improves discoverability through meaningful content, headings, metadata, accessible links and crawlable structure; keyword metadata alone does not guarantee ranking.

Caching reuses a representation/result and requires validity/invalidation rules.

Server caching can hold rendered pages or expensive query results; never share private user content under an unsafe common key.

Minification removes unnecessary source characters; bundling combines files; compression changes transfer encoding; preloading fetches anticipated resources early.

Measure performance because excess preloading/bundling may compete with critical work.

Responsive design and cross-browser compatibility need separate testing.

Web Programming module

WordPress features

Google Search Central SEO Starter Guide

Pro Git: About Version Control

WEB40 | Assemble and evaluate an end-to-end feature

For a course-search page, write semantic headings/navigation and a labeled search form.

Use a responsive grid with a single-column fallback, then register a submit handler after parsing.

Prevent default only when implementing asynchronous submission; encode search values in the query and send GET /courses?q=.... Show loading and useful error feedback without inserting untrusted HTML.

The server validates query length/type, authenticates where required, runs a parameterized SELECT with a bounded result size, and returns JSON with an appropriate status/content type.

The client checks status, parses data and builds nodes with textContent.

For an enrollment action use POST with CSRF controls, server authorization, capacity checks and a transaction; show success only after commit.

Evaluate empty/bad input, access denied, network failure, delayed responses, repeated clicks, simultaneous enrollments, narrow screens and keyboard use.

This integrates all four blueprint outcomes beyond tag recall.

Web Programming module

WEB41 | Validate syntax, context and output

A trace requires syntactically valid source, known initial values, execution context and output representation. x=4+"4"5 is a syntax error because an operator is missing; bool ans=true and type of new Date() are syntax errors too.

PHP with >> instead of ?> may fail parsing; analyze output only after checking the closing syntax.

The PHP array operator is =>; splitting it as = > is invalid syntax.

Distinguish response text from browser rendering: PHP may emit markup that the browser interprets, and visual line breaks need not appear in a plain-text choice.

For the assignment example above, the correct output is 60 even if it is absent from the choices.

Multiple storage techniques, interface libraries, script-capable languages and cryptographic channels often satisfy broad descriptions.

State the missing distinction before selecting an answer when the question admits several interpretations.

With x=4+"4";, JavaScript produces 44; x=4+"4"5 is invalid because an operator is missing.

The PHP string-offset example closes with ?> and prints r; using >> instead introduces a syntax error.

Check the complete code and closing delimiters before tracing output.

Apply each output prediction to the exact program being analyzed.

Web Programming module

WEB-S01 | Java and JavaScript object-oriented models

Java organizes object-oriented programs using classes, objects, inheritance and interfaces. A class describes the state and behavior of its instances; inheritance relates a subclass to a superclass, and an interface specifies operations a type agrees to provide.

Oracle Java Tutorials: Object-Oriented Programming Concepts

WEB-S02 | Form values and result destinations

For a text input, the value attribute supplies its initial text value; it does not specify the control's width. The type attribute selects the kind of input control.

A form's target identifies the window or frame in which the submitted result is displayed. The action identifies the processing URL, so changing the destination context is different from changing the handler.

In an input element, type="text" creates text entry, type="checkbox" an independently selectable checkbox, type="radio" a radio choice, and type="submit" a submit button. A hidden input carries a value without displaying an editable text field; it is not a password-masking control.

HTML Standard: The input element

Web Programming module · page 21

WEB-S03 | Message display versus response collection

alert(message), confirm(message) and prompt(message) all present a supplied message. Alert is a notification; confirm additionally asks for an OK/Cancel decision, while prompt additionally offers a text field. A general message-display description therefore differs from an information-only notification.

Web Programming module · page 46

Web Programming module · page 47

WEB-S04 | Database API namespaces and affected rows

PDOStatement::rowCount() is a method on a PDO statement object that reports affected rows for its DELETE, INSERT or UPDATE statement. It is not a procedural MySQLi function, and its behavior for SELECT result counts is driver-dependent rather than a portable row-count guarantee.

With an initialized PDO connection, $pdo->query($sql) executes an SQL statement without placeholders and returns a PDOStatement for processing its results. The connection's configured driver determines the database; the method name alone does not mean MySQL.

MySQL is an open-source SQL database management system. Its relational data-management role is distinct from the PHP API used to connect to it.

The procedural MySQLi query form is mysqli_query($conn, $sql), using a MySQLi connection and the correctly spelled function name. The pg_query function belongs to PHP's PostgreSQL extension; it is not a MySQL connection API.

PHP Manual: PDOStatement::rowCount

PHP Manual: PDO::query

Web Programming module · page 78

PHP Manual · mysqli_query

PHP Manual · pg_query

WEB-S05 | JavaScript execution, statement forms and error categories

JavaScript implementations may interpret or compile source; interpretation and compilation are execution strategies rather than competing language identities.

An if statement conditionally executes code. The legacy with statement temporarily extends name lookup and is forbidden in strict mode. A debugger statement may invoke an active debugging facility. These are keyword-introduced statement forms, unlike the quoted use-strict directive.

A syntax error prevents source from being parsed successfully. A runtime error arises while executing a program. A logical error is a defect in the algorithm or expression that can produce an incorrect result without throwing an exception.

Web Programming module · page 44

ECMAScript: Statements and declarations

Mozilla MDN: Troubleshooting JavaScript

WEB-S06 | PHP statement boundaries and string escapes

A complete ordinary PHP assignment statement ends with a semicolon. A string's closing delimiter must match its opening delimiter; a single quote can appear inside a double-quoted string without ending it.

In double-quoted PHP strings, backslash-n represents a line feed and backslash-r represents a carriage return. The written sequences are \n and \r; forward-slash sequences such as /n are ordinary text. A backslash before a dollar sign preserves that dollar sign rather than starting variable interpolation.

The basic print and echo constructs can be written with or without parentheses when their syntax permits it. Parentheses do not add spaces or change the value of the emitted argument.

PHP Manual: Strings

Web Programming module · page 62

Web Programming module · page 63

WEB-S07 | Default text rendering and legacy frame fallbacks

In the browser's default presentation, an i element typically displays its text in italics. This is a rendering convention: its text semantics are distinct from the stress emphasis represented by em, and CSS may override appearance.

A legacy frameset document could include noframes content for user agents that did not support frames or were configured not to display them. Frame support was therefore not universal even in that historical model.

Web Programming module · page 15

Web Programming module · page 20

WEB-S08 | Numeric functions and NaN checks

For finite numeric arguments, Math.abs(x) gives the nonnegative magnitude, Math.floor(x) gives the greatest integer no larger than x, and Math.sqrt(x) gives the nonnegative square root for nonnegative x. These operations differ from ordinary nearest-integer rounding; floor moves toward negative infinity.

Global isNaN(value) tests whether conversion to Number produces NaN. Number.isNaN(value) tests specifically for the Number value NaN without coercion. NaN is itself a special numeric value, not a callable checking function.

ECMAScript: Math object functions

MDN · isNaN

WEB-S09 | Destroying server-side session data

session_destroy() removes the data associated with the current PHP session. It does not itself unset every current script variable or delete the browser's session cookie; destroying stored data, clearing $_SESSION and expiring the identifier cookie are separate parts of a complete logout.

Web Programming module · page 77

PHP Manual: session_destroy

WEB-S10 | HTTP response status classes

The first digit of an HTTP status code identifies its class: 2xx means successful handling, 3xx means redirection or further action, 4xx indicates a client-error condition, and 5xx indicates a server-error condition. Classifying a code is different from identifying the precise condition represented by its last two digits.

RFC 9110: Status codes

WEB-S11 | Authentication versus authorization

Authentication verifies a claimed identity. Authorization checks whether a particular entity is permitted to perform a requested action. Establishing identity does not grant every permission, and neither check alone repairs unsafe SQL construction or unsafe output encoding.

OWASP Authentication Cheat Sheet

OWASP Authorization Cheat Sheet

WEB-S12 | Metadata purposes

The keywords metadata name supplies terms relevant to the page, while description supplies a page summary. Robots metadata supplies crawler indexing or search-presentation instructions. These roles differ from mobile viewport control.

HTML Standard: Standard metadata names

Google Search Central: Robots meta tag

WEB-S13 | Remainder and logical conjunction

For nonnegative integers n and positive d, n % d gives the remainder after division; zero tests divisibility. With Boolean operands, && is true only when both operands are true; it is a logical operator, not a comment delimiter.

Mozilla MDN: Remainder operator

Mozilla MDN: Logical AND

WEB-S14 | Application protocols and underlying communication

HTTP defines web application requests, response status codes and representations. An HTTP exchange can use underlying TCP connections and IP addresses; naming that network stack does not identify the application-level method or status semantics.

FTP is a file-transfer application protocol. SMTP transports electronic mail. These service roles differ from the HTTP resource request/response semantics used by a web browser. HTTPS adds TLS protection to HTTP rather than replacing its resource semantics.

RFC 9110 · HTTP Semantics, §§3 and 4

RFC 959 · File Transfer Protocol

RFC 5321 · Simple Mail Transfer Protocol

Operating System

Complete Study Notes

OS01 | Objectives, resource management and evolution

An OS allocates CPU time, memory, storage, devices and communication resources, and provides services to programs.

Convenience means making tasks easy through consistent interfaces, files and device abstractions.

Efficiency means using resources effectively; it is not synonymous with the fastest response for every process.

Application development tools use OS services but application development itself is not a core resource-management function.

Ability to evolve means adding hardware support, features and repairs through maintainable interfaces/modules.

Backward compatibility preserves supported older application behavior; it is not a promise that every old binary works.

Scalability means handling greater workload/resources; portability means adapting to other platforms.

Fault tolerance continues correct service despite specified failures; recovery restores service afterward.

Updates and monitoring help administration, but automatic updates are not a defining property of every OS.

Operating System module

Oracle Solaris 11

OS02 | Batch, multiprogramming, time-sharing and real time

Batch jobs run with little interaction after submission.

Multiprogramming keeps several jobs available so another can use the CPU while one waits; it does not require fixed time slices.

Time-sharing uses rapid scheduling/preemption to support interactive users.

A single CPU interleaves execution; multiple CPUs can execute work truly in parallel.

Shared-memory multiprocessors do not require each CPU to have private main memory.

Single-user and multiuser describe concurrent user access, not the number of applications.

Network systems expose remote services; distributed systems coordinate work across machines, sometimes hiding location.

Real-time correctness includes timing: hard deadlines must be met, while soft deadline misses degrade service. “Real time” does not simply mean fast.

Spooling and multitasking are mechanisms, not proof of a particular OS generation.

Operating System module

OS03 | Kernel, shell, privilege and boot

The kernel implements core resource management, protection and system-call handling.

A shell reads/interprets user commands; a GUI is another interface.

Browsers, office tools and editors are applications.

Device drivers are commonly kernel components, although user-space drivers also exist.

Kernel mode has privileged operations; supervisor mode often means the same privileged mode, so offering both as exclusive answers is ambiguous.

Firmware/bootloader stages execute before transferring control to a kernel loaded into RAM.

The kernel is the first core OS component, not the first program ever executed at power-on.

Loadable modules can add drivers to a running kernel; “kernel modules cannot be loaded while running” is false in modular systems.

A resident kernel remains active through the session.

Utilities such as antivirus and disk tools can combine user processes with privileged components.

Operating System module

OS04 | APIs, system calls and virtualization

A system call requests a protected kernel service, such as creating a process, reading a file or opening a socket.

A user-facing API/library may wrap that request; not every library function enters the kernel.

Distinguish the application API from the direct user/kernel boundary when describing access to OS services.

Process control, file/device management, information retrieval and communication are service categories.

Video editing and a database-specific write_database() are application operations, not standard generic syscall categories/names.

Virtualization presents a virtual machine environment, often allowing guest operating systems on one host through a hypervisor.

It differs from virtual memory, which translates a process’s addresses.

Kernel, shell, API and VM are not interchangeable labels.

Operating System module

OS05 | Programs, processes and PCB contents

A program is stored instructions; a process is an executing instance with address space and resources.

A process does not universally “manage applications” or “provide a user interface.” The PCB records identity, state, PC/registers, scheduling information, memory/accounting details and I/O/open-resource state.

A saved PC identifies where execution resumes.

A context switch saves one execution context and restores another; useful application work is interrupted by this overhead.

An interrupt or system call can switch privilege without switching process.

Task is OS-specific terminology: it may mean process, thread or schedulable entity.

No universal process-versus-task distinction follows from the word alone.

Operating System module

OS06 | Process states, queues and schedulers

New processes are created/admitted; ready means eligible for CPU but not currently running.

Running uses a CPU.

Waiting/blocked means unable to proceed until an event; terminated means execution has ended.

Running→ready follows time-slice expiry or another preemption.

Running→waiting follows blocking I/O. I/O completion normally gives waiting→ready, not a guaranteed immediate running state.

The ready queue contains eligible work; device queues hold blocked requests.

The broad textbook job queue contains processes in the system, while an admission queue more narrowly holds jobs awaiting memory/admission.

Long-term scheduling admits jobs; medium-term scheduling suspends/swaps them; short-term scheduling selects ready work.

The dispatcher performs the handoff/context restore after selection.

A bare “priority” causes running→ready only if a preemptive policy chooses another process.

Operating System module

UIC CPU Scheduling

OS07 | Creation, fork/exec and termination

Processes arise during initialization, user requests, service requests or a running process's creation call.

In POSIX-style systems fork() creates a child: zero is returned in the child and the child PID in the parent on success.

The two have distinct address spaces, although resources may be shared through explicit mechanisms. exec replaces the current process image; it does not itself create another process.

Normal completion and a program-detected error exit are voluntary; an unrecoverable fault or being killed is involuntary. exit() terminates normally through the runtime; abort() requests abnormal termination.

A reference to nonexistent/protected memory is a fatal fault, unlike an application deciding to exit after bad input. "Terminate exit" is not a precise standard category.

Parent termination does not universally kill all descendants.

Linux man-pages fork(2)

Linux man-pages execve(2)

OS08 | Threads and mapping models

A thread has its own PC, registers and stack.

Threads in a process share code, address space and usually open-file resources.

A process can contain many threads; a thread belongs to one process at a time.

Threads are not a group of processes and need not execute sequentially.

Independent processes can exist; cooperating ones communicate/share data.

User-level threads are managed by a library; kernel threads are known to the OS.

In many-to-one mapping, a blocking kernel call can block all user threads and one kernel execution context cannot use multiple CPUs simultaneously.

One-to-one permits another kernel thread to run when one blocks.

Many-to-many multiplexes user threads over several kernel threads.

Speed and reliability depend on implementation; the management layer alone guarantees neither.

Application names such as browser or Excel do not prove a particular version's threading model.

Operating System module

OS09 | IPC and blocking semantics

Shared memory lets processes access a common region and requires synchronization for conflicting updates.

Message passing exchanges data through send/receive operations.

Pipes provide streams; message queues preserve messages; sockets connect endpoints; RPC requests a remote operation.

HTTP is an application protocol over a communication mechanism and can carry interprocess communication, so it is not universally “not IPC.”

Blocking send waits for the required handoff/buffering/receipt condition defined by its API.

Blocking receive waits for a message.

Nonblocking operations return promptly with data, progress or a would-block result. “Sender blocked until received” identifies blocking send under rendezvous semantics.

File locks coordinate access rather than transfer arbitrary payloads, though locking is synchronization within IPC.

Do not equate asynchronous messaging with no synchronization ever.

Operating System module

OS10 | Races, critical sections and correctness

A race occurs when results depend on uncoordinated interleaving of conflicting accesses.

A critical section is code accessing shared state that needs protection, not the resource itself.

A correct exclusion solution requires mutual exclusion, progress and bounded waiting under stated assumptions.

Mutual exclusion is a desired safety property; races, starvation and deadlock are possible failures.

Keep critical regions small and ensure every conflicting access uses the same protection.

A barrier waits until a defined set of participants reaches a point; a monitor encapsulates protected operations with condition synchronization.

Locking can itself introduce deadlock if acquisition order is inconsistent.

Example

Worked microexample

- Two threads read counter=0, each computes 1 and each stores 1. Two increments produce 1 instead of 2. Protecting the entire read-modify-write with one lock prevents that lost update.

Operating System module

OS11 | Semaphores and resource-count traces

A semaphore provides atomic wait/P/down and signal/V/up operations.

Under the nonnegative-token model, wait consumes one available token or blocks when none is available; signal supplies a token and may wake a waiter.

Implementations using negative internal counts encode waiters differently, so state the model.

A binary semaphore represents one permit; a counting semaphore tracks several.

A mutex normally has ownership rules that a general semaphore lacks.

Example

Worked microexample

- Initialize a mutex semaphore to 1. Each thread does wait(m); critical work; signal(m). For a buffer of N slots use empty=N, full=0 and mutex=1: producer waits empty then mutex, inserts, unlocks then signals full. Waiting for empty while already holding mutex can prevent the consumer from freeing a slot.

Arpaci-Dusseau and Arpaci-Dusseau, OSTEP Semaphores

OS12 | Turn variables and atomic primitives

Strict alternation uses turn=0/1: process i waits while turn≠i, then enters and sets turn to the other process on leaving.

It enforces turns but can violate progress if the other process stays outside its critical region.

A naive lock-variable test followed by a separate set races because both processes can observe free before either sets it.

Test-and-set and compare-and-swap combine checking/updating atomically; ordinary check-then-write does not.

Peterson’s classic algorithm uses flags plus turn for two processes under its memory-model assumptions; a semaphore can coordinate more than two.

Disabling local interrupts does not exclude other CPUs and is not a general user-space locking solution.

Interrupts can notify/synchronize events, so they cannot be universally excluded from every synchronization-related classification.

UIC Process Synchronization

OS13 | Scheduling metrics and calculation method

For arrival A, first start S and completion C, response=S−A and turnaround=C−A. For a single CPU burst B with no I/O, waiting=C−A−B. With I/O, subtract all CPU and blocked time to isolate ready-queue waiting.

Throughput=completed jobs/elapsed time; CPU utilization=busy time/elapsed time.

Minimize response/turnaround/waiting under the selected goal and maximize useful throughput/utilization.

First construct a time line with arrivals, idle gaps, preemptions and overhead, then calculate per-process values before averages.

Do not confuse first response with final completion.

Example

Worked microexample

- A job arrives at 2, starts at 5, uses 4 CPU units without interruption and finishes at 9: response=3, turnaround=7, waiting=3. Six completions in two hours means throughput=3 jobs/hour.

Operating System module

OS14 | Preemptive versus non-preemptive decisions

Non-preemptive scheduling keeps a running CPU burst until completion, blocking or voluntary yielding.

Preemptive scheduling can return running work to ready, usually after a timer quantum or a more urgent arrival.

FCFS and non-preemptive SJF are non-preemptive; RR and SRTF are preemptive.

Priority can be either, so its name alone cannot decide.

Cooperative describes relying on voluntary release.

CPU preemption saves a context and reallocates processing time; taking an occupied printer or lock may be unsafe.

This is why time-slicing does not automatically remove the no-resource-preemption condition for deadlock.

Frequent switches have cost; non-preemptive policies can delay interactive work behind long bursts.

Operating System module

UIC CPU Scheduling

OS15 | FCFS and the convoy effect

FCFS runs ready bursts in arrival order and ordinarily does not preempt.

If the CPU is idle, jump to the next arrival.

Long CPU-bound work can make many short jobs wait: the convoy effect.

A finite burst with finitely many jobs ahead eventually starts, but waiting can still be large.

FCFS does not universally have the worst turnaround for every possible workload.

Example

Worked microexample

- All ready at time 0, tie order P1=24,P2=3,P3=3: schedule 0-24 P1,24-27 P2,27-30 P3. Waiting times 0,24,27 average to 17. Turnarounds 24,27,30 average to 27. If arrivals are not simultaneous, recompute using their actual times.

Operating System module

OS16 | SJF/SJN and shortest remaining time

Non-preemptive SJF/SJN chooses the shortest next burst among currently ready jobs.

SRTF reconsiders on arrivals and selects the shortest remaining work; a shorter newcomer can preempt.

Future burst lengths normally need estimates.

SJF minimizes average waiting for the usual fixed simultaneous ready set with known bursts and no preemption/switch cost; that theorem is not a workload-independent comparison against every preemptive policy.

Repeated short arrivals can starve a long job.

The shortest total original burst is not necessarily the shortest remaining burst after execution has begun.

Example

Worked microexample

- P1 arrives 0 with burst 8; P2 arrives 1 with burst 2. SRTF runs P1 0-1, P2 1-3, P1 3-10. P1 waiting=2 and P2 waiting=0. Non-preemptive SJF cannot interrupt P1 once it has started.

Operating System module

OS17 | Priority, starvation and aging

Priority scheduling chooses the highest-priority ready job.

Check whether a smaller or larger numeric value means higher priority.

State whether a high-priority arrival preempts.

Fixed-priority scheduling can starve low-priority jobs under continued high-priority load.

SJF can also starve long jobs; starvation is not exclusive to fixed-priority scheduling.

Aging increases waiting jobs' effective priority according to a rule, reducing indefinite postponement when the priority scheme and workload permit.

It is not a deadlock cure.

Priority inversion occurs when urgent work waits on a resource held by lower-priority work; priority inheritance can temporarily raise the holder's priority.

Operating System module

OS18 | Round Robin, quantum and bounded turns

RR serves a ready queue cyclically for at most quantum q per turn.

Unfinished jobs return to the tail; completed/blocked jobs leave.

Specify how arrivals at a quantum boundary enter the queue.

A large quantum approaches FCFS; tiny quanta increase switch overhead.

RR gives recurring opportunities, not identical total CPU consumption for jobs of different lengths.

With n continuously ready jobs, no priority bypass and zero overhead, a just-requeued job waits at most (n−1)q for its next turn.

This bounds one interval, not total lifetime waiting when arrivals or blocking are unrestricted.

Include context-switch costs if given.

Example

Worked microexample

- P1=5,P2=3, both arrive 0, q=2: P1 0-2,P2 2-4,P1 4-6,P2 6-7,P1 7-8. Completions 8,7; waiting=8−5=3 and 7−3=4; average 3.5.

Operating System module

OS19 | Multilevel queues and feedback

A multilevel queue separates classes, such as foreground interactive and background batch, and chooses both between queues and within them.

Fixed interqueue priority may starve lower queues; allocating time shares between classes can avoid this.

A multilevel feedback queue allows movement according to observed behavior, commonly demoting CPU-intensive jobs and periodically boosting priority.

Do not infer movement from a simple multilevel-queue label.

State initial class, quantum per level, promotion/demotion rules and interqueue policy before tracing.

A binary search tree is a data structure that might implement a queue; it is not itself a CPU selection policy. "Infinite" is not a standard scheduling-policy category.

Operating System module

OS20 | Fair-share, proportional shares and guarantees

Fair-share scheduling allocates according to stated weights, potentially per user/group rather than per process.

Equal per-process RR can let a user obtain more aggregate CPU by launching more processes.

Proportional shares address that policy choice; fair need not mean equal.

Lottery scheduling assigns tickets and selects a winning ticket each turn.

With weights 3:1, expected shares are 75% and 25%; short-run results vary and this is not a deterministic no-starvation deadline.

Stride-style scheduling tracks progress toward shares deterministically.

Any “guaranteed scheduling” answer needs the actual guarantee and assumptions; finite FCFS and RR can also avoid indefinite postponement.

Example

Worked microexample

- If user A runs four ready tasks and B runs one, plain RR gives A about 80% overall. A policy granting each user half the CPU instead gives A’s tasks about 12.5% each and B’s task 50%.

Arpaci-Dusseau and Arpaci-Dusseau, OSTEP Proportional Share

Operating System module

OS21 | How to evaluate scheduling claims

There is no universally best policy for response, waiting, fairness and throughput simultaneously.

State the workload and objective, apply the algorithm and compare measured results.

Maximum waiting at submission is not determined from burst times alone if future arrivals, priorities or overhead are unconstrained.

Known finite workloads are different from an open stream with potential starvation.

An item asking for one non-preemptive algorithm while offering FCFS and SJN has multiple answers.

Equal share needs a process/user/group definition. “Highest average turnaround” requires a concrete workload; do not memorize FCFS as an absolute.

RR is a conventional interactive choice because turns improve responsiveness, subject to quantum and overhead.

Operating System module

OS22 | Deadlock and its four necessary conditions

A set is deadlocked when each member waits for an event/resource that only another blocked member can provide.

Mere waiting for another holder is normal blocking; indefinite unfair delay can be starvation without a cycle of dependencies.

Mutual exclusion allows at most one holder of a nonshareable resource.

Hold-and-wait means retaining a resource while requesting another.

No resource preemption means release must be voluntary.

Circular wait forms a closed chain of such waits.

All four are necessary in the standard reusable-resource model.

Their availability as general system policies does not prove a deadlock exists at this instant; inspect actual allocations/requests.

Preventing any one condition prevents that class of deadlock.

A preemptive CPU scheduler can coexist with deadlocked locks.

Example

Worked microexample

- P1 holds lock A and requests B; P2 holds B and requests A. Neither releases before acquiring the other: nonshareable locks, retained holdings, no forced release and a cycle all occur.

Operating System module

OS23 | Resource graphs and missing-diagram limits

In a resource-allocation graph, P→R is a request and an instance R→P is an assignment.

Follow these directions to form waits.

No cycle implies no deadlock; with one instance of every resource type, a cycle implies deadlock.

With multiple instances a cycle alone is insufficient: another instance may permit completion.

A wait-for graph replaces single-instance resource nodes with P→Q when P waits for Q.

For a complete single-instance example, suppose P0 holds R0 and requests R1, while P1 holds R1 and requests R0.

The wait-for edges P0→P1 and P1→P0 form a cycle, so neither can proceed.

With multiple resource instances, inspect all assignments and available instances before concluding that a cycle is a deadlock.

Operating System module

OS24 | Prevention: identify the condition removed

Prevention constrains how requests are made.

Share a resource where safe to remove mutual exclusion; request all needed resources before holding any to remove hold-and-wait; permit safe resource withdrawal/restart to attack no-preemption; impose a global acquisition order to remove circular wait.

Not every physical resource is shareable or safely preemptable.

“If waiting, release all held resources and try again” removes simultaneous holding while waiting.

Distinguish voluntary resource release from forced preemption when describing resource reclamation.

Prevention can waste utilization or cause starvation; avoiding deadlock is not the same as treating every process fairly.

Operating System module

OS25 | Avoidance and the Banker safety test

Avoidance allows requests only if the resulting state remains safe.

Banker requires maximum claims.

Compute Need=Max−Allocation.

Set Work=Available and repeatedly find an unfinished process with Need≤Work componentwise; simulate completion by adding its Allocation to Work.

Finishing all gives a safe sequence.

Tentatively grant a request only if it is within Need and Available and the safety test succeeds.

Unsafe does not mean already deadlocked.

Banker is avoidance, not CPU scheduling, detection or prevention by breaking a condition.

Example

Worked microexample

- One resource type: Available=1; P1 holds 1 and needs 1 more; P2 holds 2 and needs 2 more. P1 can finish, returning its allocation so Work becomes 2; P2 can then finish. Safe order P1,P2. A maximum claim must be known before testing.

University of Illinois Chicago, Deadlocks

OS26 | Detection, recovery and intentional ignoring

Detection asks whether the current state is deadlocked.

Recovery can terminate selected processes or reclaim safely preemptable resources and roll affected work back to a checkpoint.

Termination and rollback are both possible recovery tools; choose according to the resource and process conditions.

Repeatedly sacrificing the same victim can cause starvation.

Checking on every request versus periodically trades overhead against detection delay; no universal best interval exists.

Ignoring a sufficiently rare problem is sometimes called the ostrich policy, but real systems still prevent or detect selected deadlocks. “Deadlock acceptance” can mean intentional ignoring, not necessarily an invalid approach.

A detection graph, avoidance safety graph and ordinary scheduling queue solve different problems.

University of Illinois Chicago, Deadlocks

OS27 | Address binding, dynamic loading and linking

Compile-time binding uses absolute addresses if placement is known.

Load-time binding relocates when placed into memory.

Execution-time binding translates while running and permits movement with hardware support.

Logical/virtual addresses are generated for a process; physical addresses select RAM.

An MMU translates logical addresses to physical addresses, not the reverse.

Dynamic loading brings a routine in when needed.

Dynamic linking resolves references to external libraries at load/runtime, allowing suitable library code pages to be shared; static linking copies selected code into the executable.

Swapping traditionally moves a whole process to backing store; paging moves individual pages.

These differ from a context switch, which primarily changes execution state.

Operating System module

OS28 | Contiguous allocation and fit policies

A contiguous allocation gives a process one continuous region.

In the standard base/limit model, base is that process’s physical starting address and limit is its permitted length.

Check $0 \le \text{logical_offset} < \text{limit}$, then $\text{physical} = \text{base} + \text{offset}$.

The relocation register does not normally hold the OS base.

If limit instead means last valid offset, the inequality changes; specify the convention.

First-fit selects the first adequate hole; next-fit continues from the previous search position; best-fit selects the smallest adequate hole; worst-fit selects the largest.

No policy universally optimizes both search time and future storage utilization.

Binary-fit is not one of these standard names.

Example

Worked microexample

- Holes 100,500,200,300 and request 180: first-fit selects 500, best-fit 200, worst-fit 500. For base=4000,length=100, logical offset 99 maps to 4099; offset 100 traps.

Operating System module

OS29 | Internal and external fragmentation

Internal fragmentation is unused space inside an allocated unit.

External fragmentation is sufficient total free space split into holes that cannot satisfy a required contiguous allocation.

Coalescing joins adjacent free holes; compaction moves allocated regions to consolidate free space and requires relocation support.

Basic fixed-frame paging avoids external fragmentation of page-frame allocation but can waste the final page internally.

Variable-length segmentation/contiguous partitions can fragment externally.

This is not a statement that every allocator inside a paging OS is fragmentation-free.

Track units and round allocations up rather than assuming all requests exactly fill blocks.

Example

Worked microexample

- A 10 KiB object in 4 KiB pages needs ceil(10/4)=3 pages: 12 KiB allocated and 2 KiB unused. Two 3 KiB free holes cannot serve one contiguous 5 KiB request although 6 KiB is free.

Operating System module

OS30 | Paging, page tables and TLB

Pages and physical frames have equal fixed size P. Divide virtual address v into page=floor(v/P) and offset=v mod P. The page table maps a page to a frame and protection/status flags.

Physical address=frame×P+offset.

A TLB caches translations; a TLB miss may require a table walk while the page is already resident.

Table size depends on the number of virtual pages and entry size, not just physical RAM.

Protection and present bits must be checked before access.

A page-table lookup is not a CPU scheduling operation.

Example

Worked microexample

- Page size 1024, v=2053: page 2, offset 5. If page 2 maps to frame 7, physical address=7173. With 16-bit virtual addresses and 1 KiB pages, there are 64 virtual pages and a 6-bit page number.

Operating System module

OS31 | Segmentation and contrasts with paging

Segmentation uses logical variable-length units such as code, stack and arrays.

A segment-table entry provides base, limit and permissions.

For (segment,offset), validate the segment and check the bound, then add base+offset.

An out-of-range offset traps; it must not spill into the next segment.

Under the basic models, paging uses fixed units/page tables and can waste space internally; segmentation uses variable logical units/segment tables and can fragment externally.

Segments reflect program structure whereas ordinary pages are largely transparent to applications.

Apply all three contrasts when comparing paging and segmentation.

Example

Worked microexample

- Segment 1 has base 8000 and length 500. Offset 120 maps to 8120; offset 500 is invalid under the length convention.

Operating System module

OS32 | Virtual memory and page-fault handling

Virtual memory provides translated, protected address spaces and allows nonresident portions of a program to be backed by storage.

It does not add physical RAM or remove all capacity constraints.

Demand paging brings a page in when accessed.

On a nonresident access: trap, validate mapping/permissions, find a frame or choose a victim, save a dirty victim if needed, obtain contents, update mapping and restart.

Not every fault reads a disk: some allocate zero-filled pages or establish shared mappings.

An illegal access is rejected, not automatically loaded.

Excessive active-page demand can cause thrashing: repeated faults dominate useful computation.

Reducing active load or adding suitable memory can help.

Example

Worked microexample

- If access costs 100 ns and an additional page-in costs 5 ms, a fault probability of 10^−4 adds roughly 500 ns per access; rare long delays still matter.

Operating System module

OS33 | Replacement-policy traces

Replacement chooses a victim when no suitable free frame is available.

FIFO evicts the earliest loaded page; a hit does not refresh insertion time.

LRU evicts the least recently accessed and updates recency on every access.

Optimal evicts the page used farthest in the future, serving as an offline lower bound.

LFU uses reference counts and needs tie/aging rules.

Write a row per reference, identify hits before evicting, and count initial loads as faults.

FIFO may exhibit Belady's anomaly when more frames cause more faults; LRU and optimal have the stack property.

Page buffering retains useful free/modified pages to reduce effective fault cost.

Example

Worked microexample

- Three empty frames, references 1,2,3,1,4,1: FIFO faults at 1,2,3,4 and the last 1: five faults. LRU updates page 1 on its hit, evicts 2 for 4, and ends with a hit on 1: four faults.

Operating System module

OS34 | Device controllers, drivers and I/O services

A controller is device hardware exposing status, control and data registers.

A driver converts OS requests into device-specific operations.

Device independence and uniform naming hide brand-specific details; error handling reports/retries supported failures.

Polling checks readiness repeatedly; interrupts request CPU attention; DMA transfers blocks after setup without CPU work for every byte.

Both the kernel and drivers participate in I/O; specify the abstraction level being discussed.

Plug-and-play discovers/configures compatible new hardware but may still require a driver.

Input/output/storage/NIC devices are OS-managed hardware; user accounts are identities.

Printer-paper and network errors can be handled/reported, but continued operation during total power loss requires hardware support, not OS software alone.

Operating System module

OS35 | Buffering, caching, spooling and clocks

Buffering temporarily holds data in transit to absorb speed/transfer-size differences.

Caching retains copies to accelerate reuse.

Spooling queues jobs for a device such as a printer, usually on disk, so producers need not own it throughout job creation.

These have different purposes even if one implementation combines them.

The clock/timer supports timekeeping, timeouts, accounting and preemptive scheduling.

A system monitor reports CPU, memory, storage and network usage; it measures rather than directly creating more resources.

Disk initialization/filesystem creation is OS/tool work, firmware begins boot, and drive firmware may remap bad sectors.

A broad “OS does all disk functions” claim needs these responsibility boundaries.

Operating System module

OS36 | Files, descriptors and ordinary operations

A file has content and metadata such as name, size, owner, permissions and timestamps. open obtains an open-file reference, read/write transfer bytes, seek changes an offset, close releases a reference, truncate changes size and unlink removes a name.

A file descriptor is a process-local integer handle; the associated open-file state contains the current offset and access mode.

Descriptor and offset are not synonyms.

Read advances the offset on seekable files and may return fewer bytes than requested; zero normally indicates end-of-file.

Sockets also support ordinary read/write, alongside socket/connect/send operations, so read is not uniquely “not a network call.” FAT32, NTFS and ext4 are filesystem formats; HTTP is an application communication protocol.

Operating System module

Linux man-pages read(2)

OS37 | Directories, mounting and links

A single-level directory requires unique names in one namespace.

Two-level directories separate users.

A tree allows nested directories and absolute/relative paths; links can create shared graphs.

Modern general-purpose systems commonly present hierarchical namespaces, but a feature name alone does not prove that every system excludes simpler arrangements.

A hard link is another name for the same underlying file object, usually within one filesystem; a symbolic link stores a pathname and can point across filesystems or become dangling.

Removing one hard link does not erase content while other names/open references retain it.

Mounting attaches a filesystem at a mount point; it does not format the device.

NFS provides remote filesystem access; FTP transfers files rather than itself mounting a local filesystem.

Operating System module

OS38 | Contiguous, linked and indexed file allocation

Contiguous allocation uses consecutive blocks; logical block i is start+i and supports direct access, but growth and external fragmentation are problems.

Extents allow several contiguous runs.

Linked allocation chains blocks and supports growth without reserving a complete contiguous file; locating block i requires traversing preceding links.

FAT stores chains in a table; caching that table does not change the chain into a direct arithmetic mapping.

Indexed allocation uses a per-file array/tree of block pointers and supports direct lookup with metadata overhead.

A Unix inode can combine direct pointers with indirect levels.

Check pointer size, block size, index capacity and whether metadata reads are cached.

Example

Worked microexample

- A 1024-byte index block with 4-byte pointers holds 256 block addresses. With 1024-byte data blocks, one such index level addresses 256 KiB of file data, excluding metadata and any reserved entries.

Operating System module

OS39 | Protection, authentication and threats

Authentication verifies identity; authorization checks permitted operations.

An ACL lists principals and rights; owner/group/other bits are a simpler policy model, not universal defaults granting all owners full access or everyone read.

Read, write, execute, append and delete are different rights, with filesystem-specific rules.

Least privilege limits damage.

Password hashing verifies secrets without storing plaintext; one-time passwords expire after use.

Encryption protects confidentiality under correct key use but does not automatically provide integrity.

Trojans masquerade as useful software, worms propagate independently and viruses attach to hosts; denial of service harms availability.

Modern OS families can support cryptography, authentication and firewalls, but support/configuration depends on product.

CLI, GUI, speech/natural language and VR are possible interfaces, not a closed list excluding the last two.

Operating System module

OS40 | Linux, Android and product-history limits

Linux is a Unix-compatible kernel written from scratch by Linus Torvalds; a distribution combines it with userland, libraries and tools.

Android's base kernel layer is Linux; its framework and applications are higher layers.

AOSP is open source, while a shipped device can include proprietary applications/services.

Windows and macOS are both proprietary products, although macOS contains open-source components, so "which is not open source?" offering both is ambiguous.

Microsoft Office is an application suite.

Windows is the Microsoft family commonly described as succeeding the MS-DOS desktop era; Solaris originated at Sun. "First widely used OS," "first multitasking OS," and generation/function pairings require a definition and historical evidence; UNIX was not the first system ever to multitask. "Most mobile devices" needs a population and date; architecture facts do not prove market share.

Operating System module

Android Open Source Project architecture

Linux Kernel Organization, About

Oracle Solaris 11

OS41 | Identify the operating-system conditions

Check whether the stated conditions support one answer or several before selecting a response.

A general system-call purpose should say requesting a kernel service; a process purpose should say executing a program.

CPU, memory and I/O management span multiple managers; device, process, file and security management describe particular responsibilities.

Virtual memory provides isolation and translated space; it does not remove every memory constraint.

Specify the OS/API, scheduling variant, arrivals/quantum, exact graph, base-limit convention or scope of comparison needed to answer the question.

Support the answer with the relevant operating-system rule and stated assumptions.

A process/resource graph question requires its actual edges; theory alone cannot reconstruct a missing diagram.

Operating System module

OS-S01 | UNIX origins and the first four historical generations

UNIX development began at Bell Labs in 1969, led by Ken Thompson and Dennis Ritchie with other collaborators. Ritchie is therefore a co-creator, not the sole inventor of the first operating system.

The later UNIX time-sharing system described by Ritchie and Thompson is multiuser and interactive and supports asynchronous processes. Its later versions incorporated multiprogramming. This history does not make UNIX the first system ever to multitask.

In the first four generations of the cited NYU teaching taxonomy, the first generation uses vacuum tubes and manual single-user operation without a separate resident OS. The second generation pairs transistors with uniprogrammed batch systems; the resident OS starts successive jobs without being reloaded for every job.

The third generation pairs integrated circuits with multiprogramming. The fourth emphasizes personal computers. These are characteristic-era associations, not claims that each feature existed exclusively in one era.

Nokia Bell Labs - 50 years of Unix

Ritchie and Thompson - The UNIX Time-Sharing System

NYU OS 2250 - History of Operating Systems

OS-S02 | Thread management costs and execution privilege

In the conventional library-thread comparison, user-level thread creation and switching can avoid entering the kernel. Kernel-managed thread creation and management commonly incur additional kernel work and are therefore generally slower to create and manage in this comparison. This is a typical comparison, not a universal speed or reliability guarantee.

Being scheduled or managed by the kernel does not give application thread code kernel privilege. Ordinary applications and supporting utilities execute in restricted user mode and request protected services through libraries and system calls. Kernel component code executes in privileged kernel mode and performs permitted low-level hardware operations.

Operating System module · page 28

Operating System module · page 29

Operating System module · page 53

OS-S03 | Memory allocation and page-policy decisions

Memory management tracks which primary-memory regions are free or in use, assigns memory to processes and data, and deallocates it when no longer needed or when a process terminates. Allocation and deallocation concern memory ownership; CPU selection belongs to scheduling.

A buddy allocator manages power-of-two block sizes. It takes a smallest sufficient block, splitting larger blocks when necessary, and can merge a freed block with its free buddy. This is not the worst-fit rule of selecting the largest arbitrary hole.

A fetch policy decides when contents are brought into memory; demand paging fetches a valid nonresident page when it is accessed. A cleaning policy concerns writing modified contents to backing storage. A replacement policy chooses which resident page to evict. These decisions can cooperate during one page fault, but they are different decisions.

Operating System module · page 5

Operating System module · page 31

Operating System module · page 38

Operating System module · page 40

Operating System module · page 42

Linux kernel documentation - Physical Memory, free_area

OS-S04 | Communication endpoints and timed I/O events

In the descriptor-based socket API, socket() creates a communication endpoint and returns its descriptor. connect() associates the socket with the requested peer address; send() transmits data to another socket. Receiving through read() uses the ordinary descriptor-input interface described in the file-services lesson.

IRQ means interrupt request. A controller can raise an interrupt to request CPU attention. In a clock-tick scheduler, the scheduler initializes a running process's quantum counter; each clock interrupt decrements it, and reaching zero triggers scheduling. Clock support also maintains time of day and usage accounting.

Direct message communication names the receiving process explicitly. Indirect communication uses a mailbox or port. This naming distinction is independent of whether a send or receive blocks: naming a destination does not determine synchronization semantics.

Linux man-pages - socket(2)

Linux man-pages - connect(2)

Linux man-pages - send(2)

Operating System module · page 45

Operating System module · page 46

UIC Operating Systems - Processes, message-passing naming and synchronization

OS-S05 | Compilation versus operating-system execution services

A compiler translates source code toward an executable representation. For example, the conventional GCC toolchain distinguishes preprocessing, compilation proper, assembly and linking. A compiler uses operating-system services as a development tool; it is not the runtime kernel component that selects ready processes, manages memory or handles protected service requests.

GCC - Options Controlling the Kind of Output

OS-S06 | Filesystem links in Linux

Linux provides link() to create another hard-link name for an existing file. The old and new names identify the same file, so hard links are not a Windows-only facility. unlink() removes a filesystem name; when that name is a symbolic link, it removes the link itself. Removing a name and erasing the target’s contents are different operations.

Linux man-pages · link(2)

Linux man-pages · unlink(2)

Software Engineering

Complete Study Notes

SE01 | Software engineering, processes and work products

Software engineering applies systematic methods to development, operation and maintenance.

A software product includes executable code and relevant requirements, designs, test assets, configuration and user documentation.

A process defines activities, ordering, roles, products and entry/exit conditions.

Its four broad activities are specification, design/implementation, validation and evolution.

Procedural decomposition organizes functions; object oriented decomposition organizes objects with state and behavior.

Either can use iterative or sequential development.

OOP does not automatically shorten every project.

Reuse, encapsulation and consistent models can reduce duplication and ease change; poor object design can still be complex.

Functional programming emphasizes function evaluation and controlled side effects; event-driven programs react to events.

Software Engineering module

SE02 | Lifecycle phases and model selection

A common lifecycle names requirements, design, implementation, testing, deployment and maintenance.

Analysis clarifies the problem; system design chooses architecture; object/detail design specifies interfaces and algorithms; implementation writes code; testing evaluates behavior; deployment puts it into service.

These activities may overlap and repeat.

Select a process using requirement stability, technical risk, delivery urgency, stakeholder availability, team skills and assurance needs. “Team has little experience” alone does not uniquely identify a process.

A risk-driven cycle, prototype or carefully planned staged process may each help, depending on the actual uncertainty.

No process eliminates the need for testing or documentation.

Software Engineering module

SE03 | Waterfall and the V-model

Classical waterfall separates planned phases with reviewed deliverables.

It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly.

It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback.

Rigidity offers milestone visibility but can freeze incorrect assumptions.

The V-model pairs specification/design stages with later test levels.

Requirements inform acceptance tests; system design informs system tests; component/detail design informs integration/unit tests.

Coding lies at the bottom.

Test planning starts early.

Stable requirements and strong assurance may favor this organization, but neither a V shape nor waterfall guarantees safety or correctness.

Software Engineering module

SE04 | Iteration, increments and early delivery

Iteration revisits and improves a solution; an increment adds usable functionality.

A process may do both.

Delivering registration first, then payments, then reporting is incremental; revising a registration prototype after feedback is iterative.

Each increment still needs design, implementation and testing.

Choose incremental delivery when prioritized partial capability must reach users early.

Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning.

Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules.

Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Software Engineering module

SE05 | Agile values, stories and Scrum

Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans.

The latter still have value.

Agile does not mean no planning or no documentation.

A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done.

Scope is discussed before and during planning; it is not forbidden to agree scope.

Timeboxes tend to stay fixed while content is adapted without endangering the goal.

Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Software Engineering module

Agile Manifesto

Scrum Guide

SE06 | Spiral development and risk

Spiral development is risk driven.

Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment.

A prototype may test a risky technology before a large investment.

The amount and type of work depend on the risks.

Use risk emphasis to distinguish spiral from a generic changing-requirements scenario.

Spiral can incorporate incremental, prototyping or sequential activities, but calling it the uniquely “most general” process is a textbook convention rather than a universal theorem.

Risk analysis requires skill and may impose excessive overhead on a small low-risk project.

Software Engineering module

SE07 | Prototypes, RAD and reuse-oriented development

A prototype explores requirements, usability or technical feasibility.

A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production.

A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation.

Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

RAD uses rapid construction, timeboxing, user participation and often reusable components; it requires available users and skilled designers/developers.

Reuse-oriented development searches components, evaluates fit, may adjust requirements, designs around interfaces and builds missing parts.

Product-line assets may include requirements, architecture, components and tests; all are potentially reusable.

Software Engineering module

SE08 | Feasibility and levels of requirements

Feasibility asks whether a proposal is worth doing and can be delivered within technical, economic, operational and schedule constraints.

Examine existing limitations, organizational impacts, benefits, cost, available skills and integration risk.

System analysis studies the problem and needed behavior in more detail; the activities inform each other.

Business requirements explain why the project exists and its goals.

User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints.

The Software Requirements Specification (SRS) organizes the agreed requirements.

A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Software Engineering module

SE09 | Functional requirements include prohibited behavior

A functional requirement states an observable service or behavior, including reactions to inputs, errors and forbidden actions. “When the sensor detects intrusion, send an alert” is functional. “Only a manager may approve refunds” also specifies permitted/prohibited behavior.

A negative statement is not automatically nonfunctional.

Write one clear obligation with an actor/trigger, action and observable result. “The system shall generate a PDF report” is functional. “The system shall not reveal another customer’s balance” can be a security-related functional rule.

Security as a broad quality goal and the functions implementing it must be distinguished.

Words such as must, should and not work do not by themselves determine a requirement’s classification.

Software Engineering module

SE10 | Nonfunctional qualities and measurable constraints

Nonfunctional requirements include quality attributes and constraints.

Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning.

Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

“Search responds within two seconds under 100 concurrent users” is measurable performance. “New users finish checkout within one minute without help” is usability. “Use Java” is a technology/pseudo-requirement, often separately classified as a system constraint.

Dependability includes reliability, safety and robustness; portability is not its usual central category.

Performance degradation at peak load primarily concerns performance, not automatically reliability.

Software Engineering module

SE11 | Validate requirements and maintain traceability

Correct requirements reflect actual stakeholder needs.

Complete requirements address the agreed scope, normal and exceptional behavior.

Consistency excludes contradictions.

Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints.

Verifiability means objective evidence can check fulfillment.

Replace “fast for most users” with a defined metric, workload and threshold.

Trace each requirement backward to its source/rationale and forward to design, code and tests.

Record dependencies among requirements.

A traceability matrix links these artifacts and helps impact analysis; it is not just a cost/schedule ledger.

Stakeholders must negotiate conflicting requirements rather than independently approving incompatible demands.

Scope, understanding and volatility are distinct elicitation difficulties.

Software Engineering module

SE12 | Elicitation techniques, FAST and QFD

Interviews ask stakeholders about needs; document analysis examines manuals, contracts and current records; observation/ethnography studies actual daily work.

Prototypes reveal tacit expectations.

Workshops/JAD bring users, managers and developers together with a facilitator and recorder to build consensus.

Use several sources and check assumptions against observed work.

In the taught FAST technique, participants list environment elements, outputs and inputs, consolidate lists and develop smaller specifications.

QFD prioritizes customer value: normal needs are explicitly stated, expected needs are assumed, and exciting needs provide unanticipated benefit.

Brainstorm first to generate ideas, then evaluate and prioritize; do not confuse that with letting every idea become a committed requirement.

Software Engineering module

SE13 | From scenarios to analysis objects

A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios.

Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models.

Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case.

A UI form is normally boundary, not entity.

Use nouns as candidate objects, possessive descriptions as candidate attributes, and verbs such as manages/reports-to as candidate associations.

These are heuristics, not automatic parsing rules.

CRC means Class, Responsibilities, Collaborators.

A control per use case or participating actor can be a useful starting point, not a mandatory UML rule or fixed object lifetime.

Software Engineering module

SE14 | Actors, use cases and scope

An actor is an external role interacting with the system; it can be a person, device or another system.

A person may play several roles, and several people may play one role.

A primary actor seeks a goal; a supporting actor provides a service.

Ask which user groups perform main functions, administer or maintain the system, and which external systems exchange information.

Draw actors outside the system boundary rectangle and use-case ovals inside.

Name a goal with a verb and noun.

The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout.

Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Software Engineering module

SE15 | Write a complete use case with alternatives

Record name/goal, actors, trigger, preconditions, main success flow, alternatives/exceptions and postconditions.

Use active subject-verb steps at one abstraction level; separate errors from the normal flow.

Example: Submit Incident.

Precondition: officer authenticated. 1 Officer supplies location and description. 2 System validates details. 3 System records the incident. 4 System returns its ID.

Alternative 2a: location missing → report the problem and request correction; save nothing.

Success postcondition: incident and ID persist; failure guarantee: no partial record.

An alternate flow is a conditional variation; an exception flow handles a failure.

Preconditions hold before entry, while a trigger starts the use case.

Software Engineering module

SE16 | Include, extend, generalization and abstract use cases

Include factors required reused behavior: Submit Order includes Validate Order, with the dashed arrow from the including/base use case to the included one.

Extend adds conditional behavior at extension points: Apply Discount extends Checkout, with the arrow from the extension to the base.

The base is meaningful independently.

For an extension, draw the arrow from the extending use case to the base use case.

Generalization relates a specialized actor/use case to a more general one, with a hollow triangle toward the general element.

An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case.

It is not defined merely by reuse or lack of a direct actor line.

Included behavior is incorporated where the inclusion is reached; do not infer every conditional branch in the base always executes.

Software Engineering module

SE17 | Class diagrams, visibility and multiplicity

A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility.

A class diagram shows static types and relationships, while an object diagram shows particular instances/links.

UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process.

Structural things describe conceptual or physical structural elements.

Multiplicity at one end counts how many objects there may correspond to one object at the opposite end: 1 exactly one, 0..1 optional, 0..* any nonnegative number, 1..* at least one.

A one-to-many association does not automatically mean composition.

Read each end separately before choosing cardinality.

Software Engineering module

SE18 | Association, aggregation, composition and generalization

Association connects related instances.

A dashed dependency arrow means a client uses a supplier.

Generalization uses a solid line and hollow triangle pointing to the superclass/general element.

Shared aggregation uses a hollow diamond at the whole end and weak whole-part meaning; it does not by itself define a detailed lifecycle policy.

Composition uses a filled diamond at the whole end and strong ownership: a part is in at most one composite at a time, and parts still owned are destroyed with that composite.

Parts may be detached/transferred under a defined model.

Thus “cannot ever exist independently” is an oversimplification.

For Car composing Engine, the diamond belongs beside Car; saying it points to Engine is imprecise.

The Car-Engine example describes composition.

Software Engineering module

OMG UML2.5.1

SE19 | Sequence diagrams and the emergency-report example

Sequence diagrams show interactions across lifelines, with order/time generally downward.

Messages cross between participants; activation rectangles show execution.

A creation message starts a lifeline; an X marks modeled destruction, not a guarantee about when a garbage collector runs.

Synchronous calls wait for completion; asynchronous messages permit the sender to proceed.

One interaction-diagram layout heuristic puts the initiating actor first, boundary second, control third, then entities.

This is not a required UML column order.

In ReportEmergency, FieldOfficer activates ReportEmergencyButton, enters details through EmergencyReportForm, and later sees AcknowledgmentNotice; DispatcherStation belongs to the dispatcher side.

Choose the button for initiating this particular scenario.

Software Engineering module

SE20 | Activity, state and data-flow models

An activity diagram models workflow: filled circle start, rounded actions, guarded decision diamond, thick fork/join bars and partitions/swimlanes for responsibility.

A diamond can also merge alternative flows; a join synchronizes concurrent flows.

A state machine models states and transitions triggered by events, often with guards/actions, for one object's lifecycle.

Choose a sequence diagram for message order between participants, a state machine for event-dependent state changes, and activity for workflow.

Both sequence and state diagrams describe behavior over time.

State machines are useful for objects with significant states, whether entity or control.

A DFD shows data stores/flows and transformations from inputs to outputs; it is not a UML diagram.

Software Engineering module

SE21 | Design and subsystem decomposition

System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies.

Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms.

A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Group closely related objects and functionality, minimize associations crossing boundaries, and isolate communication infrastructure where useful.

User workflows inform decomposition but do not alone dictate the internal structure.

Analysis describes domain needs; subsystem decomposition is principally a design result.

Architecture documentation supports communication and records major decisions, not every source-level execution step.

Software Engineering module

SE22 | Cohesion, coupling, modularity and refactoring

Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules.

Seek high cohesion and low coupling.

Shared mutable data and knowledge of another module's internal representation increase coupling.

Information hiding exposes stable operations while concealing changeable decisions.

Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task.

Splitting an unrelated three-purpose component improves modularity.

If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers.

Smaller modules can aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Software Engineering module

SE23 | Layers, tiers and architecture views

A layer is a logical abstraction level; a tier is a deployment separation.

A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping.

For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

A structure is the elements/relations themselves; a view represents selected aspects.

Module structures show code organization, component-and-connector structures runtime interaction, and allocation structures map software to hardware, teams or environment.

Processes/threads belong in concurrency views.

Evaluate architectures against quality attributes and scenarios; processor allocation, decomposition and style selection can all be architectural concerns.

Software Engineering module

Rose-Hulman: What is software architecture

SE24 | Architecture styles and integration boundaries

Repository style shares central data, simplifying common access but creating bottleneck/schema dependencies.

MVC separates domain model, display view and interaction controller.

Client-server separates requesters from service providers; peer-to-peer permits both roles.

Pipe-and-filter applies successive transformations to data streams.

Three-tier separates presentation, application logic and storage; some four-tier variants split presentation client/server.

An API gateway presents one entry point and routes/combines calls to services, hiding service boundaries from clients.

It can centralize policies but also become a bottleneck.

A component diagram shows components/interfaces/dependencies; a deployment diagram places software artifacts on nodes.

Choose the style for the required interactions and qualities rather than selecting a universally best architecture.

Software Engineering module

SE25 | Design principles and operation contracts

Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities.

Top-down design refines high-level structure; bottom-up design assembles existing components.

A hybrid combines both.

SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion.

Depend on abstractions; add compatible extensions; avoid forcing unused operations.

DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

A contract states preconditions, postconditions and invariants.

For withdraw(amount): require amount>0 and amount<=balance; ensure balance=oldBalance-amount; preserve balance>=0.

Validation checks inputs; implementation enforces the transition.

A complete feature needs both UI/API and persisted behavior, not only a diagram.

Software Engineering module

University of Washington: Software Design and Style

SE26 | A small analysis-to-implementation example

For Submit Incident, let a boundary read input, a service validate and coordinate, and a repository save an Incident entity.

Pseudocode: submit(input){require authenticated; if missing(location) return validationError; incident=new Incident(input); repository.save(incident); return incident.id;}.

Save and ID assignment must succeed together, or roll back.

Derive tests from the model: missing location saves nothing; valid input returns a saved ID; a storage failure yields an error without a partial record; unauthorized input is rejected.

Trace requirement→use-case step→service operation→repository effect→test.

Use the implementation exercise to connect analysis and design tools to working code.

Software Engineering module

SE27 | Patterns, variability and testability

Factory Method lets subclasses choose a concrete product through a creation operation, reducing coupling to concrete constructors.

A simple factory centralizes creation but is not identical to Factory Method.

Singleton controls access to one instance; Observer notifies subscribers; Memento stores/restores state while hiding representation; Iterator traverses; Decorator adds behavior through wrapping. The pattern name is Memento, not Momento.

Variability can use selectable features, omission/inclusion and extension points; overloading supplies alternate signatures in code but is not every variability mechanism.

Testability improves with controllable inputs, observable outputs, replaceable dependencies and limited structural complexity.

A sandbox can isolate execution.

Assess each design claim independently before evaluating a combined statement.

Software Engineering module

Design Patterns introduction, university-hosted author text

SE28 | Planning, WBS, estimation and tools

Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes.

A WBS is hierarchical and covers project scope without adding unrelated work.

Tools help coordinate ownership, dependencies and progress; they do not replace judgment or automatically fix bugs.

Function points estimate functional size using inputs, outputs, inquiries, internal logical files and external interface files, weighted under the chosen method.

They are not just lines of code and are not a direct exact measure of complexity.

Git tracks versions; issue trackers such as Jira track work/defects; CI tools such as Jenkins automate checks; Docker packages application/runtime dependencies into containers.

Software Engineering module

SE29 | Critical path: forward and backward passes

In a dependency network, ES=max predecessor EF and EF=ES+duration; start tasks have ES=0 unless another constraint applies.

Project earliest finish is the maximum terminal EF.

Backward, LF=min successor LS and LS=LF-duration, starting from the required finish.

Total float/slack=LS-ES=LF-EF.

A critical path has the longest total duration and normally zero float in an unconstrained schedule.

If A takes 5 days and B=3,C=4 both follow A, project duration is 5+max(3,4)=9 days.

B finishes at 8, but the project waits for C at 9.

If paths B5+C6 and D8+E3 share a zero-duration start/end, both total 11; there are two critical paths, not one.

Software Engineering module

SE30 | Slack assumptions and PERT calculations

Early start/finish values alone do not determine slack without successor relationships and the project completion constraint.

If C3→D6 is a complete path parallel to A4→B5 and both finish at day9, C has zero float.

That network cannot be silently inferred from four unrelated ES/EF rows.

Traditional CPM uses point durations; PERT models uncertainty with optimistic a, most-likely m and pessimistic b. The classic estimate is TE=(a+4m+b)/6 and variance=((b-a)/6)^2.

For 3,6,15 days, TE=42/6=7 days and variance=4 days².

Do not replace the weighted estimate by a simple average.

These formulas are approximations, not guaranteed actual completion times.

Software Engineering module

PMI: PERT estimation

SE31 | Business value, risks and resource optimization

With benefits and costs already discounted to a common date, ROI=(benefits-costs)/costs*100%.

Birr120000 benefits and Birr100000 costs give 20%, not 120%.

Identify project risks to resources/schedule, product risks to quality, and business risks to the organization.

Analyze likelihood and impact, plan responses and monitor.

Avoidance changes the plan to remove a risk, mitigation reduces likelihood/impact, transfer assigns consequences contractually, and retention/acceptance keeps the risk; passive acceptance may take no action.

Resource leveling adjusts dates to respect resource availability and can change project finish; smoothing uses available float within schedule constraints.

Neither magically creates extra staff.

Software Engineering module

PMI Lexicon of Project Management Terms

SE32 | Team organization, development and maturity

Functional organizations group specialists by discipline; project organizations group people around projects.

Functional specialization helps staffing and knowledge continuity but may create handoff barriers.

Democratic teams share decisions; chief-programmer teams centralize technical coordination.

These are different classification dimensions.

Tuckman’s sequence is forming→storming→norming→performing→adjourning: orientation, conflict, norms, effective work and closure.

Historical Software CMM levels are 1 Initial, 2 Repeatable, 3 Defined, 4 Managed, 5 Optimizing.

Level2 establishes basic project discipline including requirements management, project planning, quality assurance and configuration management.

These historical labels must not be silently replaced by modern CMMI terminology.

A maturity model concerns process capability, not a guarantee that every product is defect free.

Software Engineering module

SEI: Software CMM v1.1

UC Berkeley Team Building Toolkit

SE33 | Configuration, change control and version control

Configuration management identifies controlled items, versions/baselines, change procedures, status and audits.

Items may include source code, requirements, manuals, test plans, environments and build configurations.

A team's ordinary browser is usually a tool rather than product CI, but a controlled test-browser version may be an environment CI.

Change management handles request, impact assessment, decision/prioritization, implementation and validation; configuration management keeps all affected versions consistent.

Distributed Git gives each clone local history and supports independent branches followed by merging; it does not eliminate merge conflicts or approvals.

Plan test configuration management early and apply it throughout testing, not at just one exclusive phase.

Software Engineering module

SE34 | Quality assurance, verification and validation

Quality assurance establishes and checks processes/standards; quality control evaluates work products.

An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records.

Quality of conformance means following the design/specification; a product can conform to a poor specification.

Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs.

Both can use reviews, analysis or execution, so "human-only versus computer-only" is false.

A fault/defect is an underlying problem; an error may be an incorrect internal state; a failure is an externally observed departure from required service.

Reliability is a product quality attribute; low cost or short development time are project outcomes.

Tests increase confidence and reveal defects but cannot generally guarantee correctness over every possible input.

Software Engineering module

SE35 | Testing levels and acceptance

Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency.

Integration testing exercises interfaces and interactions among combined parts.

System testing checks the complete integrated system against functional and nonfunctional requirements in a representative environment.

Acceptance testing supports the customer/user's acceptance decision against agreed needs and criteria.

Alpha typically happens in a controlled developer setting with selected users; beta uses external users in their environments.

None of these levels is universally the longest.

Unit is a level, while white-box is a technique; a unit can be tested using black-box cases too.

Software Engineering module

SE36 | Test types, regression and exploratory work

Functional tests check required behavior against the SRS and acceptance criteria.

Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits.

Usability, security, compatibility and reliability tests target their respective qualities. "Normal operating conditions" alone does not distinguish functional from performance testing.

Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge.

Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair.

Exploratory testing combines learning, design and execution and is experience based.

Manual/automated describe execution support, not a compulsory black-/white-box classification.

Software Engineering module

SE37 | Equivalence, boundaries and decision tables

Equivalence partitioning groups inputs expected to behave alike, then samples valid and invalid groups.

For allowed age18..60, useful partitions are below18,18..60,above60.

Boundary tests target edges and neighbors, such as 17,18,19,59,60,61, with the exact set chosen under the stated coverage scheme.

A decision table lists condition combinations and resulting actions.

For validCard and enoughFunds, enumerate TT,TF,FT,FF; specify dispense only for TT and the proper rejection for others.

Cover feasible rules rather than blindly using every impossible combination.

These are specification-based techniques.

Test requirements can also define state-transition cases, such as rejecting dispatch from an already closed incident.

Software Engineering module

ISTQB CTFL4.0.1 syllabus

SE38 | Statement, decision and branch coverage

Statement coverage is the number of distinct executable statements exercised divided by the total executable statements in the declared test scope.

Unreachable statements can prevent100%; any justified exclusions must be documented with the adjusted scope.

Decision coverage counts decision outcomes; branch coverage counts control-flow edges, including unconditional edges under the stated convention.

Testing both true and false of each decision may need more than two overall test cases when branches nest.

Coverage is evidence of exercised structure, not proof of all behavior.

In the width/length/height program, the outer if has separate inner decisions in each branch.

Need outer true with inner true and false, and outer false with its inner true and false: four tests, for example (w,l,h)=(3,2,4),(3,2,1),(2,3,4),(2,3,1).

One run cannot exercise both outer branches.

All paths may be infeasible or infinite with loops; specify the chosen coverage target.

Software Engineering module

ISTQB CTFL syllabus v4.0.1, sections 4.3.1-4.3.2

SE39 | Test process, plans and defect reports

Plan objectives, scope, resources, schedule, risks and entry/exit criteria.

Analyze the test basis for testable features and defects; design cases/expected results; implement data, scripts and environment; execute and compare; report defects; evaluate completion.

Analysis asks what to test and assesses testability; design asks how.

Monitoring/control run across these activities.

A test plan describes intended work; incident/defect reports are normally execution outputs.

A defect report should identify build/environment, steps, expected versus actual behavior, evidence and severity.

Anyone finding a fault should record it; testers often coordinate reporting, but it is not their exclusive permission.

Configuration control makes the tested version reproducible.

Software Engineering module

ISTQB CTFL4.0.1 syllabus

SE40 | Maintenance and software evolution

Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code.

Motive determines category; refactoring can support preventive or perfective work.

Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

A typical evolution sequence is change request→impact analysis→release planning→implementation/testing→release.

Among a list lacking the request step, impact analysis may be the earliest listed technical activity.

Major restructuring is reengineering; if the architecture itself changes, architectural transformation is the more specific description.

Markets, technology, stakeholders and funding can all change scope or priorities.

Software Engineering module

SE41 | Apply network calculations and compare design claims

The earliest-finish example yields 9 days.

Zero slack follows only from an explicit completion-path assumption when latest times or the full network are unavailable.

In the path-duration example, two paths each total 11 days.

Use the network rules to justify the calculation before comparing it with an answer key.

Evaluate descriptions of architectural structures and views separately; both can be true.

Evaluate the individual claims before considering a combined option, especially if the option refers to itself.

Requirements, tests and architecture can all be reusable product-line assets.

When several choices satisfy the rules, identify the needed distinction rather than selecting a false rule.

Software Engineering module

SE42 | Context-dependent claims and model limitations

Context matters: no test level is always most time consuming; no process is determined by team inexperience alone; no waterfall rule makes every later change impossible.

Requirements may describe required and prohibited behavior, including interface functions as well as usability qualities.

Identify an interface requirement's exact obligation before classifying it as functional or nonfunctional.

Sequence/state diagrams both show dynamic behavior.

Composition needs the diamond's placement and ownership semantics. "Agile least suitable for safety-critical government work" assumes a particular lightweight process without required assurance; it is not universally true.

Treat the guide's conventions as scenario hints and keep the stronger engineering rules explicit.

Software Engineering module

SE43 | Design choices, evidence and economical improvement

Choose designs by their effect on stated goals.

Fine-grained self-contained components, separation of producers/consumers and limited shared mutable data often improve maintainability.

Trade-offs remain: interfaces can add latency, isolation can complicate transactions, and global state can simplify one operation while coupling many others.

Prefer measurable quality scenarios to counting components alone.

Pareto's 80/20 rule is a prioritization heuristic: a small subset of causes may account for much of the observed impact.

It does not guarantee 80% of defects can be fixed with exactly 20% effort.

Refactor to preserve behavior while improving structure; optimize only against measured needs; record rationale and update traceability and regression evidence after change.

Software Engineering module

SE-S01 | Scope, understanding and changing requirements

Elicitation difficulties have several distinct sources. Scope problems concern agreeing on system boundaries and the appropriate level of detail. Understanding problems arise when stakeholders and analysts have different domain knowledge, omit tacit needs or communicate conflicting interpretations. Volatility means that requirements change as circumstances and understanding develop.

These difficulties call for repeated communication, explicit boundaries, negotiated meanings and controlled revision. They can occur together; resolving one does not automatically remove the others.

CMU SEI - Issues in Requirements Elicitation

SE-S02 | Objects, shared models and specialization

An object-oriented approach can use a consistent vocabulary of objects, classes and relationships across analysis, design and implementation. Models are refined for different purposes rather than being identical artifacts. This continuity can reduce translation errors and effort; reusable classes can shorten suitable projects, but neither shorter duration nor good design is guaranteed.

Generalization places common concepts in a more general superclass and specialized concepts in subclasses. Available inherited attributes and operations are shared by subclasses, which may extend or override behavior. Defining common features once can reduce duplicated maintenance. The top of the hierarchy is more general, not more specialized.

Encapsulation groups state with operations and hides representation behind an interface. Polymorphism permits different object types to respond differently through a common operation or interface. Abstraction exposes the relevant contract while omitting unnecessary detail. Reusability means using suitable existing classes or components in new contexts rather than writing them again. These ideas support each other but name different properties.

Software Engineering module · page 31

Software Engineering module · page 32

Software Engineering module · page 38

Software Engineering module · page 104

Software Engineering module · page 141

SE-S03 | Modeling categories and data diagrams

In the course's UML building-block terminology, structural things describe static conceptual or physical elements. Behavioral things describe dynamic interactions and state behavior. Grouping things organize model elements into packages. Annotational things attach explanatory remarks or constraints, usually as notes.

An entity-relationship diagram models entity types, their attributes and the relationships between them, commonly for data design. Its focus differs from a use-case diagram of external

roles and goals, a class diagram of classes and their features, or a deployment diagram mapping artifacts to execution nodes.

Software Engineering module · page 45

Software Engineering module · page 46

IBM - Entity relationship diagrams

SE-S04 | Plan-driven and adaptive planning

In a plan-driven process, activities are planned in advance and progress is assessed against that plan. Agile planning is incremental and adapts to changing needs. This distinction concerns planning strategy; a process can combine advance planning with staged deliveries and feedback.

Software Engineering module · page 13

SE-S05 | Exploration artifacts and collaborative implementation

A storyboard sketches a sequence of moments in a proposed interaction, helping people explore how a concept would be used. A wireframe sketches interface structure and the arrangement of content or controls. Both can inform requirements, but neither automatically supplies a complete use-case description with actors, preconditions, flows and postconditions.

Pair programming is a way for two programmers to work together on the same implementation task, commonly sharing a workstation with complementary writing and reviewing roles. It is an implementation collaboration practice rather than a replacement for modeling a user goal.

IDEO Design Kit - Storyboard

IBM Enterprise Design Thinking Toolkit - Wireframes

Agile Alliance - Pair programming

Software Engineering module · page 70

SE-S06 | Distinguish a requirements specification from neighboring terms

A Software Requirements Specification records agreed software services and constraints. A Capability Development Document is a defense capability-requirements artifact describing needed capability characteristics and performance parameters; its specialized acquisition context differs from the ordinary software SRS. Domain-driven design is an approach that organizes software models around the business domain, not the name of an agreed requirements specification.

Software Engineering module · page 62

U.S. Army - Getting Requirements Right

Microsoft - Designing a DDD-oriented microservice

SE-S07 | Scalability and robustness are different qualities

Scalability is the ability to accommodate additional workload while maintaining the required service, often by increasing available resources. It concerns growth in demand. Compatibility concerns supported cooperating environments, such as browsers and devices; usability concerns successful and learnable user tasks.

Robustness concerns reliable operation across varied or difficult operating conditions and controlled failure outside the supported range. It is different from usability, which concerns people successfully learning and carrying out tasks. Improving an interface's task success does not by itself establish robustness under adverse conditions.

Microsoft - Scaling cloud-native applications

NIST CSRC Glossary - Robustness

Design and Analysis of Algorithms

Complete Study Notes

DAA01 | Specify the problem and prove correctness

State inputs, output, constraints and empty-solution policy.

Algorithms are finite, definite executable methods independent of source language; feasible satisfies constraints, optimal is best feasible.

Conditional branches are legitimate control flow.

Prove an invariant initially, after each step and at termination; for recursion prove base cases and the inductive reduction.

Testing finds failures but finite tests do not prove correctness for all inputs.

Design and Analysis of Algorithms module

DAA02 | Cost model, experiments and input size

Analysis counts operations and storage as functions of input size: array length n, graph vertices V/edges E, or the number of bits in integers.

A RAM model commonly gives constant cost to bounded-word arithmetic/access.

Arbitrarily large integer arithmetic and printing long outputs need their own costs.

Elapsed measurements also depend on hardware, compiler, cache, I/O and competing load.

Experimental/dynamic measurement observes actual runs; asymptotic analysis abstracts growth; probabilistic analysis studies expectations.

There is no universal taxonomy proving “graph analysis” or “dynamic analysis” is invalid.

Auxiliary space excludes input; I/O complexity counts transfers.

Representation affects costs; “data type complexity” is not a standard metric name.

Design and Analysis of Algorithms module

DAA03 | Input cases and bound notation are independent

At fixed size n, best case minimizes cost, worst case maximizes it, and average case takes an expectation under a specified input distribution.

Randomized expected cost may average internal choices for one fixed input.

Smallest input size is not best case.

O/Ω/Θ bound functions independently of case or resource.

A worst-case upper bound covers every size-n input, but O does not mean worst case.

Constant best case and quadratic worst case are not universal.

Design and Analysis of Algorithms module

DAA04 | Formal asymptotic bounds and constants

For eventually nonnegative f, g, $f \in O(g)$ means $f(n) \le c \cdot g(n)$ for all $n \ge n_0$, with fixed $c > 0$. Ω reverses that inequality; Θ requires both bounds.

Little-o means $f/g \to 0$, and little-ω means $f/g \to \infty$ where the ratio exists. Θ specifies growth up to constant factors, not exact seconds or identical best/worst cases.

Prove the inequality for the whole tail, not selected values.

Transitivity allows chained upper bounds; sums are dominated by the largest nonnegative growth term.

Example

Worked microexample

- $10n+5 \le 15n$ for $n \ge 1$, so $c=15, n_0=1$ works. Also $3n^2+4n+1 \le 8n^2$ for $n \ge 1$. The inequality $7n^2+3n+5 \le 8n^2$ is false at $n=1$; $c=15$ works for all $n \ge 1$.

Design and Analysis of Algorithms module

DAA05 | Growth ordering and loose bounds

Representative growth is $1, \log n, n, n \log n, n^2, n^3, \ldots, b^n, n!$ for fixed $b > 1$.

Log bases differ by a constant.

Both $n \log n$ and n^2 are superlinear; $\log n$ is sublinear.

Polynomial growth has a fixed exponent in input size.

$O(n^2)$ also implies $O(n^3)$: upper bounds need not be tight.

Several asymptotic bounds can hold simultaneously for the same function.

Neither representative growth ordering nor Big-O labels alone determine finite-input speed.

Example

Worked microexample

- For $f=4n^2$ and $g=8n$, $f/g=n/2 \to \infty$. Therefore $f \in \Omega(g)$ and even $\omega(g)$, while f is neither $O(g)$ nor $\Theta(g)$.

Design and Analysis of Algorithms module

DAA06 | Loops, exact counts and additive costs

Sequential costs add; independent nested loop counts multiply.

A dependent loop requires summation. n iterations of a constant body give Θ(n); ∑i=1.. n i=n(n+1)/2 gives Θ(n^2).

Doubling i=1,2,4,… until i≥n takes ceil(log 2 n) iterations for n>1.

A 10n loop is linear under constant-cost output.

If m operations each have expected cost f(n) at the same fixed size, expected total is m f(n), by linearity of expectation; independence is unnecessary.

If sizes change, sum f(n_i).

Exact coefficients depend on whether tests, increments, arithmetic and assignments count separately.

Example

Worked microexample

- For sum=0; for(i=1;i<=n;i++) sum=sum+1; return sum: counting two initializations, n+1 tests, n increments,2n body operations and one return gives 4n+4. A different primitive-cost convention changes this exact expression, not Θ(n).

Design and Analysis of Algorithms module

DAA07 | Analyze conditional nested loops and code validity

For the loop i=0.. n−1, j=i.. i^2−1, and k=0.. j−1 only when j%i==0, i=0 and 1 have empty j ranges, so no remainder by zero is evaluated.

For i≥2, successful j values are i,2i,…,(i−1)i. The inner work is i∑m=1.. i−1 m=i^2(i−1)/2=Θ(i^3).

Summing over i gives Θ(n^4); condition checks add only Θ(n^3).

Neither n^5 nor n^7 is tight for this loop, though both are valid Big-O upper bounds.

This assumes mathematical integers and constant-cost output; literal signed C++ overflow in i*i is undefined, and executable runtime analysis requires valid syntax in a specified language.

Redeclaring parameter n and using return @;3 are syntax errors; neither establishes logarithmic complexity.

Design and Analysis of Algorithms module

DAA08 | Peak storage and recursion stacks

Space complexity measures peak simultaneously live storage, not the sum of every allocation ever made.

Count input, auxiliary structures and stack according to the requested convention. n active recursion levels with constant-size frames need Θ(n) stack; depth log n with constant frames needs Θ(log n).

The number of total calls does not determine peak depth.

Slicing/copying arrays can add storage omitted by an index-based pseudocode analysis.

Tail-call elimination is language/runtime-dependent and cannot be assumed for ordinary C++/Java recursion.

Resource tradeoffs include caching subproblem results to save time while consuming memory.

Design and Analysis of Algorithms module

DAA09 | Recurrences: substitution and expansion

Include every recursive call, nonrecursive work and base case.

Expansion gives $T(n)=T(n-1)+\Theta(1)\rightarrow\Theta(n)$, and $T(n)=T(n/2)+\Theta(1)\rightarrow\Theta(\log n)$.

For $2T(n/2)+\Theta(n)$, logarithmically many levels each cost $\Theta(n)$, giving $\Theta(n \log n)$.

Substitution guesses a bound and proves it inductively with constants/base cases; a lower-order correction may close the induction.

Unequal/decrementing recurrences need separate analysis.

Design and Analysis of Algorithms module

DAA10 | Recursion trees and the Master theorem

For $T(n)=aT(n/b)+f(n)$, $a\geq1$, $b>1$, let $p=\log_b a$. Level i has a^i nodes of size n/b^i. Compare f with n^p: a polynomial gap below gives $\Theta(n^p)$; $f=\Theta(n^p)$ gives $\Theta(n^p \log n)$; a polynomial gap above gives $\Theta(f(n))$ if a $f(n/b)\leq c\, f(n)$ for fixed $c<1$.

The third case needs this regularity condition.

A log-factor boundary may need an extended theorem or explicit summation.

State base work and include leaves.

This version does not apply to $T(n)=T(n-1)+n$ or unequal-split quicksort.

Example

Worked microexample

- $4T(n/2)+n$ gives $\Theta(n^2)$;$2T(n/2)+n$ gives $\Theta(n \log n)$; $T(n/2)+n^2$ gives $\Theta(n^2)$. Each follows a different case. Also $4T(n/2)+n^2$ gives $\Theta(n^2 \log n)$, case 2.

Design and Analysis of Algorithms module

MIT6.006 Recitation3

DAA11 | Comparison sorting lower bounds

A comparison decision tree needs at least n! leaves for distinct-key orderings.

Height h allows at most 2^h leaves, so $h\geq\log 2(n!)=\Omega(n \log n)$; merge sort matches this worst-case bound.

Array indexing is not forbidden.

Counting/radix sorting exploits structured keys outside the comparison-only model.

Recurrences analyze particular algorithms; this decision tree bounds the entire comparison class.

Assess a lower-bound argument by its computational model and reasoning rather than by a technique label alone.

Design and Analysis of Algorithms module

Virginia Tech OpenDSA, Sorting Lower Bound

DAA12 | Choose a design from structural conditions

Divide and conquer solves smaller largely independent instances and combines results.

DP reuses a defined set of subproblem states.

Greedy commits to a feasible local choice only when a safety argument supports it.

Backtracking grows/retracts candidates when constraints fail; branch and bound also rejects candidates whose best possible objective cannot beat an incumbent.

“Breaks into subproblems,” “sequence of choices,” “incremental,” or “optimization” alone describes several strategies.

Exponential runtime is an outcome, not enough information to select a method.

Specify overlap, local-choice safety, revisability, exactness and usable bounds.

Ordinary selection sort repeatedly selects one minimum; it is not a divide-and-conquer algorithm.

Design and Analysis of Algorithms module

DAA13 | Binary search and simultaneous minimum/maximum

Binary search requires sorted random-access data.

Maintain inclusive low/high bounds, compute mid=low+(high−low)//2, and replace high with mid−1 or low with mid+1 according to the comparison.

Stop when found or low>high.

$T(n)=T(n/2)+\Theta(1)$: worst $\Theta(\log n)$, best $\Theta(1)$; iterative auxiliary space $\Theta(1)$, recursive $\Theta(\log n)$.

Finding both extrema by two unconditional comparisons per element costs $2n-2$ comparisons, regardless of order.

Pair elements first, compare within each pair, then compare the smaller with minimum and larger with maximum: even n uses $3n/2-2$ comparisons.

A balanced recursive version has $T(n)=2T(n/2)+\Theta(1)=\Theta(n)$.

An unconditional two-if scan does not have an $n-1$-comparison best case.

Design and Analysis of Algorithms module

DAA14 | Merge sort: invariant, trace and costs

Split by indices, recursively sort each half, then repeatedly copy the smaller remaining front element into a temporary result.

The merge invariant says the output prefix is sorted and contains the smallest elements already consumed.

Append the unexhausted suffix.

Equal keys must be taken from the left half first to preserve stability.

Ordinary array merge sort has $T(n)=2T(n/2)+\Theta(n)=\Theta(n \log n)$ across input orders, $\Theta(n)$ auxiliary buffer and $\Theta(\log n)$ recursion depth.

A written $+O(n)$ recurrence alone establishes an upper bound; the tight result also uses linear merge work.

Empty/singleton subarrays are base cases.

Example

Worked microexample

- [4,1,3,2] splits into[4,1] and[3,2], which sort to[1,4] and[2,3]. Merge selects 1,2,3 and appends 4. No assumption that the original array is already sorted is needed.

Design and Analysis of Algorithms module

DAA15 | Quicksort: partition variants and worst cases

Choose a pivot, partition, then sort the resulting smaller ranges.

In a two-pointer scheme, advance i while values are below pivot and j while above; when $i \le j$, swap and advance both.

Bound the scans and ensure progress on equal values.

Recurse on the resulting ranges.

Hoare-style partition boundaries need not put the pivot at its final position; Lomuto-style partition commonly does.

Do not mix their return conventions.

Balanced splits give $\Theta(n \log n)$; repeatedly highly unbalanced splits give $\Theta(n^2)$.

Random pivots give expected $\Theta(n \log n)$, not a deterministic guarantee.

Median-of-three does not eliminate adversarial cases; three-way partitioning helps repeated keys.

Standard quicksort is unstable, uses expected $\Theta(\log n)$ stack but worst $\Theta(n)$; recursing on the smaller side and iterating on the larger bounds stack by $O(\log n)$.

Design and Analysis of Algorithms module

DAA16 | Sorting choices and comparison costs

Insertion shifts larger sorted-prefix keys: stable with strict comparisons, $\Theta(n)$ best and $\Theta(n^2)$ worst.

Optimized bubble sort swaps adjacent inversions and stops after a no-swap pass: stable, $\Theta(n)$ best, $\Theta(n^2)$ worst.

Selection repeatedly finds a suffix minimum: $\Theta(n^2)$ comparisons in every order, normally unstable.

These use $\Theta(1)$ auxiliary array space.

Heap sort builds a max-heap in $\Theta(n)$, then extracts/restores in $O(\log n)$ per item: $O(n \log n)$ total, $\Theta(1)$ auxiliary space, ordinarily unstable.

It is not standard D&C. Merge sort offers stability/worst-case guarantees; quicksort's practical advantage depends on implementation/workload.

Design and Analysis of Algorithms module

DAA17 | Strassen multiplication and divide/combine tradeoffs

Ordinary n×n multiplication computes n products/sums per output entry: $\Theta(n^3)$.

Eight half-size block products still give $8T(n/2)+\Theta(n^2)=\Theta(n^3)$.

Strassen uses seven products plus block sums/differences:$7T(n/2)+\Theta(n^2)=\Theta(n^{\log_2 7})\approx\Theta(n^{2.807})$.

Zero-padding preserves the required product.

The smaller exponent trades extra additions/temporary storage and possible numerical issues against multiplication work.

Use an ordinary base case for small matrices and specify exact versus floating-point arithmetic.

Design and Analysis of Algorithms module

DAA18 | Maximum contiguous subarray: two valid strategies

A maximum subarray is contiguous; it is not an arbitrary subsequence.

Divide-and-conquer compares the best left, best right and best crossing result.

The crossing result is the best suffix of the left half plus best prefix of the right, found in $\Theta(n)$.

This yields $2T(n/2)+\Theta(n)=\Theta(n \log n)$.

Kadane's DP keeps the best sum ending at i: $end[i]=\max(A[i], end[i-1]+A[i])$; keep the largest end seen.

This is $\Theta(n)$ time and $\Theta(1)$ extra space.

For a required nonempty subarray initialize from A[0], not zero; all-negative input then returns its largest element.

Empty-allowed variants may return zero.

Both D&C and DP can solve the maximum-subarray problem; specify the algorithm before assigning its complexity.

Example

Worked microexample

- For[−2,3,−1,4,−5], ending sums are−2,3,2,6,1; optimum 6 comes from[3,−1,4]. For[−4,−2], a nonempty answer is−2.

University of Alberta CMPUT498/501 Lecture2

DAA19 | Greedy correctness, limits and counterexamples

A greedy algorithm repeatedly selects a locally attractive feasible option without later undoing it.

Optimal substructure alone does not prove this safe.

An exchange proof inserts the greedy choice into an optimum without worsening cost; recurse.

MST cut proofs use this exchange.

Advantages are often simple decisions and low overhead; limitations are irrevocable errors and problem-specific applicability.

A heuristic has no approximation guarantee merely because it is greedy.

For coins{1,3,4} and target 6, largest-first gives 4+1+1 while 3+3 uses fewer coins.

Activity selection instead has a safe earliest-finish rule for maximizing the number of nonoverlapping unweighted intervals; replacing the first chosen activity in an optimum with an earlier-finishing one leaves at least as much room.

Design and Analysis of Algorithms module

DAA20 | Fractional knapsack versus indivisible items

For positive weights/profits and divisible items, sort by decreasing value/weight, take complete items while they fit, then the fitting fraction of the next.

The exchange argument replaces lower-density selected weight by available higher-density weight; value cannot decrease.

Sorting costs O(n log n), followed by a linear scan.

Handle zero-weight beneficial items separately rather than divide by zero.

This proof depends on divisibility.

With capacity 10 and items(6,12),(5,8),(5,8), ratio-greedy chooses value 12; the two weight 5 items give 16 under 0/1 rules.

Specify whether the knapsack problem is fractional,0/1 or unbounded.

Feasibility alone does not guarantee optimality.

Example

Worked microexample

- Weights 15,10,18 and values 24,15,25 with capacity 20: take the weight 15 item and half the weight 10 item. Value 24+7.5=31.5; initial remaining capacity must be 20, not the number of items.

Design and Analysis of Algorithms module

DAA21 | Unit-time job sequencing with deadlines

Assume one machine, initially available jobs, unit durations, integer deadlines and profit only for on-time completion.

Process nonnegative-profit jobs by decreasing profit; place each in the latest free slot finishing by its deadline, or reject it.

Latest placement preserves earlier slots for tighter deadlines; feasible unit-job sets support profit-first exchange.

Sorting plus slot scans costs O(n log n+nD), with D capped at n. Equal-profit order is arbitrary; earlier deadline is optional.

Variable durations/release times invalidate this rule.

Example

Worked microexample

- Slots finishing 1…5 contain J 2,J 4,J 3,J 5,J 1. Profits 180+300+190+120+200=990; deadlines hold. J 6 cannot finish by 2. The five largest profits sum to 990, giving a matching upper bound.

Design and Analysis of Algorithms module

DAA22 | Optimal merge patterns and Huffman coding

Combine the two smallest weights, add their sum to total cost, and reinsert it.

A min-heap gives O(n log n) time/O(n) space.

Cost p+q counts moved records; worst-case comparisons are p+q−1 for nonempty lists, and actual comparisons may be fewer.

Huffman uses frequencies as weights and 0/1 branches to leaf symbols, making a prefix-free code.

Cost is∑frequency×depth.

An exchange makes the two least frequencies deepest siblings; contract them and induct.

Ties can change codes without changing optimal cost.

Header overhead can outweigh tiny-file savings.

Example

Worked microexample

- For weights 2,3,5,6: merge 2+3=5, then 5+5=10, then 10+6=16. Total weighted merge cost 31; the same tree construction minimizes weighted prefix-code length.

Design and Analysis of Algorithms module

DAA23 | MSTs: Prim, Kruskal and the cut proof

For a connected weighted undirected graph, an MST spans every vertex without cycles and minimizes total edge weight; it has V−1 edges.

Negative edges are allowed.

Equal weights may yield multiple MSTs; distinct weights suffice for uniqueness.

A tree is itself a graph.

Prim repeatedly adds the cheapest edge crossing from its current tree to an outside vertex.

Kruskal sorts edges globally, accepting one only when union-find says its endpoints are in different components.

The cut proof exchanges a light crossing edge into some optimum without raising its weight.

Matrix Prim costs $O(V^2)$; binary-heap Prim O(E log V), and Kruskal O(E log E).

Dense-graph comparisons depend on these implementations.

Disconnected input gives a minimum spanning forest.

Example

Worked microexample

- Triangle AB=1,BC=2,AC=4: Kruskal accepts AB and BC for cost 3. If every edge is 1, any two edges give an MST, so uniqueness fails.

Design and Analysis of Algorithms module

DAA24 | Dijkstra: relaxation and the nonnegative condition

Initialize dist[source]=0, others∞ and predecessors unset.

Repeatedly extract the unsettled vertex u of minimum distance; relax each edge by dist[v]=min(dist[v], dist[u]+w(u, v)), updating predecessors when improved.

Once the minimum is∞, remaining vertices are unreachable.

A destination may be finalized before stopping, not merely discovered.

Nonnegative edges ensure a later path through unsettled vertices cannot improve a finalized distance.

With s→a=2, s→b=5, b→a=−4, finalizing a at 2 misses the later distance 1.

Binary-heap/list implementation costs $O((V+E)\log V)$, matrix scanning $O(V^2)$.

An MST minimizes total connection cost; Dijkstra minimizes each source-path distance, a different objective.

Design and Analysis of Algorithms module

DAA25 | DP states, optimal substructure and proof

Define state meaning, choices, recurrence, bases and dependency order.

Prove replacing a subsolution by a better one improves the full solution; otherwise state information may be missing.

Show the recurrence covers all legal final choices and generates feasible candidates, then induct in dependency order.

Store choices for reconstruction.

Overlapping states motivate reuse; optimal substructure supports optimization.

These are textbook signatures, not requirements that every DP problem optimize or every state have multiple users.

DP also counts/decides.

Cost=states×work per state and can remain exponential.

Design and Analysis of Algorithms module

DAA26 | Memoization, tabulation and Fibonacci

Memoization recursively checks a cache before solving a state and stores its result.

Tabulation orders dependencies explicitly; chosen domains may exclude irrelevant states.

Cache keys include all relevant parameters; recursion/locality/sparsity affect implementation cost.

Naive Fibonacci recomputes F(n−2) across branches: call count is $\Theta(\varphi^n)$, with $O(2^n)$ a valid loose upper bound.

F(0)=0, F(1)=1 and F(i)=F(i−1)+F(i−2) yield $\Theta(n)$ unit-cost additions with memoization/tabulation.

Keeping only two recent values uses $\Theta(1)$ machine words; arbitrary-precision output needs growing bit storage.

This naive overlapping recursion is not the independent-subproblem ideal of D&C.

Design and Analysis of Algorithms module

DAA27 | Multistage/DAG paths and Bellman-Ford

For a DAG or multistage graph, compute cost-to-go in reverse topological order: F(t)=0 and F(u)=min over outgoing(u, v) of[w(u, v)+F(v)].

Missing routes have∞.

Record the minimizing next vertex.

Equivalently propagate source distances forward.

Complexity is Θ(V+E) with adjacency lists; negative edges are harmless without cycles.

Follow the recurrence's dependencies when choosing the evaluation direction.

For a general graph, Bellman-Ford relaxes all edges for up to V−1 passes and tests an additional pass for a reachable negative cycle: O(VE).

Thus "single-source shortest path" does not uniquely imply greedy; graph assumptions determine the algorithm.

Example

Worked microexample

- s→a=2,s→b=5,a→t=4,b→t=−1 gives F(a)=4,F(b)=−1 and F(s)=min(6,4)=4 via b.

Design and Analysis of Algorithms module

DAA28 | Floyd-Warshall, reachability and negative cycles

For all-pairs distances initialize D[i, i]=0, D[i, j] to the minimum direct edge weight, and missing edges to∞.

Preserve a negative self-loop by taking min(0, its weight).

For k outermost, update D[i, j]=min(D[i, j], D[i, k]+D[k, j]); before phase k, intermediates are restricted to earlier vertices.

Guard unreachable sums and numeric overflow.

Three loops give $\Theta(V^3)$ time and $\Theta(V^2)$ storage.

A negative diagonal afterward identifies a negative cycle; pairs able to traverse it have no finite minimum walk distance.

Warshall reachability replaces min/+ by OR/AND.

Floyd's cost is not a universal best-achievable bound for all APSP settings; sparse nonnegative graphs permit repeated Dijkstra and other models permit other methods.

Example

Worked microexample

- If D[a,b]=4,D[b,c]=2,D[a,c]=11, phase b reduces D[a,c] to 6. Later phases must retain that improvement rather than restoring 11.

Design and Analysis of Algorithms module

DAA29 | 0/1 knapsack DP and dominance

For integer capacity W, K[i, w] is maximum value using the first i items within capacity w. Initialize K[0, w]=0.

If weight_i>w, use K[i−1, w]; otherwise take max(K[i−1, w], value_i+K[i−1, w−weight_i]).

The previous-row reference ensures each item is used at most once.

Backtrack changed values to recover chosen items.

Time O(nW) is pseudo-polynomial because W takes only O(log W) binary input bits.

Storage can be O(W); update capacities downward for 0/1 selection.

Upward iteration allows reuse and solves an unbounded variant.

In sparse(weight, profit) states, discard an option dominated by another with no greater weight and no smaller profit, provided both have the same remaining items.

Example

Worked microexample

- Weights 2,3,4, values 1,2,5 and W=6 yield value 6 by choosing weights 2+4. A state(weight 5,profit 3) is dominated by(weight 4,profit 5).

Design and Analysis of Algorithms module

DAA30 | LCS: prefix states and reconstruction

A subsequence preserves order but need not be contiguous.

For prefixes of lengths i, j define L[i, j].

Empty-prefix rows/columns are 0.

If X[i−1]=Y[j−1], set L[i, j]=1+L[i−1, j−1]; otherwise use max(L[i−1, j], L[i, j−1]).

Keep prefix lengths i and j distinct from the zero-based character indices i−1 and j−1.

Fill rows/columns in increasing order.

Reconstruct by taking matching characters diagonally or following a neighbor with the same optimal value; ties may produce different LCSs.

Time Θ(mn), table storage Θ(mn); two rows reduce value-only storage to O(min(m, n)), while ordinary reconstruction needs retained decisions or another technique.

Example

Worked microexample

- X=ABC,Y=AC: LCS is AC, length 2. Contiguous-substring matching would require a different recurrence, resetting a mismatch to zero.

MIT6.006 Lecture16

DAA31 | Interval DP: matrix chains and optimal BSTs

For matrix A_i of size p[i−1]×p[i], M[i, i]=0 and M[i, j]=min over i≤k<j of(M[i, k]+M[k+1, j]+p[i−1]p[k]p[j]).

Fill by increasing interval length and store the split.

Three matrices 10×30,30×5,5×60 cost 4500 under(AB)C versus 27000 under A(BC).

Time O(n^3), space O(n^2); multiplication order stays fixed while parentheses change.

For an OBST use sorted-key success weights p_i and gap failure weights q_i.

Let W(i, j)=∑p_i.. p_j+∑q_(i−1).. q_j.

Under the comparison-count convention, C(i, i−1)=0 and C(i, j)=W(i, j)+min_r(C(i, r−1)+C(r+1, j)); store r. CLRS-style dummy-node depths instead initialize empty cost to q_(i−1); final values differ by the fixed total gap weight.

Basic time O(n^3), space O(n^2).

Design and Analysis of Algorithms module

Stony Brook University, Dynamic Programming

DAA32 | Reliability design under a budget

Assume independent device failures.

With m_i parallel copies of stage i, each reliability r_i, stage reliability is φ_i(m_i)=1−(1−r_i)^m_i.

Series stages multiply: total∏φ_i.

Choose integers m_i≥1 with∑m_i c_i≤B; extra copies improve reliability but consume budget.

Correlated failures invalidate the simple independence formula.

Let R[i, b] be best reliability for the first i stages within budget b. Base R[0, b]=1; maximize R[i−1, b−m c_i]φ_i(m) over feasible m≥1.

Empty candidate sets give 0.

Time is O(stages×budget×maximum copies), with O(budget) value storage; reconstruct retained copy counts.

Example

Worked microexample

- For costs 30,15,20, reliabilities 0.9,0.8,0.5 and budget 105, copy counts(1,2,2) cost 100 and yield 0.9×0.96×0.75=0.648. Three copies of the weak last stage with the others single cost 105 but yield only 0.63.

Design and Analysis of Algorithms module

DAA33 | Exact TSP with subset DP

TSP asks for a minimum-cost tour visiting every vertex once and returning to its start; a shortest path need not visit all vertices or return.

Fix start s. Let F(S, j) be minimum cost from s through exactly S, ending at j∈S. Base F({s}, s)=0.

Transition F(S, j)=min_i∈S−{j}[F(S−{j}, i)+w(i, j)].

Finish with min_j≠s[F(V, j)+w(j, s)].

Missing edges cost∞.

Store predecessor choices to reconstruct a tour.

Held-Karp uses O($V^2$2^V) time and O(V 2^V) space, improving factorial enumeration but remaining exponential.

Greedy nearest-neighbor is a heuristic without a general exactness guarantee; backtracking and branch and bound can also solve exact TSP.

Specify the problem variant and algorithm before choosing a solution method.

For Held-Karp, S contains the start s. Initialize F({s},s)=0 and impossible states to infinity; for larger sets, evaluate endpoints j other than s. These boundary cases prevent using a nonexistent predecessor state.

Design and Analysis of Algorithms module

DAA34 | BFS, DFS and graph representation

BFS marks a vertex when enqueued, explores neighbors with a FIFO queue and records predecessor/level.

Levels give shortest paths by edge count in unweighted/equal-weight graphs.

DFS marks on entry, recursively or iteratively explores an unvisited neighbor to completion, then resumes the parent.

Run from each unvisited vertex for a full disconnected traversal.

Adjacency lists give O(V+E) traversal time and O(V) auxiliary visited/queue/stack space; adjacency matrices give O(V^2) scanning.

State neighbor order before requesting one exact traversal sequence.

Reverse DFS finish order gives a topological order only for DAGs.

BFS/DFS being printed under a DP unit does not make ordinary traversal a DP optimization algorithm.

Design and Analysis of Algorithms module

DAA35 | Articulation points and biconnected components

An articulation vertex increases the number of connected components when removed.

In an undirected DFS, disc[u] is discovery time and low[u] is the smallest discovery number reachable from u's subtree using downward tree edges and at most one back edge.

Update low from children and nonparent back edges.

A root is articulation if it has at least two DFS children; a nonroot u is articulation if some child v has low[v]≥disc[u].

Such a child cannot reach an ancestor of u without u. A bridge has the stricter condition low[v]>disc[u].

An edge stack can output a vertex-biconnected block when low[v]≥disc[u], popping through edge(u, v).

With adjacency lists these checks cost O(V+E).

Distinguish vertex-biconnectivity from strong connectivity in directed graphs.

For a simple undirected graph initialize disc[u]=low[u]=++time on entry.

For an unvisited neighbor v, push tree edge(u,v), DFS(v), then low[u]=min(low[u],low[v]).

For a visited nonparent ancestor v, push the back edge once when disc[v]<disc[u] and use low[u]=min(low[u],disc[v]).

After a child returns with low[v]>=disc[u], pop through(u,v) to output one block.

Keep the separate root-child articulation rule and bridge test low[v]>disc[u].

Restart DFS at each unvisited vertex.

For multigraphs skip the parent edge by identity rather than skipping every edge to the parent.

Example

A small procedure check

- Two triangles A-B-C-A and C-D-E-C, DFS order A,B,C,D,E: disc=(1,2,3,4,5), low=(1,1,1,3,3). Only C is articulation; no edge is a bridge; stack yields blocks{A-B,B-C,C-A} and{C-D,D-E,E-C}. low[D]=disc[C] shows why articulation uses>= but bridge uses>.

Design and Analysis of Algorithms module

IARCS, Articulation Points

CMU15-451: Undirected DFS and Biconnected Components

DAA36 | Backtracking template and safe pruning

Represent a partial assignment x[1.. k].

For each allowed next choice: assign it, test whether an extension remains possible, recurse if promising, and undo the assignment on return.

Report complete feasible candidates; stop after one only if the task requests existence/a single answer.

Explicit constraints restrict each variable's domain; implicit constraints relate choices.

Correctness requires exhaustive consideration of every feasible extension and pruning only impossible branches.

Merely disliking a partial objective is not enough.

DFS recursion uses space proportional to search depth, while total search can remain exponential.

Memoizing equivalent states or stronger constraints can reduce work, but branch count depends on actual choices/tests.

Design and Analysis of Algorithms module

DAA37 | N-queens: row, column and diagonal tests

Use x[r]=column for one queen per row.

Reject against earlier row i when x[r]=x[i] or $|x[r]-x[i]|=|r-i|$; distinct rows are built into the representation.

Recurse after legal placement and remove it on return.

Column and r−column/r+column sets permit constant-time conflict tests.

Distinct columns leave at most n! complete permutations plus partial-node work; pruning does not guarantee polynomial time.

Placement/conflict storage is O(n).

Boards 2 and 3 have no solutions; state empty-board conventions.

Example

Worked microexample

- For n=4, column vector(2,4,1,3) is valid. Rows 1 and 2 with columns 2 and 3 would share a diagonal because both differences are 1, so that partial branch is rejected.

Design and Analysis of Algorithms module

DAA38 | Subset sum: include/exclude and admissible bounds

Include/exclude each item while tracking current sum s and remaining sum R. For nonnegative weights prune s>target or s+R<target.

On equality report the subset, considering zero-weight extensions if all subsets are needed.

Sorting positive weights can strengthen tests but cannot force inclusion of the next item.

With negative values, s>target may recover later; use valid attainable-range bounds.

There are up to 2^n subsets and O(n) recursion depth.

Define whether duplicate-valued items count by index.

Example

Worked microexample

- For{3,4,5,6}, target 13, subset{3,4,6} succeeds. At s=9 with remaining total 3, s+R=12<13 proves failure. At s=9 with remaining 4, equality can still be reached.

Design and Analysis of Algorithms module

DAA39 | Graph coloring: feasibility versus minimum colors

For a fixed m, assign colors 1…m to vertices, rejecting a color used by an already colored neighbor.

Backtrack when all colors fail.

At most m^V complete assignments exist, with additional checking cost; a simple O(V)-per-test implementation gives an exponential bound.

A self-loop makes proper coloring impossible.

The decision question asks whether m colors suffice; optimization asks for the chromatic number.

Trying increasing m with a complete decision search finds the minimum.

Greedy coloring chooses a color by vertex order and may use too many; it is not a general optimum proof.

An odd cycle needs 3 colors; an edge-containing bipartite graph needs 2, and an edgeless nonempty graph needs 1.

Design and Analysis of Algorithms module

DAA40 | Hamiltonian cycles and permutation search

Fix a start vertex to avoid rotational duplicates.

Extend a path with an unused vertex adjacent to its current end.

At V vertices require an edge back to the start; otherwise backtrack.

A Hamiltonian cycle visits vertices once, while an Euler cycle concerns visiting edges.

Fixing the start leaves up to(V−1)! candidate permutations, with polynomial checking overhead.

For a simple undirected graph with at least 3 vertices, a degree 1 vertex or articulation point rules out a Hamiltonian cycle, but their absence does not prove one exists.

TSP optimizes a weighted Hamiltonian tour; the decision existence search alone does not minimize its cost.

Reversed undirected tours may be equivalent if enumerating unique cycles.

Design and Analysis of Algorithms module

DAA41 | Branch and bound: node states and bound direction

A live node is generated and still eligible for expansion; the E-node is currently expanded; a dead node is finished or pruned.

FIFO selects live nodes by queue order, LIFO by stack order, and least-cost/best-bound uses a priority queue.

Selection changes work/storage; valid bounds preserve exactness.

For minimization, an incumbent feasible solution supplies an upper bound U; a partial node supplies a lower bound L on every completion.

Prune if L≥U when only one optimum/value is needed.

For maximization, feasible profit is a lower bound and an optimistic completion estimate is an upper bound; prune if upper≤incumbent.

Keep equality branches when all optimal solutions must be enumerated.

Design and Analysis of Algorithms module

DAA42 | Knapsack branch and bound

Sort positive-weight/value items by decreasing density and branch include/exclude.

Reject overweight nodes.

To bound a node’s possible profit, fill its remaining capacity fractionally from remaining items; this relaxation can only improve on 0/1 feasibility, so its value is an upper bound.

A whole-item feasible completion updates the incumbent; the fractional result itself is not a legal 0/1 answer.

With capacity 10 and items(6,12),(5,8),(5,8), the root fractional bound is 18.4: take the first and 4/5 of the second.

The first-item-in branch has whole-item value 12; excluding it permits the two other items, value 16.

Continue until every live upper bound is≤16 to certify optimality.

Avoid downward floating-point rounding that could invalidate the bound; worst-case search remains exponential.

Design and Analysis of Algorithms module

DAA43 | TSP branch and bound by matrix reduction

For directed TSP, set diagonal/forbidden entries to∞.

Subtract each finite row minimum, then each finite column minimum, accumulating reductions.

Every tour uses one entry per row/column, so their sum is a lower bound; the relaxed assignment may contain disjoint subtours.

When including edge i→j, remove its used row/column, prevent premature subtours, reduce again and add the selected reduced-edge cost and new reductions to the parent bound.

Compare bounds with the best complete tour and expand a promising live node.

Never treat a reduced matrix's zeros as proof of a complete valid tour.

Branch on an undecided edge i->j. The include child deletes row i/column j and adds its selected reduced cost plus new reductions to the parent bound.

The exclude child sets that entry to infinity and adds only new reductions.

Keep forced in/out-degree at most one.

Track each forced path from head h to tail t and forbid t->h until the path contains every vertex; this blocks premature cycles.

A required residual row/column with no finite choice is infeasible and pruned.

Accept only one complete cycle through every vertex, not a set of disjoint cycles.

Example

A small procedure check

- For four cities, cheap forced edges1->2,2->1 and3->4,4->3 satisfy one incoming/outgoing edge per city but are two disjoint2-cycles, not a tour. Once forced path1->2 is formed, forbid2->1 while fewer than4 vertices are in the path. Preserve the existing3-city reduction example: bound25 equals tour cost25.

Design and Analysis of Algorithms module

BYU CS312: TSP Branch and Bound

DAA44 | P, NP, reductions and encoding size

P contains decision problems solvable deterministically in polynomial input-bit time.

NP contains decision problems with polynomial-length certificates verifiable in polynomial time, equivalently solvable by a nondeterministic polynomial-time machine.

NP does not mean non-polynomial; P⊆NP.

Whether P=NP remains open.

A polynomial many-one reduction A≤p B transforms each A-instance into a B-instance preserving yes/no. To prove B NP-hard, reduce a known hard A to B, not the reverse.

NP-complete means NP-hard and in NP; optimization TSP is normally called NP-hard, while its bounded-tour decision form is NP-complete.

A valid proposed tour and its cost are easy to check, but certifying global optimality is another task.

Pseudo-polynomial O(nW) knapsack is not polynomial in the bit length of W.

University of Waterloo CS365, P versus NP

UC Davis ECS120, NP-Completeness and Reductions

DAA45 | Check algorithm specifications and cost assumptions

For p=10,n=5, if(n>p) cout<<(n+p) skips output and terminates.

Skipping this output does not by itself violate feasibility, finiteness or control flow.

Efficiency or a missing required output needs a specification.

A precise degree for 3n?+7n+12 requires the missing exponent; fixed nonnegative integer exponents still give polynomials.

Use valid declarations rather than ini/inl or a redeclared n; exact counts also need a primitive-cost model.

Justify the result from the specification, valid code and cost assumptions.

Design and Analysis of Algorithms module

Introduction to Artificial Intelligence

Complete Study Notes

AI01 | AI capabilities and scope

Artificial intelligence studies systems that perform tasks requiring intelligence: reasoning, learning, planning, perception and language.

Applied AI builds useful solutions; cognitive AI uses computational models to investigate biological intelligence.

A model should be compared with human behavior or experimental observations before claiming it explains human thought.

Narrow or weak AI addresses bounded tasks, such as face recognition, spam filtering or a voice assistant.

General AI denotes broad competence transferable across domains; strong AI also has a philosophical meaning involving actual understanding.

Do not equate a humanoid body, autonomous vehicle, or many features with general intelligence.

Exam tip

Exam trap

- Data analysis overlaps AI when it uses learning or inference. It is not categorically excluded from AI.

Introduction to Artificial Intelligence module

AI02 | Four approaches and the Turing test

Thinking humanly models human mental processes; thinking rationally derives conclusions by valid logic.

Acting humanly seeks humanlike behavior; acting rationally selects actions expected to achieve objectives.

Thought processes and observed behavior are different evaluation targets.

The conversational Turing test asks whether a judge can reliably distinguish a machine from a human through their dialogue.

It requires language use, knowledge, reasoning and often learning.

It tests an operational behavior criterion, not proof of consciousness.

Cognitive models can be tested against response times, errors and psychological experiments.

Example

Choose the approach

- Finding a best action sequence to a goal fits acting rationally. Explaining human errors with a computational model fits thinking humanly.

Introduction to Artificial Intelligence module

AI03 | Agents, percepts and action

An agent perceives an environment through sensors and acts through actuators or effectors.

Cameras sense; motors move.

Software sensors may be incoming messages and software actions may be API requests.

Humans are agents but are not artificial agents.

A percept is one observation; a percept sequence is the entire observation history.

An agent function maps percept sequences to actions: f:P*→A. An agent program implements that mapping on an architecture; agent = architecture + program.

Environment means the setting in which it senses and acts, not merely the search space.

Autonomy means relying substantially on experience rather than only a designer's prior instructions.

Introduction to Artificial Intelligence module

AI04 | Rationality and performance

A performance measure defines success externally: safety, useful deliveries, elapsed time, or cost.

Rationality depends on that measure, prior knowledge, percept history, and available actions.

A rational agent selects the action with the best expected performance given its information.

Omniscience would mean knowing actual action outcomes in advance.

Rationality does not require omniscience or guaranteed success.

Even complete outcome knowledge cannot create a successful action when every available action fails.

Information gathering and exploration may therefore be rational actions.

Example

Uncertain route

- A route with expected travel time 20 minutes can be the rational choice even if an unforeseen closure makes it take 40 minutes.

Introduction to Artificial Intelligence module

AI05 | Specify the task with PEAS

PEAS means Performance measure, Environment, Actuators, Sensors.

Specify all four before designing an agent.

For a taxi: performance includes safety and travel time; environment includes roads, traffic and passengers; actuators include steering, brakes and displays; sensors include cameras, positioning and speed readings.

A sensor is not an actuator just because it influences a decision.

The performance measure is not the percept sequence: one defines success, the other supplies evidence.

Separate what the agent can observe from what a designer knows about the world.

Introduction to Artificial Intelligence module

AI06 | Simple and model-based reflex agents

A simple reflex agent matches condition-action rules against the current percept only: if dirty, clean.

It has no persistent world-state memory.

This is adequate when the current observation contains everything needed for its rule, but partial observability can make the same percept require different actions.

A model-based reflex agent maintains internal state using the previous state, previous action, current percept, and models of world change and action effects.

It then applies rules to that state.

Both kinds can use condition-action rules; memory and a world model distinguish them.

Example

Remember a hidden room

- Remembering that a room was cleaned after it leaves sensor range requires internal state. Merely detecting current dirt does not.

Introduction to Artificial Intelligence module

AI07 | Goals, utility and model updates

A goal-based agent considers future action sequences that reach a desired state.

A utility-based agent compares degrees of desirability and trade-offs, choosing maximum expected utility: $EU(a)=\Sigma s\ P(s|a,evidence)U(s)$.

Reaching the destination is a goal; balancing arrival time, safety and cost requires preferences among outcomes.

Updating an internal state after selecting an action predicts the action's effect; it does not itself modify the external environment.

A stored variable named goal is insufficient to prove a goal-based policy if BEST_ACTION never uses it.

Model-based, goal-based, utility-based and learning capabilities can coexist.

Introduction to Artificial Intelligence module

AI08 | Learning-agent components

The performance element selects actions.

The learning element changes the agent to improve future performance.

The critic evaluates behavior against a performance standard and supplies feedback.

The problem generator suggests exploratory actions likely to yield useful experience.

Learning is improvement on task T, measured by P, through experience E. For spam filtering, T is classifying email, E is past labeled messages, and P might be recall at a required precision.

Exploration trades immediate reward for information; exploitation uses the best action currently known.

Learning can improve reflex, goal-based or utility-based agents.

Introduction to Artificial Intelligence module

AI09 | Environment dimensions

Fully observable means sensors reveal relevant state; partially observable means relevant information is hidden or noisy.

Deterministic means state and action determine the next state; stochastic means outcomes involve randomness.

Strategic uncertainty about an opponent is distinct from dice-like randomness.

Episodic decisions are independent; sequential decisions affect later situations.

Static worlds do not change while deliberating; dynamic ones do.

A clock affecting score makes an otherwise static game semidynamic.

Discrete quantities have distinct alternatives; continuous quantities vary along real-valued ranges.

Known versus unknown concerns knowledge of the transition model, not observability.

Example

Classify examples

- Chess is sequential, discrete and deterministic given both players' chosen moves; untimed chess is static, timed chess semidynamic. A fixed crossword is static. Driving is partially observable, sequential, dynamic and partly continuous.

Introduction to Artificial Intelligence module

AI10 | Single-state, multiple-state and contingency problems

With a known deterministic model and full observation, a single current state can be used.

Without observation but with known deterministic transitions, a sensorless or multiple-state problem tracks a belief state: a set of possible states.

Nondeterminism or partial observation may require a contingency plan with branches based on later percepts.

Exploration or online search is required when the state space or transition model is unknown; observation alone does not identify this category.

These classifications can overlap.

A pairing based only on "partially observable, deterministic" cannot distinguish a known sensorless problem from an unknown exploration problem.

Russell and Norvig, problem-solving slides

AI11 | Formulate before searching

First formulate the goal; then formulate the problem by specifying initial state, legal actions, transition model, goal test, and action costs.

Search selects a sequence; execution applies it.

A state abstracts away irrelevant world details.

In route finding, states are cities, actions are roads, successors are neighboring cities, and path cost is accumulated distance or time.

A solution is a legal path ending at a goal; an optimal solution has minimum cost.

Automatic programming or program synthesis searches for a program satisfying a specification; conditional plans and loops require richer representations than a fixed action list.

Introduction to Artificial Intelligence module

AI12 | Frontiers and evaluation criteria

A search node records a state plus parent, action, depth and path cost.

Expanding a node generates its successors.

OPEN or frontier stores discovered candidates awaiting expansion; CLOSED records expanded states.

Reaching the same state through different paths does not imply the same node or cost.

Completeness means finding a solution if one exists under stated assumptions; optimality means returning the best solution.

Time counts work, often generated nodes; space counts simultaneous storage.

Let b be branching factor, d shallowest goal depth, m maximum search depth, and l a depth limit.

Graph-search bounds in vertices V and edges E differ from implicit-tree bounds.

Introduction to Artificial Intelligence module

AI13 | Breadth-first search

BFS removes the oldest frontier entry using a FIFO queue.

Start with the root, remove the head, test it, append unseen successors at the tail, and repeat.

It explores depths 0,1,2,... and is complete for finite b and a finite-depth goal.

It minimizes number of edges, so minimum cost requires equal positive step costs.

The tree contains 1+b+...+b^d nodes through depth d, giving O(b^d) under generation-time goal testing.

Expanding depth-d nodes before testing can generate depth d+1 and yields O(b^(d+1)); specify the convention.

Explicit finite graph traversal with visited-state tracking takes O(V+E) time and O(V) space.

Example

Trace a queue

- If S has children A,B and A has C,D, BFS removal order starts S,A,B,C,D, assuming that child order.

Introduction to Artificial Intelligence module

AI14 | Depth-first search

DFS follows the deepest pending branch using a LIFO stack or recursion.

FILO means the same order as LIFO.

With the same children as BFS, a left-first DFS visits S,A,C before B; push rightmost children first to pop the leftmost first.

Implicit-tree worst-case time is O(b^m), and space is O(bm) when pending siblings are stored, or O(m) for path-only recursive generation.

DFS is not generally optimal and can fail on infinite paths or cycles.

On a finite graph, visited-state tracking makes traversal complete but needs O(V) storage.

Do not confuse the O(b^m) worst-case time bound with the DFS space bounds above.

Introduction to Artificial Intelligence module

AI15 | Uniform-cost search and Dijkstra

UCS uses a priority queue ordered by actual accumulated path cost g(n).

Remove a minimum-g node; accept a goal when removed, not when first generated.

Relax each edge: if a cheaper path is found, update its frontier cost and parent.

With finite branching and each step at least ε>0, UCS is complete and optimal.

Finite graphs permit nonnegative edges with proper duplicate handling.

Dijkstra uses the same minimum-distance principle, often computing all reachable distances; UCS commonly stops at a goal and generates states on demand.

Queue initialization is an implementation choice, not a fundamental distinction.

Negative edges invalidate the usual finalized-distance argument.

Introduction to Artificial Intelligence module

AI16 | Depth limits, iterative deepening and two directions

DLS runs DFS to depth l and reports cutoff separately from failure.

It misses goals deeper than l.

Iterative deepening runs DLS with limits 0,1,2,...; for finite b it finds the shallowest solution with O(b^d) time and O(bd) frontier space, optimal for equal step costs.

Repeated shallow work is small compared with the deepest level.

Bidirectional search grows from start and goal until frontiers connect.

With suitable reverse actions, meeting tests and equal-cost BFS, time and space may fall to O(b^(d/2)).

Weighted variants require a sound stopping bound; the first intersection need not be cheapest.

IDA* deepens an f=g+h threshold rather than only a depth limit.

Introduction to Artificial Intelligence module

AI17 | Heuristics and greedy best-first search

A heuristic h(n) estimates remaining cost or evaluates promise using problem-specific knowledge.

It can reduce explored work, but computing it also costs time.

Uninformed algorithms use only the problem definition and known costs; UCS remains uninformed despite using g.

Best-first is a family that orders OPEN by an evaluation function.

Greedy best-first uses f=h and removes the smallest estimate, ignoring the cost already paid.

It can choose an expensive route, loop without duplicate checks, or follow an infinite misleading branch.

On a finite graph with full duplicate handling it is complete, but not generally optimal.

Its implicit finite-tree worst case can be O(b^m).

Introduction to Artificial Intelligence module

AI18 | Admissibility, consistency and A*

A* orders by f(n)=g(n)+h(n).

Admissibility means 0≤h(n)≤h*(n), the true cheapest remaining cost.

Consistency means h(n)≤c(n,n′)+h(n′) on every edge, with h(goal)=0; it implies admissibility on states with goal paths.

Consistency makes f nondecreasing along a path.

Admissibility suffices for optimal A* tree search; graph search that never reopens CLOSED states normally requires consistency.

With an admissible inconsistent heuristic, reopen states when g improves.

Test a goal when removed at minimum f. Completeness also needs finite branching and positive lower-bounded costs in infinite spaces.

Worst-case time and memory can be exponential; no heuristic universally guarantees speed.

Introduction to Artificial Intelligence module

UC Berkeley CS188, Search Note 1

AI19 | Work a heuristic-search example

For S→A cost 2, S→B cost 1, A→G cost 2, B→G cost 10, set h(A)=2, h(B)=1, h(G)=0.

After expanding S, greedy chooses B because 1<2 and can then choose G, returning cost 11.

A* compares A:f=2+2=4 and B:f=1+1=2; after B, its G entry costs 11.

It next expands A and replaces G with cost 4, which is removed and accepted.

For a sliding puzzle with unit moves, misplaced-tile count and summed Manhattan distances excluding the blank are lower bounds.

Straight-line distance is a lower bound on road distance; it is not automatically a lower bound on travel time without conversion using a valid speed bound.

Introduction to Artificial Intelligence module

AI20 | Local optimization and evolutionary search

Local search keeps a complete candidate configuration and improves its objective; the path may be irrelevant.

Hill climbing chooses better neighbors.

First-improvement accepts the first improvement; steepest-ascent examines all neighbors; stochastic variants choose among improvements.

Local maxima, plateaus and ridges can stop progress.

Random restarts or limited sideways steps can help without guaranteeing success.

A genetic algorithm maintains a population, evaluates fitness, selects parents, applies crossover and mutation, and repeats replacement.

It searches without guaranteeing global optimality.

Means-ends analysis selects a difference between current and goal states, chooses an operator reducing it, and recursively solves its preconditions.

Neither method is simply A* running backward.

Introduction to Artificial Intelligence module

AI21 | Constraint satisfaction

A CSP specifies variables X, domains D, and constraints C. A solution assigns every variable a value and satisfies every constraint.

Complete means all variables assigned; consistent means no violated constraint.

A complete assignment can still be inconsistent.

Backtracking assigns a variable, tries a legal value, and undoes the choice on failure.

Choose the variable with fewest remaining values; break ties by degree; try values that rule out fewest neighbor choices.

Forward checking removes incompatible neighbor values.

Arc consistency removes a value when a binary constraint offers it no support in the other domain.

CSP is a problem framework using search and propagation, not a universal fourth uninformed traversal.

Introduction to Artificial Intelligence module

AI22 | Games and minimax

Adversarial search models agents with competing objectives.

A game specifies states, player to move, legal actions, successor function, terminal test and utility.

Perfect information means previous moves and current state are visible; it does not reveal future choices or imply absence of chance.

Zero-sum utility makes one player's gain the other's loss.

Minimax assigns terminal utilities, takes minima at MIN nodes and maxima at MAX nodes, then chooses a maximum-valued root action.

It gives the best guaranteed outcome against optimal opposition within the modeled tree.

For stochastic outcomes add chance nodes weighted by probabilities.

Example

Back up leaves

- If two MIN children have leaves [3,5] and [2,9], their values are 3 and 2. MAX chooses the first branch with value 3.

Introduction to Artificial Intelligence module

AI23 | Alpha-beta and game evaluation

Alpha is MAX's best lower bound; beta is MIN's best upper bound.

Pass bounds down, update alpha at MAX and beta at MIN, and prune when alpha≥beta.

Pruning preserves the minimax value for the same searched tree.

Good move ordering increases pruning: worst-case O(b^m), ideal ordering about O(b^(m/2)).

At a depth cutoff use an evaluation function estimating position desirability.

Positive favors the designated MAX player, negative MIN; the scale is a convention, with +1/0/−1 often win/draw/loss.

Material counts weighted pieces; mobility counts legal options; territory scores controlled regions.

These are different features, often combined.

Heuristics improve limited computation, not guarantee victory.

Introduction to Artificial Intelligence module

AI24 | Represent knowledge for inference

Knowledge representation encodes facts, objects, relations, events and rules so a machine can use them.

Syntax defines legal expressions; semantics defines their meaning.

A knowledge base stores assertions; an inference system derives consequences.

Declarative knowledge says what is true; procedural knowledge specifies how to act.

A semantic network connects concept nodes by labeled relations such as is-a. Frames describe stereotyped entities through slots, values and defaults.

Production systems use IF conditions THEN conclusions or actions, matching rules against working memory and selecting among applicable rules.

A graph suits relationships such as frequently bought together; frames suit attributes.

No representation is always best for recommendation without task requirements.

Introduction to Artificial Intelligence module

AI25 | Propositional logic and models

Propositional atoms represent whole true-or-false claims. ¬P negates; P∧Q requires both; P∨Q requires at least one; P→Q is false only when P is true and Q false; P↔Q requires matching truth values.

Thus P→Q ≡ ¬P∨Q. De Morgan gives ¬(P∧Q) ≡ ¬P∨¬Q.

A model is an interpretation making a sentence true.

Valid means true in every interpretation; satisfiable means true in at least one; unsatisfiable means true in none.

Invalid means not valid, and may still be satisfiable.

P∨¬P is valid; P is satisfiable but not valid; P∧¬P is unsatisfiable.

Entailment KB⊨α means every model of KB also satisfies α.

Introduction to Artificial Intelligence module

AI26 | First-order logic and quantifier scope

First-order logic adds objects, predicates, functions and quantified object variables.

Human(a) is a property; Likes(a,b) a relation; father(a) a function term.

Propositional logic has no object quantifiers, but its truth still depends on the assigned atom values.

Use ∀x(Human(x)→Mortal(x)) for all humans, and ∃x(Student(x)∧Tall(x)) for at least one tall student.

These are useful translation patterns, not rules forbidding other connectives. ∀x∃y Likes(x,y) allows a different y for each x; ∃y∀x Likes(x,y) requires one shared y. Negation swaps quantifiers: ¬∀x P(x) ≡ ∃x¬P(x).

Only occurrences within the correct quantifier's scope are bound.

Example

Someone at the door

- ∃x(StandingAt(x,door)∧Tall(x)) requires one individual satisfying both predicates. Using implication instead allows an unrelated individual to satisfy it vacuously.

Introduction to Artificial Intelligence module

AI27 | Inference, unification and chaining

Modus ponens derives Q from P and P→Q; modus tollens derives ¬P from P→Q and ¬Q. Affirming Q does not prove P. Universal instantiation substitutes an object for a universally quantified variable.

Unification finds a consistent substitution: Knows(x,Ada) and Knows(Ben,y) unify with x=Ben,y=Ada.

Distinct constants cannot be unified by renaming them.

Resolution combines clauses P∨Q and ¬P∨R into Q∨R. To prove α, add ¬α to KB in clause form and derive contradiction.

Forward chaining starts with facts and fires rules; backward chaining starts with a query and proves required premises.

Sound inference preserves truth; completeness means finding all entailed conclusions in its specified setting.

Introduction to Artificial Intelligence module

AI28 | Deduction, induction and abduction

Deduction applies truth-preserving rules: all people are mortal; Ada is a person; therefore Ada is mortal.

Induction generalizes observations to a broader rule and can be overturned by new examples.

Abduction proposes a plausible explanation: wet ground suggests rain, although sprinklers could explain it.

Analogy transfers a pattern between similar situations.

Classical logical entailment is monotonic: adding premises preserves previous consequences.

Default reasoning can be nonmonotonic: infer a bird usually flies, then withdraw this when told it is a penguin.

Common-sense defaults are useful with incomplete knowledge, but are not guaranteed truths.

Blind search is a search strategy, not a definition of reasoning.

Introduction to Artificial Intelligence module

AI29 | Probability and uncertain models

Probability describes uncertainty about events.

$P(\neg A)=1-P(A)$; $P(A|B)=P(A\cap B)/P(B)$, for $P(B)>0$.

Bayes gives $P(H|E)=P(E|H)P(H)/P(E)$: posterior equals likelihood times prior divided by evidence probability.

For mutually exclusive exhaustive hypotheses, $P(E)=\Sigma_i P(E|Hi)P(Hi)$.

A Bayesian network is a directed acyclic graph with conditional probability tables; its joint distribution factors as the product of each node's probability given its parents.

A Markov model assumes the next state depends on the current state rather than the entire past.

These represent uncertainty; fuzzy membership represents a different issue.

Example

Update a hypothesis

- If $P(H)=0.1$, $P(E|H)=0.8$ and $P(E|\neg H)=0.2$, then $P(E)=0.26$ and $P(H|E)=0.08/0.26\approx 0.308$. Likelihood 0.8 is not the posterior.

Introduction to Artificial Intelligence module

AI30 | Fuzzy logic and vague categories

Fuzzy sets assign graded membership $\mu(x)$ between 0 and 1 to concepts such as warm or tall.

A temperature can belong to warm with degree 0.7.

This is not a 70% probability that an otherwise crisp warm proposition is true.

Boolean logic uses only true and false; probability describes uncertainty about events.

A fuzzy controller fuzzifies measurements, applies graded IF-THEN rules, combines outputs and often defuzzifies to a numerical action.

Fuzzy control suits linguistic rules about degrees; reinforcement learning suits reward-based adaptation.

Merely saying an environment is complex or uncertain does not uniquely select either method.

James Brule, Fuzzy Systems tutorial, CMU archive

AI31 | Supervised learning and prediction

Supervised learning fits a mapping from labeled input-target examples.

Classification predicts discrete labels, such as institution generations or spam/nonspam; regression predicts numeric quantities, such as prices.

Predictive analytics uses historical patterns to forecast outcomes and can use either.

Specify the target before choosing a method.

A decision tree tests an attribute at each internal node, follows the corresponding branch and predicts at a leaf.

ID3 chooses high information gain: $H(S)=-\Sigma p_i \log_2 p_i$; gain is parent entropy minus weighted child entropy.

Gini impurity is $1-\Sigma p_i^2$, zero for a pure class.

Pruning limits overfitting.

Neural networks, support-vector machines and trees can all be supervised.

Introduction to Artificial Intelligence module

AI32 | Unsupervised, semi-supervised and active learning

Unsupervised learning seeks structure without target labels: clustering groups similar examples; association finds co-occurrence; dimensionality reduction compresses features.

Good clusters have high within-cluster similarity and low between-cluster similarity under an appropriate distance.

K-means alternates assignment to nearest centroids and centroid updates.

Semi-supervised learning combines some labels with additional unlabeled data.

Active learning chooses informative cases for an oracle to label; it still uses labels. “Exploratory learning” is not an exclusive standard synonym: it can mean unsupervised exploration or active data acquisition, so define the feedback.

A task named summarization or time-series analysis is not automatically supervised; its training targets determine that.

Introduction to Artificial Intelligence module

Burr Settles, From Theories to Queries

AI33 | Reinforcement learning

An RL agent observes state information, takes an action, receives reward and a new observation, and improves its policy to maximize expected cumulative reward.

Rewards evaluate outcomes rather than supply the correct action label.

Rewards may be delayed; not every correct-looking action receives immediate reward.

An illustrative discounted return is $G_t=r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\ldots$, with $0\le\gamma<1$.

The policy maps states or histories to action choices.

Exploration gathers information; exploitation pursues currently estimated reward.

RL is part of machine learning, which is part of AI. “No labels” alone fits both unsupervised learning and RL; reward-driven interaction distinguishes RL.

Introduction to Artificial Intelligence module

AI34 | Neural networks and deep learning

A neuron computes $z=\Sigma iwixi+b$ and output $a=\varphi(z)$.

For $x=(2,1)$, $w=(0.5,-1)$, $b=0.2$, $z=0.2$.

Linear activation returns z; sigmoid returns $1/(1+e^{(-z)})$; tanh is bipolar; ReLU returns $\max(0,z)$.

Nonlinearities let multiple layers represent nonlinear relationships.

Feedforward networks pass signals toward outputs without recurrent state loops; recurrent networks carry state across sequence steps.

Training adjusts weights, often through gradients and backpropagation.

Backpropagation is not the same as a recurrent connection.

Deep learning uses multiple representation layers and can learn features from images, sound and text.

CNNs share filters across image locations.

ANN is a brain-inspired abstraction, not a literal brain simulation; $DL \subset ML \subset AI$.

Introduction to Artificial Intelligence module

AI35 | Generalization and validation

Overfitting means fitting training-specific details so performance drops on unseen data; underfitting means the model is too limited or poorly fitted even on training data.

A learning curve plots performance against training size or training progress.

Complexity control, regularization, pruning and more representative data can help.

Train data fits parameters; validation data selects models or settings; a held-out test set estimates final performance.

In k-fold validation, train k models, each on $k-1$ folds and evaluate on the remaining fold, then average scores.

Stratification approximately preserves label proportions.

Fit preprocessing within each training fold.

Time-ordered data needs splits respecting time; repeated tuning on the final test set leaks information.

Introduction to Artificial Intelligence module

scikit-learn, Cross-validation

AI36 | Measure the right errors

With a fixed positive class, TP means true positives, FP false positive predictions, FN missed positives, TN true negatives.

Accuracy=(TP+TN)/N; precision=TP/(TP+FP); recall=TP/(TP+FN); F1=2PR/(P+R).

State the averaging and zero-denominator convention when relevant.

For TP=30, FP=10, FN=20, TN=40, accuracy=70%, precision=75%, recall=60%, and F1≈66.7%.

Accuracy can hide failure on a rare class.

Choose recall when missed positives matter; choose precision when false alarms are costly.

Regression errors include MAE=mean|y−ŷ| and MSE=mean(y−ŷ)²; squared error penalizes large misses more strongly.

Introduction to Artificial Intelligence module

AI37 | Language processing tasks

NLP interprets and generates human language: translation, dialogue, summarization, question answering and text generation.

Speech recognition converts audio to text; speech synthesis generates speech.

They overlap language processing but solve different input/output tasks.

Tokenization divides text into processing units; tokens may be words, punctuation or subwords.

POS tagging assigns grammatical roles; named-entity recognition identifies people, organizations and locations.

Parsing identifies grammatical structure; semantics concerns meaning; discourse links sentences; pragmatics uses context and speaker intent.

Sentiment analysis predicts evaluative tone.

Ambiguity can occur at every level: “bank” has multiple senses, and “I saw the man with a telescope” has multiple attachments.

Introduction to Artificial Intelligence module

AI38 | Morphology, sounds and prosody

Morphology studies meaningful word parts: un-happi-ness contains a prefix, stem and suffix; cats contains cat and plural -s. It differs from tokenization, which identifies processing boundaries.

Phonetics studies physical speech sounds; phonology studies a language’s sound patterns and contrasts.

Prosody concerns stress, rhythm, timing, prominence and intonation across syllables and phrases.

A pitch rise can help signal a question; shifting stress can change emphasis.

These cues assist dialogue and speech processing.

Pragmatics concerns intended meaning in context, not the internal components of words.

Introduction to Artificial Intelligence module

Jurafsky and Martin, Speech and Language Processing, phonetics chapter

AI39 | Perception, vision and robotics

Computer vision interprets images or video: classification names image content, detection locates objects, segmentation labels pixels, and recognition identifies objects or people.

Occlusion, lighting change, noise, unfamiliar conditions and incomplete context make open-world perception difficult even when a face detector works well.

Robotics combines mechanical, electrical, control and computing systems to sense, plan and act physically.

Components include sensors, controller, power supply, actuators and an end effector.

Locomotion may be legged, wheeled or tracked, with trade-offs in terrain, stability and efficiency.

Software can act autonomously without being a robot.

Humanoid shape is not evidence of general intelligence.

Introduction to Artificial Intelligence module

AI40 | Expert systems and choosing methods

An expert system applies specialized knowledge to a domain such as equipment diagnosis.

The domain expert supplies expertise; the knowledge engineer elicits and encodes it; the inference engine reasons over a knowledge base and working facts.

The user interface handles interaction; an explanation facility presents why or how a conclusion was reached.

Rules offer inspectable domain reasoning but can be brittle and costly to maintain.

Learned models handle data-rich patterns but depend on representative data and validation.

Search suits explicit action alternatives; probabilistic models suit uncertain evidence.

Bias in data or evaluation can create unfair outcomes.

Evaluate errors across relevant groups, robustness, privacy and human oversight rather than assuming automation is always beneficial.

Introduction to Artificial Intelligence module

AI41 | Clarify search and logic questions

A* is forward best-first search combining actual past cost with estimated future cost; it does not inherently run a backward frontier.

Both A* and IDA* can have exponential worst cases and optimality under their conditions. “Minimum heuristic” and “estimated closest” describe the same greedy choice.

LIFO and FILO are synonyms.

Using information from the current state alone does not establish that a search uses a heuristic.

Ask whether remaining cost is estimated.

Both “invalid iff unsatisfiable” and “satisfiable implies valid” are false in general; distinguish validity from satisfiability.

Evaluate each validity claim separately, or identify all false statements when several are offered.

Introduction to Artificial Intelligence module

UC Berkeley CS188, Search Note 1

AI42 | Distinguish agent categories and components

The common teaching list contains five agent classes when learning agents are included, or four decision architectures before learning is added.

State the convention when counting agent types. “Target-based” can mean goal-based; “unity-based” is not utility-based.

A generic rule does not distinguish simple from model-based reflex without specifying memory.

A goal declaration unused by the action policy does not establish goal-based behavior. “Sensors and effectors” describes interaction, not omniscience or intelligence by itself.

Define the mechanical distinction when comparing effectors and actuators as acting components.

Introduction to Artificial Intelligence module

AI43 | Clarify overlapping learning and AI labels

ANNs, genetic search and reinforcement learning all improve candidate behavior or performance through experience in different ways.

Deep learning is an architecture family usable with supervised, unsupervised or reinforcement objectives.

Decision trees and deep models may both be supervised.

To distinguish learning regimes, specify labels, reward or unlabeled structure.

Face recognition is both an applied task and a narrow capability.

Broad machine-intelligence goals can include emulating human intelligence and building systems that learn.

Humanlike problem-solving alone does not uniquely select expert systems.

Python is widely used, but C++ and Java also implement AI; “commonly used” does not isolate one language.

McCarthy is associated with naming AI, not its sole invention.

Introduction to Artificial Intelligence module

AI44 | Interpret quantified statements and search assumptions

The formula ∀x∃y∃t¬F(x,y,t) means that for each person there is a person and time at which fooling fails.

By quantifier negation this is ¬∃x∀y∀t F(x,y,t), so nobody fools everybody all the time.

Forbidding every successful instance is a much stronger claim than this formula.

Preserve the quantifiers and negation when translating between a formula and a verbal statement.

Compare Dijkstra and uniform-cost search under stated implementations rather than treating queue initialization as an intrinsic difference.

For chess, specify the unit of decision: moves within a match are sequential; independently reset matches can be episodes.

Select a recommendation design from explicit requirements; graph relationships can support a choice without excluding other designs.

Introduction to Artificial Intelligence module

AI-S01 | Programmed computation and learned patterns

A conventional computer manipulates digital information according to a program or sequence of instructions. Programs can implement storage, retrieval and learning operations. Neural networks can learn pattern recognition from examples, so executing instructions and learning patterns are not mutually exclusive descriptions. This provides no universal claim that every machine is better or worse than every human at completing missing information.

AI algorithms are not exclusive to one programming language. Rumale is a Ruby machine-learning library with classification and clustering implementations, demonstrating that Ruby can implement learning algorithms as well as the general-purpose languages already discussed. Language choice and learning regime are different decisions.

NIST glossary - Computer

Introduction to Artificial Intelligence module · page 71

Rumale - official project README

AI-S02 | Ordered lookup versus state-space traversal

Binary search exploits the order of a sorted sequence: a comparison narrows the remaining candidate interval instead of exploring arbitrary successor states. Array bisection therefore differs from breadth-first or depth-first traversal of a general state-space graph. A storage container such as an array does not by itself specify FIFO or LIFO frontier behavior.

Python documentation - Array bisection and searching sorted lists

Introduction to Artificial Intelligence module · page 18

AI-S03 | Agent task models and practical planning

An AI task is modeled as an agent interacting with its environment: observations enter through sensors and selected actions affect the environment through actuators. This task-level description differs from an agent implementation, which combines a program with a computing architecture.

In ordinary usage, rationality means reasonable, sensible judgment. For an AI agent, the operational criterion is the action with the best expected performance given its percept history and available knowledge, rather than guaranteed success or knowledge of the actual future.

Planning constructs actions to reach a goal. Scheduling additionally assigns activities to times and resources while respecting constraints such as resource conflicts and precedence. It is an AI application when search or constraint reasoning chooses feasible or useful assignments; automated telescope experiment scheduling is an example.

Introduction to Artificial Intelligence module · page 11

Introduction to Artificial Intelligence module · page 7

AI-S04 | Attributing the name of artificial intelligence

John McCarthy is credited with coining the term artificial intelligence in the 1955 proposal for the Dartmouth summer research project. The Dartmouth workshop took place in 1956 and helped establish the field. Naming the field does not mean that one person invented all its ideas or methods.

Stanford HAI - What is Artificial Intelligence?

Dartmouth - Artificial Intelligence Coined at Dartmouth

AI-S05 | Related search and game-evaluation methods

Simulated annealing follows a current candidate and randomly samples a neighboring move. It accepts improvements and sometimes accepts worse moves according to a temperature that decreases over time. Unlike genetic search, its defining mechanism is not a population undergoing parent selection, crossover and mutation.

For two admissible heuristics, h_2 dominates h_1 when $h_2(n) \geq h_1(n)$ for every node. Dominance compares estimates; it is not itself a condition ensuring that an arbitrary heuristic never overestimates. Both estimates must separately satisfy the relevant admissibility assumptions.

A Monte Carlo game simulation samples possible continuations and uses their outcomes to estimate position value. An evaluation function is the scoring mapping itself; sampling is one way of obtaining an estimate, while minimax backs values up and alpha-beta removes branches that cannot affect that backed-up decision.

Russell and Norvig - Informed Search and Exploration

Silver et al. - Mastering the game of Go with deep neural networks and tree search

AI-S06 | Prediction, transfer and explicit representations

Transfer learning reuses representations or a model learned on an earlier task for a new task, for example adapting pretrained image features to new image classes. This is a reuse strategy, not a specification that the new task has no labels or rewards.

Support-vector machines include supervised classification and regression methods. They use support vectors in their decision function rather than being defined by a next-state conditional-dependence assumption. In estimation, variance measures how predictions change across different training samples; overfitting instead names poor generalization caused by fitting training-specific details.

An ontology describes entities or individuals, classes and properties and may state logical relationships and restrictions. It organizes explicit conceptual knowledge. Conditional probability tables and a probabilistic factorization are additional modeling commitments, not inherent in the word ontology.

An application programming interface (API) exposes defined operations through which software requests functionality. It can be an interface used by an agent, but the interface alone is not the general entity that perceives an environment and selects actions.

TensorFlow - Transfer learning and fine-tuning

scikit-learn - Support vector machines

scikit-learn - Bias-variance decomposition

W3C - OWL 2 Primer

NIST glossary - Application Programming Interface

AI-S07 | Behavior tests, understanding and software tests

Searle’s Chinese-room argument challenges whether executing formal symbol-manipulation rules is sufficient for understanding. It is a philosophical argument about understanding, not the conversational judge-and-hidden-participant procedure of the Turing test.

Black-box software testing examines externally visible functionality without inspecting implementation internals. A specification may concern any software behavior; this testing method does not by itself require conversational indistinguishability from a human.

John Searle - Minds, Brains, and Programs (author draft)

NIST glossary - Black box testing

Computer Security

Complete Study Notes

SEC01 | CIA and the property actually harmed

Confidentiality restricts disclosure: copying a credit-card database violates secrecy even if the original stays intact.

Integrity restricts unauthorized or improper changes: altering a balance or inserting a false message damages correctness.

Availability supplies timely, usable service to authorized users: crashes, resource starvation and denied access violate it.

One event can harm several goals.

Authentication establishes an identity or other claim; authorization decides permitted actions.

Accountability links actions to responsible entities through reliable identity and protected logs.

Non-repudiation supplies evidence supporting origin or receipt; signatures do not make denial physically impossible.

Information assurance includes confidence, risk management, protection, detection, response and restoration across the information lifecycle.

Data security protects information at rest, in use and in transit, including accidental corruption as well as intentional attacks.

Computer Security module

SEC02 | Threat, vulnerability, risk and control

An asset is something valuable: data, software, hardware, service or reputation.

A threat is a potentially harmful circumstance; a vulnerability is an exploitable weakness; an attack is an actual attempt; a control reduces likelihood or impact.

A software flaw is not automatically an active attack.

Risk concerns likelihood and consequence, not a universally meaningful numeric product of two unmeasured labels.

Preventive controls block events, detective controls reveal them, corrective/recovery controls restore safe operation, and deterrents discourage attempts.

Encryption is preventive for disclosure; an audit log is principally detective/accountability evidence.

Physical, technical and administrative are control categories, not competing stages.

An external firewall cannot replace a written policy or a locked equipment room.

Computer Security module

SEC03 | Attack classifications and attacker intent

Interception copies or observes data, modification alters it, interruption blocks it, and fabrication inserts false material.

The 4D classification uses disclosure, deception, disruption and usurpation, the last meaning unauthorized control.

DoS primarily demonstrates disruption/availability loss; it need not give control of the victim.

Passive attacks observe without changing the target exchange: sniffing and traffic analysis.

Active attacks inject, alter, replay or disrupt.

These are not synonyms for offline and online.

Traffic analysis can infer relationships from timing/size even when content is encrypted.

Neither category is universally easy to prevent or detect.

White-hat testing requires authorization and scope; black-hat activity is malicious or unauthorized.

A competitor may seek commercial secrets; hacktivists pursue a cause. “Hacker” alone does not establish legality.

Insider versus outsider describes origin/trust position, not the specific mechanism.

Computer Security module

SEC04 | Malware: propagation, disguise and payload

A virus replicates by attaching to a host program or document; execution of the infected host spreads it.

A worm replicates independently, commonly across networks.

A Trojan disguises malicious behavior as something useful and does not require self-replication.

These labels describe different properties, so a malware package may contain several components.

Spyware monitors users; a keylogger records keystrokes.

Adware displays unwanted advertising and may also track users.

Ransomware denies access through encryption/locking and extortion.

A logic bomb waits for a trigger.

A backdoor bypasses ordinary access paths; a rootkit hides malicious presence or privileged activity.

A zombie/bot follows remote commands; a botnet is a collection.

Malware is the umbrella term, not a specific replication method.

Update, restrict execution, isolate compromise and maintain tested offline backups; no antivirus guarantees complete protection.

Computer Security module

SEC05 | Social engineering, spoofing and sessions

Phishing impersonates a trusted source to obtain secrets or induce an unsafe action; spear phishing targets a specific person.

Verify a request through an independently known channel.

Shoulder surfing observes entry; pretexting invents a reason for disclosure.

Monitoring software used with authorization is not necessarily an attack merely because it records activity.

Masquerade claims another identity.

ARP spoofing poisons local IPv4-to-MAC associations; DNS spoofing/cache poisoning supplies false name resolutions; IP spoofing falsifies a source address.

An intermediary attack intercepts and can alter exchanges.

A replay reuses a valid old message; nonces, timestamps and sequence checks can reject it.

Session hijacking steals or predicts a valid session token.

Protect transport, regenerate session IDs after login, restrict cookies and check authorization server-side on every sensitive request.

Computer Security module

SEC06 | DoS, wireless attacks and layer-specific defenses

DoS exhausts or disables a resource; distributed DoS uses multiple sources, often bots or reflected/amplified responses.

A SYN flood leaves many incomplete TCP handshakes occupying state.

DNS flooding targets query capacity; NXDOMAIN attacks generate queries for nonexistent names to stress resolvers.

Fragment floods abuse reassembly or packet-processing resources.

Rate limits, SYN cookies, filtering and upstream capacity controls address different bottlenecks.

Rogue access points imitate or bypass approved wireless service.

Bluesnarfing steals Bluetooth information; bluejacking sends unsolicited Bluetooth messages.

Phishing is not specific to wireless.

Match a defense to its layer: authenticated IPsec can resist IP impersonation, DNSSEC authenticates DNS data, TLS authenticates/protects application transport, and bounds-safe code prevents overflows.

HTTPS cannot fix an unsafe memory copy in a server.

Computer Security module

SEC07 | Identification, authentication and factors

A common access sequence is identify the subject, authenticate the claim, authorize the operation, then record relevant events.

Identification is the username/claim, not its proof.

When a sequence begins with authentication, distinguish it from the preceding identification stage.

User-to-host means a user proves identity to a host; host-to-user means the host proves its identity to a user; the arrow names the prover and verifier.

Knowledge is a password/PIN, possession is a token/card, and inherence is a biometric such as a fingerprint.

Behavioral biometrics include typing rhythm or handwriting; some textbooks call these “something one does.” A body characteristic is not “three-factor authentication”: one biometric is one factor.

Password plus PIN remains one factor class; card plus PIN is two.

An OTP’s factor depends on its delivery/generation mechanism.

Computer Security module

SEC08 | Passwords and delegated authorization

Brute force enumerates guesses; dictionary attacks prioritize likely words; credential stuffing tries previously leaked credentials on another service.

Online attempts face server rate limits; offline attacks against stolen verifiers do not.

Short passwords can fail in either setting.

Use unique long secrets and password managers, rate limiting and suitable MFA; store salted slow password hashes, not reversible plaintext passwords.

A salt prevents shared precomputed lookup, but is not secret.

OAuth is an authorization framework: a client obtains limited access tokens, the authorization server authenticates/obtains approval from the resource owner, and the resource server validates tokens for protected data.

The authorization server can delegate identity checks.

OAuth alone does not define a complete end-user login identity protocol; OpenID Connect adds an identity layer.

Computer Security module

IETF RFC 6749

SEC09 | Access control and least privilege

Subjects request operations on objects.

An ACL associates an object/resource with allowed subjects or actions; a network ACL matches traffic such as address, protocol and port.

Classification and filtering are direct effects; high availability or malware detection is not guaranteed by an ACL.

DAC lets owners grant permissions; mandatory control enforces labels and policy that ordinary owners cannot override.

RBAC gives permissions to roles and assigns users to roles.

ABAC evaluates subject, object, action and environment attributes.

Rule-based controls apply explicit conditions, such as time or source network.

Least privilege gives only needed rights, for needed time.

Separation of duties splits sensitive steps across parties.

Deny by default, mediate every access and revoke stale permissions.

Allowing SELECT but denying UPDATE is an access-control decision that helps integrity; it does not by itself enforce all valid business-state transitions.

Computer Security module

SEC10 | Confidentiality and integrity models

Bell-LaPadula protects confidentiality.

The simple-security rule prohibits reading above a subject's level; the star rule prohibits writing down.

The strong-star variant permits writing only at the same level.

With TS>S>C>U, an S subject can read C under the simple rule but cannot write C under star; compartments and discretionary restrictions may still deny access.

Strict Biba protects integrity by preventing read down and write up: untrusted lower-integrity input should not contaminate higher-integrity results.

Clark-Wilson protects integrity through certified well-formed transformations, integrity checks and separation of duties.

It is not a confidentiality model. “MAC” is an enforcement category and does not force every system to use Bell-LaPadula rather than another label policy.

Paul Krzyzanowski, Rutgers: Computer Security exam guide

SEC11 | Physical protection and resilient design

Protect equipment against theft, unauthorized connections, heat, humidity, water, electrical surges and fire.

A UPS supplies temporary conditioned power; it is normally protective, but faulty installation/battery failure can itself be hazardous.

A disconnected machine loses many network attack paths yet remains exposed to insiders, removable media, supply-chain attacks and physical access.

There is no universal “most secure computer” without a threat model.

Defense in depth layers independent controls.

Minimization removes unnecessary software and services; compartmentalization limits blast radius.

Fail securely means errors do not silently grant permission.

Uniform configurations simplify patching and prediction but create correlated failures; diversity may reduce common-mode compromise while raising administration cost.

Choose protection proportional to asset value and credible threats.

Computer Security module

SEC12 | Cryptographic vocabulary and security assumptions

Plaintext is the original message, ciphertext its encrypted representation, a key is input to an algorithm, encryption computes C=E_k(P), and decryption recovers P=D_k(C).

The five conventional symmetric ingredients are plaintext, encryption algorithm, secret key, ciphertext and decryption algorithm.

Cryptanalysis seeks recovery or other breaks without the authorized secret; cryptology includes cryptography and cryptanalysis.

Steganography hides that a message exists; encryption hides its meaning.

Assume the algorithm is known and protect keys: secrecy of implementation is not the main security argument.

A b-bit key has 2^b possibilities; exhaustive search averages roughly half the space if keys are uniformly random.

Security depends on algorithm, key size, randomness, mode and implementation.

Symmetric cryptography is not inherently less secure than public-key cryptography, and speed ratios are not fixed constants.

Computer Security module

SEC13 | Substitution, shift and affine ciphers

Substitution replaces symbols; transposition changes their positions.

A monoalphabetic substitution preserves one mapping throughout the message and leaks frequency patterns despite a large nominal key space.

Classical ciphers are teaching devices, not modern data protection.

With A=0 through Z=25, Caesar encryption is C=(P+k) mod26 and decryption P=(C-k) mod26.

A shift of 3 maps A→D and HELLO→KHOOR.

Key25 moves back one: ATTACK→ZSSZBJ.

Ensure negative results wrap into 0-25.

A multiplicative cipher uses C=kP mod26 with gcd(k,26)=1; decryption uses k's modular inverse.

Affine uses C=aP+b mod26 and requires gcd(a,26)=1.

The inverse-key relation, not simply the same numeric multiplication, recovers plaintext.

Computer Security module

SEC14 | Vigenère, Playfair and Hill

Vigenère repeats or otherwise supplies shifts K_i and computes C_i=(P_i+K_i) mod26.

The same plaintext letter may encrypt differently by position; repeated short keys still leak periodic structure.

An autokey scheme extends its initial key using message material.

Playfair uses a keyed 5×5 grid, commonly combining I/J, and encrypts letter pairs.

Specify duplicate-letter fillers and padding.

Same row moves right, same column moves down, and rectangle pairs take the other column in their original row; decryption reverses shifts.

Hill uses C=KP mod26 for letter vectors.

K needs an inverse modulo26, equivalently determinant coprime to26.

It is linear: enough independent known plaintext/ciphertext blocks reveal K=CP^-1 mod26.

Hill is vulnerable to known-plaintext attacks; polygraphic substitution alone is not modern strength.

Computer Security module

Chris Christensen, Northern Kentucky University: Hill cipher

SEC15 | Transposition and one-time pads

Rail-fence writes along a specified zigzag and reads rails; columnar transposition writes rows and reads columns in a specified order.

They preserve the symbols and their counts.

Always state rows, column order, padding and direction before computing a transposition cipher.

Two-rail zigzag for ABCDEF writes ACE on the first rail and BDF on the second, producing ACEBDF.

A one-time pad XORs bits with an independent uniformly random secret key at least as long as the message and never reused.

Its perfect secrecy depends on every condition.

Reusing a pad leaks C1 XOR C2=P1 XOR P2.

A Vernam XOR construction is not automatically a one-time pad: a predictable/reused keystream loses that guarantee.

Key delivery and storage remain substantial costs.

Computer Security module

SEC16 | Symmetric keys, AES and distribution

Symmetric encryption uses a shared secret and efficiently protects bulk data.

Pairwise independent keys for n users require n(n-1)/2 shared keys; a trusted key distribution center changes that architecture.

A session key is temporary for a communication context; a master/key-encryption key protects subordinate keys.

Distribute keys physically, via an authenticated protected channel, or through a trusted third party.

An uncompromised old key can protect a new key, although this does not restore secrecy after the old key is stolen.

AES is a 128-bit block cipher with 128/192/256-bit keys and 10/12/14 rounds.

Its substitution/permutation design uses SubBytes, ShiftRows, MixColumns and AddRoundKey; the final round omits MixColumns.

Block ciphers need a mode and correct nonce/IV handling; authenticated encryption also detects tampering.

A stream cipher combines data with a keystream.

Never reuse a nonce when the chosen construction requires uniqueness.

Computer Security module

SEC17 | DES and Feistel mechanics

DES has a 64-bit block, 56 effective key bits (eight other bits serve parity), and 16 Feistel rounds.

After initial permutation, each round computes L_i=R_(i-1), R_i=L_(i-1) XOR f(R_(i-1),K_i).

Its right-half expansion makes 48 bits, XOR mixes a 48-bit subkey, eight S-boxes each map six bits to four, and permutation rearranges the 32 output bits.

The outer S-box input bits select a row and the inner four select a column.

The nonlinear S-boxes resist simple linear descriptions.

Feistel inversion uses the same overall structure with reversed round keys; f need not itself be invertible.

Final swapping/permutation completes DES.

DES and triple DES are legacy algorithms, not choices for new protection.

Triple DES repeats DES operations but does not provide 168-bit effective security merely because three 56-bit keys are supplied.

Computer Security module

SEC18 | Public keys and hybrid protection

For confidentiality to B, encrypt with B’s authenticated public encryption key; B decrypts with B’s private key.

Do not encrypt to A’s public key if only B should read it, and never assume A possesses B’s private key.

For a signature, A signs with A’s private signing key and others verify with A’s public key.

These are distinct operations; not every public-key algorithm supports encryption.

Public keys ease distribution but require authentic bindings to identities.

Hybrid systems establish or encapsulate a symmetric secret with public-key mechanisms, then protect bulk data symmetrically.

Diffie-Hellman agrees on a secret but unauthenticated DH permits intermediary substitution.

With public p,g, A=g^a modp and B=g^b modp, both derive B^a=A^b=g^(ab) modp.

Authenticate the exchange and derive separate traffic keys.

For one-way public-key confidentiality the sender need not own a key pair; it needs the recipient’s authenticated public encryption key.

A sender key pair is needed when the protocol requires its signature or separate recipient role.

Computer Security module

SEC19 | RSA arithmetic and key-validity conditions

For textbook RSA choose distinct primes p,q; n=pq and φ=(p-1)(q-1).

Choose 1<e<φ with gcd(e,φ)=1, then compute d=e^-1 modφ.

Public key is (n,e), private includes d. Encrypt 0≤m<n by c=m^e modn; decrypt by m=c^d modn.

Use modular repeated squaring for large exponents.

Real RSA requires appropriate standardized padding; raw arithmetic is not a secure deployment recipe.

With p=7,q=19, n=133 and φ=108, e=5 is the smallest admissible e>1, but 7 and other coprime choices also work.

For e=5, d=65 since 325≡1 mod108.

With p=3,q=13,e=3,m=5, 5^3 mod39=8 arithmetically, but gcd(3,24)=3, so e has no inverse and this is not a valid RSA key.

Check key validity before treating arithmetic results as a working encryption/decryption pair.

Computer Security module

SEC20 | Hashes, MACs and signatures

A cryptographic hash maps arbitrary-length input to a fixed-length digest.

It is deterministic and designed to resist preimages and collisions; collisions must exist, so hashes are not mathematically unique fingerprints.

An attacker can recompute an unkeyed hash of modified data, so a hash alone needs a trusted reference.

A keyed MAC provides integrity and shared-secret authentication; both holders can create it, limiting third-party non-repudiation.

A signature algorithm signs a message or its encoded digest with a private key and verifies with the public key.

It is not generically “encrypting with a private key,” especially for DSA/ECDSA.

Verification checks origin/integrity under key-identity and key-custody assumptions; it does not conceal content.

Password verification uses a salt and deliberately expensive password hash.

Historical MD5/SHA-1 examples do not justify their use for new collision-resistant signatures.

Computer Security module

SEC21 | Certificates, TLS and HTTPS

A certificate binds a public key to a named subject through an issuer signature.

Validate the chain to a trusted root, intended hostname, time validity and applicable key usage/revocation policy.

A genuine certificate is not proof that a business is honest.

TLS succeeds obsolete SSL and supplies an authenticated protected channel.

A handshake negotiates parameters, authenticates the server and sometimes the client, and establishes keys; record protection uses symmetric authenticated encryption.

Exact messages depend on TLS version: the old ChangeCipherSpec narrative is not a universal TLS1.3 sequence.

HTTPS is HTTP over TLS.

It protects HTTP paths, headers and content on the channel, but does not hide all metadata such as IP endpoints and traffic timing, nor protect data after a compromised endpoint decrypts it.

Computer Security module

SEC22 | SSH, IPsec, email protection and SET

SSH protects remote login, commands and file transfer such as SFTP; it supports host keys and several client-authentication methods.

IPsec protects IP traffic at network layer.

AH provides integrity/origin authentication, not encryption; ESP can provide confidentiality and integrity; IKE negotiates security associations and keys.

Transport mode protects upper-layer payload; tunnel mode encapsulates an inner IP packet inside an outer header.

A VPN is a logical private connection; its actual security depends on the chosen authenticated tunnel.

PGP/OpenPGP combines symmetric content protection with public-key key handling and signatures.

S/MIME uses certificates for signed and/or enveloped MIME content.

Signatures can be sent separately/clear-signed without confidentiality; enveloped data encrypts content and wraps content keys for recipients.

SET, Secure Electronic Transaction, was a Visa/Mastercard-led specification for protecting card transactions using certificates and signatures; it is distinct from a payment-processing system or ordinary HTTPS.

One signed-and-encrypted S/MIME construction first signs the MIME entity, packages its content and signature, then encrypts that signed entity with a fresh symmetric content key.

Protect a copy of that key for each recipient.

A recipient recovers the key, decrypts the signed entity and verifies its signature using the validated signer certificate/public key.

Signing is not generic private-key encryption.

S/MIME permits other nesting orders; specify the order and resulting coverage.

Computer Security module

IBM KeyWorks service-provider interface

S/MIME4.0 specification

SEC23 | Firewalls and protected boundaries

A firewall enforces traffic policy.

Stateless packet filtering matches headers such as source/destination, protocol and ports.

Stateful inspection tracks permitted connection context; application proxies terminate and relay application exchanges and can inspect their content.

A circuit gateway relays connections, commonly SOCKS, without full application-content interpretation; textbooks associate it with session/transport functions.

Such a gateway need not encrypt and is not automatically a VPN.

Host firewalls protect individual machines; network firewalls segment networks and can create a DMZ for public-facing services.

Rules should be explicit, ordered as implemented, least-privilege and logged appropriately.

Stateful and next-generation devices can also filter ports; port filtering is not exclusive to one firewall category.

Bypass paths, insiders, encrypted uninspected content and infected removable media limit firewall protection.

Hardware firewall does not imply any fixed number of ports.

Computer Security module

SEC24 | Detection, prevention and diagnostic measures

An IDS observes and alerts; an IPS can enforce blocking, usually inline.

Host-based sensors examine host events, network-based sensors inspect traffic.

Signature detection recognizes known patterns; anomaly detection compares with a baseline and may detect novel behavior.

Neither guarantees every attack is detected; anomaly detection can raise many false positives, and signatures can sometimes match variants.

A true positive is a real attack detected, false negative is a missed attack, false positive is innocent activity flagged, and true negative is innocent activity accepted.

Recall=TP/(TP+FN); precision=TP/(TP+FP).

Detecting 90 of 100 spam messages with five innocent messages flagged gives recall90%, precision90/95≈94.7%.

Obfuscation makes code harder to understand and can hamper analysis, but does not reliably protect embedded secrets or remove vulnerabilities.

Computer Security module

SEC25 | Memory safety and secure input handling

A buffer overflow writes beyond allocated bounds and can corrupt data or control information; code execution is a possible consequence, not its definition or an inevitable outcome.

An array of eight bytes cannot store eight text characters plus a terminating zero.

Bounds-check lengths and indexes, consider integer overflow, use safe abstractions and validate library return values.

Canaries, non-executable memory and address randomization reduce exploitability but do not repair the underlying write.

Validate expected type, length, range and structure at trust boundaries.

Server validation is essential because clients are controllable.

Encode output for its destination; validation is not a universal substitute for query parameterization or HTML-context encoding.

Security patches repair known weaknesses; a zero-day may lack a vendor fix, but segmentation and other mitigations can still help.

MITRE CWE-787

SEC26 | SQL injection, XSS and request forgery

SQL injection occurs when untrusted values alter SQL syntax through unsafe string construction.

A tautology can change which rows match.

Bind data values in prepared statements, allow-list identifiers that cannot be parameters, and restrict the application’s database account.

Stored procedures/ORMs are safe only when they avoid unsafe dynamic SQL.

XSS causes attacker-controlled script to execute in a victim’s browser under the site’s origin.

Stored content persists; reflected content returns in a response; DOM-based XSS can arise from unsafe browser sinks.

Use safe text insertion and context-specific encoding; carefully sanitize allowed HTML.

CSP is added protection, not the sole fix.

CSRF exploits ambient credentials to submit an unwanted authenticated action.

Use unpredictable request tokens and suitable cookie/origin controls.

HttpOnly restricts script access to cookies but does not stop every XSS effect.

Fuzzing sends unexpected inputs to discover faults; it is a testing technique, not itself a category of injection.

OWASP SQL Injection Prevention Cheat Sheet

OWASP Cross Site Scripting

SEC27 | Races, testing and secure development

A race occurs when correctness depends on timing/interleaving over shared state.

In a check-then-open file operation, another actor may replace the file between the check and use.

In a balance update, two workers can both read100 and each write90, losing one debit.

Protect the whole invariant with suitable atomic operations, locks or database transactions; locking an unrelated variable does not help.

Vulnerability assessment finds and prioritizes weaknesses; authorized penetration testing attempts to demonstrate consequences.

Static analysis inspects code, dynamic testing executes it, and fuzzing varies inputs.

Host/network/database scanners see different exposures and have false results.

Correct findings, patch, test and recheck; testing is not proof of absence.

Build threat modeling, review, least privilege, dependency management and safe error handling into development.

Computer Security module

MITRE CWE-362

SEC28 | Trusted OS requirements and reference mediation

A trusted OS is relied on to enforce a stated security policy.

Relevant mechanisms include identification/authentication, process and memory isolation, controlled privileges, object access enforcement, protected auditing, safe resource reuse, trusted boot/update paths and recovery behavior.

The trusted computing base comprises components whose failure can violate policy; smaller, analyzable boundaries simplify assurance.

A reference monitor is an abstract access-checking mechanism: it must mediate every relevant access, resist tampering and be sufficiently small/simple to analyze.

A security kernel implements core enforcement.

A firewall can mediate its traversing traffic but cannot enforce every host-memory access.

Assurance is justified confidence that controls implement the claims; functionality is what controls claim to do.

A GUI or multitasking support alone does not establish trust.

Computer Security module

NIST SP800-53 Rev5 AC-25 Reference Monitor

SEC29 | Independent evaluation, certification and assurance

Under Common Criteria, define the Target of Evaluation (TOE), its configuration and environment.

A Security Target (ST) states threats, assumptions, objectives, functional requirements and assurance requirements; a Protection Profile describes reusable requirements for a product category.

Evaluators examine developer evidence, test and analyze vulnerabilities against those claims.

A certification body oversees/reviews the evaluation and issues the certification decision; sponsors commission work and developers supply product/evidence.

An Evaluation Assurance Level packages assurance requirements; EAL1-7 express increasing evaluation rigor, not a universal ranking of security functions.

A higher number does not make a product with different scope automatically safer.

CCRA is a mutual-recognition arrangement, not a single worldwide testing laboratory.

A certificate applies to a specified product/configuration and claims, not immunity to all future attacks.

Common Criteria 2022 Part 1

Indian Common Criteria Certification Scheme FAQ

SEC30 | Database secrecy, permissions and key boundaries

Database security needs confidentiality, integrity, availability and accountability together.

Authenticate connections, give roles only necessary table/column/row privileges, restrict views and audit sensitive changes.

A user allowed to read a report need not read the underlying full table.

Check application-level authorization as well as database rights.

Encryption at rest protects stolen storage/backups when keys remain separate; transport encryption protects the connection.

Neither prevents a fully authorized query or guarantees correctness.

Strong access controls, encryption and protected transmission can all contribute to secrecy; compare their roles and assumptions.

Replication is useful for availability but also copies malicious updates unless separate recovery controls exist.

Computer Security module

SEC31 | Database integrity and reliability mechanisms

Integrity requires valid data and valid transitions.

NOT NULL rejects missing mandatory values; UNIQUE/primary keys prevent duplicate identifiers; foreign keys preserve references; CHECK enforces declared predicates.

Transactions group a transfer's debit and credit atomically; concurrency control prevents unsafe interleavings.

Roles/ACLs limit who may update, but constraints and transaction logic determine whether an authorized update is valid.

Write-ahead logging persists recovery records before dependent data-file changes, supporting recovery after a crash.

Backups, restore tests and separately protected logs recover from deletion/corruption; replication provides alternate service capacity but does not replace backups.

Redundancy improves availability only with appropriate failover and independent failure handling.

Distinguish confidentiality controls from reliability controls and from integrity constraints.

PostgreSQL constraints documentation

PostgreSQL write-ahead logging documentation

SEC32 | Inference control and multilevel databases

Inference reveals protected facts by combining individually permitted outputs with other knowledge.

If a report gives a two-person salary total and one salary is known, subtraction reveals the other.

Blocking direct SELECT of a salary column does not stop that inference.

Restrict small groups and overlapping queries, track query history, coarsen/suppress results, or introduce carefully designed noise; each trades utility against leakage.

A multilevel database stores items at different labels and enforces mandatory policy using subject clearances, classifications and compartments.

Bell-LaPadula-style read/write restrictions prevent direct downward confidentiality flow; trusted declassification is a separate controlled action.

Views alone are not proof of multilevel isolation.

Covert channels, query results and error messages may still reveal information, so evaluate both direct access and indirect inference.

University of South Carolina, Information Security Lab

SEC33 | Policy, incident response and recovery objectives

A security policy states responsibilities and rules; a procedure supplies the steps.

Policies coordinate protection, reduce risk and define acceptable use, access review, incident handling and recovery.

They do not prescribe merely “passive system use.” Awareness training teaches reporting and independent verification; enforcement and usable procedures make it operational.

Prepare response roles/tools, detect and analyze, contain the incident, eradicate the cause, restore verified service and learn from the evidence.

Preserve useful protected logs.

Disaster recovery restores IT after disruption; business continuity sustains business functions through workarounds and alternate facilities.

RTO is a recovery-time target; RPO measures the tolerable age of lost data.

A backup schedule must satisfy the required recovery point, and a restore test must demonstrate that data can actually be recovered.

Computer Security module

SEC34 | Interpret security mechanisms and conditions precisely

Compare each security mechanism with the stated requirement and conditions.

Check asymmetric-encryption key directions carefully; signatures do not generically encrypt whole messages with private keys.

Body biometrics do not imply three factors.

SSL and SSH can both describe protected channels; SSH specifically supports remote-shell services.

Read the full option text before matching it to a security mechanism.

Undefined “arbitrary code execution” does not uniquely identify buffer overflow: several mechanisms can produce that consequence.

A database-security requirement can include both availability and confidentiality.

Generic ACLs and RBAC both restrict unauthorized writes but neither independently enforces all integrity constraints.

Evaluation organizations must be identified by the actual national scheme; CCRA is recognition, not an evaluator.

An answer is supported only after the missing distinction is supplied.

Computer Security module

SEC-S01 | Command injection and the data-to-command boundary

OS command injection occurs when an application lets untrusted input affect a command executed by the host operating system, often through unsafe construction of a shell command. The command runs with the application's effective privileges. This is an injection flaw in command interpretation, unlike fuzzing, which supplies varied inputs to discover defects.

Prefer a suitable library API over constructing a shell command. Where command execution is required, keep the executable and arguments under strict control and validate against the expected permitted structure; merely blocking a few suspicious characters is not a general proof of safety.

OWASP - Command Injection

SEC-S02 | WannaCry: ransomware and worm propagation

WannaCry, also known as WannaCrypt, is ransomware with worm-like propagation. It encrypts files and demands payment to restore access. Its name identifies malicious software rather than a network-security protocol. Malware names must be distinguished from defensive controls such as intrusion detection, intrusion prevention and physical protection.

Microsoft Defender Security Research Team - WannaCrypt ransomware worm targets out-of-date systems

CERT-EU - WannaCry Ransomware Campaign

SEC-S03 | Distinguish network roles and protocol names

A router forwards IP packets between networks; forwarding alone is different from detecting suspicious behavior or enforcing a firewall's security policy. SNMP means Simple Network Management Protocol and provides a framework for monitoring and controlling managed devices through management information. It is not a remote-shell service.

AAA names authentication, authorization and accounting functions, rather than a particular remote-shell channel. SSH expands to Secure Shell and provides secure remote login and related network services. PGP expands to Pretty Good Privacy; its message-protection design combines conventional and public-key encryption with digital signatures. Protocol purpose matters when several technologies all contribute to security.

RFC 1812 - Requirements for IP Version 4 Routers

RFC 3411 - SNMP Management Frameworks

RFC 2903 - Generic AAA Architecture

RFC 4251 - The Secure Shell Protocol Architecture

RFC 1991 - PGP Message Exchange Formats (historical terminology)

Network and System Administration

Complete Study Notes

NSA01 | Roles, principles and change discipline

System administrators maintain hosts, accounts, storage, software and availability; network administrators maintain connectivity, addressing, routes, network devices and performance.

Security administrators coordinate controls and policy.

These duties overlap, so account management, testing patches and training users are not exclusive to one job title.

Patience, communication, ethics, technical understanding and organization matter alongside command knowledge.

Use documentation, standardization, least privilege, automation and measured change.

Record the current state, intended change, rollback and validation.

Schedule maintenance according to service impact and urgency, not merely because a calendar date arrived.

Fault management restores failures; performance management measures service behavior; configuration management tracks settings; security management controls access; accounting tracks usage.

Data Communication and Computer Networking module

Network and System Administration module

NSA02 | Network foundations for administrators

Clients request services; servers provide them.

Peer-to-peer systems can do both.

A server can also be a client of DNS or backups, so dedicated role does not mean it never makes requests.

A network OS is not universally installed only on a dedicated server: clients also require networking support.

Specify centralized service software versus the OS on all participants.

Switching normally forwards frames using MAC addresses within a LAN; routing forwards IP packets between networks.

TCP supplies an ordered reliable byte stream, while UDP preserves datagrams without that guarantee.

Ports distinguish application endpoints, not physical connectors.

Draw switches, links, subnets, gateways, service hosts and dependencies in a network diagram.

A topology's shape alone does not guarantee security.

Data Communication and Computer Networking module

Network and System Administration module

NSA03 | Address, prefix, gateway and loopback

An IPv4 address is 32 bits; a CIDR prefix determines its network.

Compute network=address AND mask; devices use a route/gateway for remote subnets.

Private ranges are 10/8,172.16/12 and 192.168/16; private addresses are not directly Internet-routable.

Neither IP alone nor a class label determines today's subnet.

127.0.0.1 conventionally names the local IPv4 loopback interface; 127/8 is reserved for loopback.

Loopback tests local protocol processing, not cable, switch or remote reachability.

Programs can use it for local services.

Several loopback descriptions or several 127/8 addresses may be valid; compare each with the stated role.

Example

Worked microexample

- 192.168.2.70/26 has 64-address blocks: network .64, broadcast .127, ordinary host range .65-.126. A gateway must be reachable through an appropriate route; NAT does not repair a missing route.

Data Communication and Computer Networking module

Network and System Administration module

RFC 1122 host requirements

NSA04 | DHCP exchange and lease lifecycle

DHCP supplies addresses and configuration options.

Typical initial IPv4 allocation is Discover→Offer→Request→ACK.

The server manages its scope/pool, allocated leases, exclusions and reservations; options may include mask, gateway and DNS servers.

A dynamic lease lasts for a specified interval, not forever.

A client renews before expiry; it later tries rebinding if renewal fails.

A disconnected host does not automatically renew forever.

On lease expiry the address becomes eligible for reuse, subject to server policy.

Reconnection normally triggers validation/renewal/new allocation, so an expired lease alone does not prove persistent failure.

A reservation gives a predictable address to a known client while retaining DHCP configuration.

Example

Worked microexample

- If a disconnected client's lease expires while other clients obtain addresses, its old address may be reassigned. It must not simply resume using it without validating its configuration.

RFC 2131 DHCP

NSA05 | DHCP deployment, fallback and diagnosis

Use a scope matching the subnet, exclude manually assigned server/router addresses, set correct options and provide a relay when client broadcasts must cross a router.

Static assignments and DHCP can coexist; avoid address conflicts.

DHCP lowers repetitive administration but availability/security depend on deployment and protection from rogue replies.

It does not guarantee connectivity whenever a device changes location.

For a returning client, inspect current lease/address, reachable server/relay, free pool capacity and logs.

New devices can exhaust a pool, but absent observations this is only a hypothesis.

Some clients self-configure IPv4 link-local addresses such as 169.254.2.33 after DHCP failure; these support same-link communication, not ordinary routed Internet access. 0.0.0.0 is unspecified and 255.255.255.255 is limited broadcast.

Data Communication and Computer Networking module

Network and System Administration module

RFC 3927 IPv4 Link Local

RFC 2131 DHCP

NSA06 | DNS forward/reverse resolution and evidence

Forward resolution obtains an address from a name, using A for IPv4 and AAAA for IPv6.

Reverse lookup uses a PTR record in the reverse namespace to obtain a name from an address.

CNAME aliases a name, MX identifies mail exchangers and NS identifies authoritative name servers.

Forward and reverse records need not both exist or be perfect inverses.

Resolvers cache answers for their TTL.

Query nslookup name or specify the DNS server explicitly; compare the configured resolver with the authoritative answer when appropriate.

A successful ping by address but failed name lookup suggests naming/configuration trouble.

Ping by name may display a resolved address, but nslookup is the direct DNS diagnostic. “Resolving” is a general term that includes forward resolution.

Data Communication and Computer Networking module

Network and System Administration module

RFC 1035 DNS

Microsoft nslookup

NSA07 | Choose the command that exposes the required state

Windows ipconfig shows addresses, mask and gateway; ipconfig /all adds complete adapter configuration including physical/MAC address, DHCP and DNS information. /renew requests renewal, /release releases a dynamic address and /flushdns clears the client DNS cache.

Linux ip addr and ip route show interface addresses and routes; exact persistent configuration varies by distribution.

netstat displays connections/endpoints/protocols and TCP states; netstat -ano displays numeric addresses/ports and PIDs.

Use the exact command name netstat rather than netstart. ping probes IP reachability; traceroute/tracert reveals responding hops using TTL/hop-limit expiry.

ICMP filtering can prevent replies without proving the service is down. nslookup inspects DNS.

Data Communication and Computer Networking module

Network and System Administration module

Microsoft ipconfig

Microsoft netstat

Microsoft nslookup

NSA08 | A repeatable troubleshooting procedure

Define the symptom, scope, time and recent changes; gather the configuration, baseline and logs.

Form a testable hypothesis, test the least disruptive explanation, isolate the failing component, apply a targeted fix, verify the user service and document the result.

Change one relevant factor at a time when possible.

A reboot can hide evidence and is not a diagnosis by itself.

Use bottom-up checks from physical link, top-down checks from application errors, or divide-and-conquer by testing a middle layer.

Divide-and-conquer narrows the working/failing boundary; it is not random trial-and-error.

Rebooting, updates, resetting settings and scanning can all be troubleshooting actions in suitable circumstances; use fault evidence to choose among them.

Example

Worked microexample

- One client fails while others work: inspect that client's link, address and policy first. All clients fail name lookup but reach server IPs: inspect shared DNS/its configuration before replacing every cable.

Data Communication and Computer Networking module

Network and System Administration module

NSA09 | Connectivity diagnosis by layers

Check power, cable/connector and link state, then NIC/driver state, VLAN membership, address/prefix, local gateway, route, DNS and finally application port/authentication.

A recognized NIC makes a missing driver less likely, not proof of a working physical link.

Ethernet commonly uses RJ45 connectors; an RJ11 telephone jack is not the expected Ethernet connection.

If same-subnet traffic works but remote traffic fails, compare gateway/routes and filtering.

If ping fails but the application connection works, ICMP filtering is plausible.

If an application port is refused, the destination is reachable enough to reject it; inspect service/listener policy.

A timeout can result from loss, filtering or a down host and needs more evidence.

Example

Worked microexample

- After NIC replacement, recognized/enabled adapter but no link light: try a known-good Ethernet cable and switch port. Merely knowing that the OS recognizes the NIC cannot prove one particular cable is defective.

Data Communication and Computer Networking module

Network and System Administration module

NSA10 | Service catalog and remote access

DNS resolves names; DHCP supplies host configuration.

SMTP sends/relays mail, while IMAP/POP retrieve it.

FTP is a named file-transfer protocol; HTTP also transfers files, so “transfers files” is not an exclusive FTP description.

HTTP/HTTPS serve web resources.

SSH provides encrypted remote login; SFTP transfers files over SSH and differs from FTP or FTPS.

A VPN extends private connectivity over another network; secure tunneling commonly encrypts/authenticates traffic but depends on protocol/configuration.

RTP carries real-time media timing/sequence information, often over UDP; it does not guarantee timely delivery.

Network printers expose services shared by clients.

Provide accounts, permissions, configuration, monitoring and recovery for each service, not just its protocol name.

Data Communication and Computer Networking module

Network and System Administration module

NSA11 | Routing, NAT and port translation

Routing selects a next hop using a destination prefix; NAT rewrites addresses.

Many-to-one address-and-port translation (PAT/NAPT) maps internal flows to one public address using distinct port bindings.

For example, two private client sockets can appear as public-address ports 40001 and 40002, with replies translated back to the correct clients.

NAT is often used when private IPv4 clients access public networks; it does not itself replace a gateway, route, firewall or authentication.

One-to-one NAT and destination translation are other forms, so NAT does not always mean many hosts share one address.

Public addressing inside a LAN does not automatically require NAT.

Unsolicited inbound service access may require explicit mapping and filtering.

Data Communication and Computer Networking module

Network and System Administration module

NSA12 | Proxies, gateways and quality of service

A proxy acts on behalf of a client or backend service and can cache, filter or mediate application requests.

A default IP gateway is a next-hop router; an application gateway may translate protocols.

It need not be a modem/router combination.

These terms describe functions that can coexist on one machine.

QoS classifies, queues and prioritizes traffic to meet performance objectives such as low delay or limited loss.

It can operate across layers and support applications, so it is not exclusively an application-layer service.

Installing a new service is an administrative action; email configuration is another action and proxies/agents are components.

Distinguish service-management actions from components using a stated taxonomy.

Data Communication and Computer Networking module

Network and System Administration module

NSA13 | Workgroups, domains and directory services

A workgroup has locally managed accounts on individual machines; sharing may require local credentials and per-host configuration.

A domain centralizes identities and policies through directory infrastructure.

These are administrative arrangements, not physical topologies.

Avoid treating an old recommended workgroup size as a hard protocol limit.

HomeGroup is a historical Windows sharing feature, not another name for an AD domain.

A domain controller runs AD DS, authenticates domain accounts and participates in directory replication.

Redundant controllers improve service continuity only if dependencies and failure domains are considered; two virtual controllers on one failed physical host are not independent availability.

Provisioning requires correct addressing, DNS, time and supported OS/hardware; version-specific minimum sizes are not universal architecture rules.

Data Communication and Computer Networking module

Network and System Administration module

NSA14 | LDAP operations and secure directory access

LDAP accesses directory entries/attributes identified by distinguished names.

Bind authenticates a session; search retrieves matching entries; compare tests an attribute value; add/delete/modify change entries; modify-DN renames or moves; unbind ends the session.

StartTLS upgrades a connection to TLS where supported; LDAPS uses TLS from connection setup.

Protect credentials and verify server identity.

LDAP commonly uses TCP 389 and LDAPS TCP 636; an acronym/port alone does not make authentication secure.

AD also uses DNS and Kerberos; LDAP is not identical to the entire directory product.

Do not equate modern TLS-protected LDAP or LDAPS with retired LDAPv2.

Data Communication and Computer Networking module

Network and System Administration module

Microsoft, Enable LDAP over SSL

NSA15 | AD logical structure and delegation

A directory object represents a user, computer, group or resource and has schema-defined attributes.

Security principals have SIDs; display names are not the sole security identity.

A domain groups directory objects and a replicated domain database.

A tree groups domains in a contiguous DNS namespace; a forest groups trees with a shared schema/configuration and global-catalog framework.

OUs organize objects and provide convenient scopes for delegated administration and Group Policy; they are not separate forests/security boundaries.

Use groups to assign access and OUs to structure policy/delegation.

Schema changes require care because they propagate broadly; ordinary directory objects can be deleted, unlike the special restrictions on schema definitions.

Apply schema restrictions to the specific directory object type being discussed.

Data Communication and Computer Networking module

Network and System Administration module

NSA16 | AD sites, replication and trusts

Sites group IP subnets to model network locality, influence controller selection and manage replication traffic.

They are distinct from domains/OUs.

AD replicates directory changes among controllers; notification can trigger pulls within a site while schedules/link costs influence traffic between sites.

Replication is not an independent backup of previous good states.

If domain A trusts B, B’s identities can be authenticated for resources in A, subject to permissions.

Trust alone grants no automatic file access.

Direction (one/two way) and transitivity (extends/does not extend) are independent.

An explicit trust is not necessarily one-way or nontransitive; the trust type/configuration determines that.

State the direction in words before drawing arrows.

Data Communication and Computer Networking module

Network and System Administration module

Microsoft AD trust concepts

NSA17 | Accounts, rights and password diagnosis

Create named accounts, assign only needed groups/rights, disable departed users and audit privileged activity.

Authentication verifies the account; authorization permits operations.

A user right such as remote logon differs from a permission on one file.

Group-based administration is easier to review than many hidden individual grants.

If only one domain user cannot change a password, inspect that account's cannot-change-password setting and relevant policy/permission before blaming all directory infrastructure.

Other possibilities include minimum password age, password rules or account state; the setting is a plausible exam cause, not a conclusion from failure alone.

Membership in Administrators does not inherently prohibit changing one's password.

Privileged reset and user-initiated password change are different operations.

Data Communication and Computer Networking module

Network and System Administration module

NSA18 | Windows permissions and effective access

Windows file/folder permissions include read, write, read/execute, modify and full control, plus more granular rights such as delete.

Inherited and explicit entries, group membership, deny entries and object scope affect effective access; use the effective-access view for a real account rather than adding labels informally.

Network access can be constrained by both share and filesystem permissions.

Deleting can be an explicit permission in Windows; specify the operating system before comparing permission names.

Modify and full control are not identical: full control includes changing permission/ownership capabilities.

Separate data access from logon rights.

A user allowed to read a file is not necessarily allowed to write it or administer its parent folder.

Data Communication and Computer Networking module

Network and System Administration module

NSA19 | Unix rwx and octal permissions

The nine ordinary permission bits are owner, group and other triplets.

In each triplet read=4,write=2,execute=1; sum enabled bits to get an octal digit.

A leading ls -l character denotes file type, not another permission bit.

For directories, read lists names, execute permits traversal/search, and write changes entries subject to other checks.

File deletion usually depends on parent directory permissions, with sticky-bit/ACL rules also possible.

Example

Worked microexample

- -rwxr--r-- means ordinary file, owner7/group4/other4, hence 744. 755 means rwxr-xr-x. A permission string must include all nine ordinary permission bits; missing hyphens cannot be inferred. Confirm the complete nine-bit permission string before converting it to octal.

Data Communication and Computer Networking module

Network and System Administration module

Linux man-pages chmod(2)

NSA20 | Linux paths, filesystems and checks

/ is the filesystem root; /root is the administrative user's home. /etc stores configuration, /home ordinary user homes, /var variable data/logs, /dev device nodes, /proc kernel/process information, /boot boot material and /tmp temporary files.

Cleanup/mount layouts vary; do not assume every partition has lost+found or every reboot erases every temporary directory.

ext2 is a Linux filesystem; ext4 is a later filesystem; HPFS is historically associated with OS/2, while swap is backing space rather than an ordinary user-file filesystem. fsck checks and optionally repairs filesystems through filesystem-specific tools; it is not init, halt or an inittab file.

Repairs require the filesystem's supported offline/safe procedure.

Mount attaches a filesystem to a directory; partitioning, mounting, quotas and capacity belong to storage administration.

Linux can access some foreign-origin filesystems, including HPFS when its driver is available.

Filesystem origin and Linux driver support are different questions.

Swap is a paging area rather than an ordinary user-file filesystem.

ext2 is a native Linux-filesystem example; broader Linux filesystem support can also include HPFS, so distinguish origin from driver support.

Data Communication and Computer Networking module

Network and System Administration module

util-linux fsck(8)

Linux kernel Read/Write HPFS documentation

NSA21 | Profiles, registry and persistent configuration

A local profile keeps settings on one client; a roaming profile synchronizes supported user settings/data through a server at logon/logoff.

Large profiles can slow these operations; redirected folders avoid repeatedly copying selected folders.

A central profile is backed up only if the server's backup policy actually includes it.

A cached profile is not a newly created independent local security account.

The Windows registry is a hierarchical configuration database with keys, subkeys and typed values.

HKLM describes machine settings and HKCU the current user view.

ACLs protect registry access.

Not all historical root handles apply to current versions.

Record/export the relevant configuration before an authorized change, and verify the intended service behavior afterward.

Data Communication and Computer Networking module

Network and System Administration module

NSA22 | Performance monitoring and bottleneck decisions

Measure a baseline, sample over an interval, compare with workload and correlate symptoms before changing hardware.

CPU utilization/run queues identify processing pressure; low free memory alone is weak evidence because cache may be reclaimable.

Persistent paging with high latency indicates memory pressure.

Storage capacity and I/O latency are separate: a disk can be nearly empty but saturated, or nearly full while lightly loaded.

Measure interface bytes/second, errors, drops, packet delay/loss and utilization against link rate.

Distinguish capacity (maximum available rate) from observed throughput.

Logs can reveal both security incidents and performance faults.

Performance counters expose measured events; system calls/APIs and log parsers are ways scripts can collect them.

Example

Worked microexample

- Transferring 80 MB in 10 s gives 8 MB/s≈64 Mb/s before overhead. A 95%-busy CPU with a growing runnable queue suggests CPU pressure; a 95%-full disk requires capacity action even if its latency is low.

Data Communication and Computer Networking module

Network and System Administration module

NSA23 | Monitoring, capture and assessment tools

Task Manager shows running processes/resource use; Resource Monitor/Performance Monitor give more detailed counters.

Wireshark captures and dissects packets.

Nmap discovers hosts/ports/services.

Nessus assesses vulnerabilities/configuration.

Snort can detect suspicious traffic and alert, and can prevent traffic in suitable inline deployments; it also has sniffing/logging roles.

Thus packet-capture capability is not exclusive to one product, although Wireshark is the conventional analyzer choice.

An IDS detects/alerts; an IPS can block inline.

Scanning identifies potential weaknesses rather than proving successful exploitation.

A captured packet, open port or alert needs context.

Monitoring network usage is not the same as configuring access control or encrypting communications.

Data Communication and Computer Networking module

Network and System Administration module

Wireshark User Guide, Introduction

Nmap Reference Guide

Snort project

Tenable Nessus User Guide

NSA24 | Backups, restore and continuity

Backups recover previous data after deletion, compromise, corruption, failed upgrades/migrations or hardware loss.

Authorization controls current access, a different function.

A mirror/replica also copies deletions or corruption and does not replace versioned independent backups.

Keep protected copies in separate failure domains, define retention and test restores.

A full backup copies the chosen dataset; incremental copies changes since the last backup; differential copies changes since the last full backup.

Restoring incrementals generally needs the full plus each subsequent incremental, whereas differential needs the full plus latest differential.

RPO is tolerable lost-data time and RTO tolerable restoration time.

Compression reduces storage, scheduling runs the operation and restore testing establishes whether recovery actually works.

Data Communication and Computer Networking module

Network and System Administration module

NSA25 | RAID layouts, capacity and selection

RAID0 stripes for capacity/throughput with no disk-failure tolerance.

RAID1 mirrors; a two-disk mirror has one disk's usable capacity and survives one disk failure.

RAID2 uses bit striping/Hamming-code redundancy, RAID3 byte striping with dedicated parity, RAID4 block striping with dedicated parity and RAID5 block striping with distributed parity.

RAID6 distributes dual parity and tolerates two disk failures.

RAID10 stripes mirrored pairs; survival depends on which disks fail.

With N equal disks of capacity D, RAID0 usable=ND, paired RAID1/10≈ND/2, RAID5=(N−1)D and RAID6=(N−2)D. Hardware RAID may offload work to controller resources/cache but does not universally outperform software RAID.

Rebuilds consume bandwidth; parity writes have overhead.

A large university/bank alone does not select a level.

Striping plus parity is not uniquely RAID5 unless distributed block parity is specified.

Example

Worked microexample

- Four 2 TB disks give 8 TB RAID0, 4 TB RAID10, 6 TB RAID5 or4 TB RAID6, ignoring metadata. Choose using failure tolerance, workload, capacity and recovery requirements; none of these layouts replaces backups.

Data Communication and Computer Networking module

Network and System Administration module

Intel server storage options

NSA26 | Security policy development and purpose

A security policy states required behavior, scope, accountable roles and enforcement.

Begin with assets/service objectives, assess threats and weaknesses, choose controls, define exceptions/incident procedures, approve/communicate, monitor and periodically review.

CIA means confidentiality, integrity and availability; neither sensitive-data protection nor availability alone is the universal primary goal.

A policy can require password controls, approved encryption, restricted administrative access, backups and patch deadlines; firewall rules implement part of it.

Open unrestricted wireless access conflicts with controlled internal access unless intentionally isolated as a guest service. “Use the system passively” is not a purpose of security policy.

Policies coordinate action and reduce risk; they do not guarantee every attack will fail.

Data Communication and Computer Networking module

Network and System Administration module

NSA27 | Hardening, controls and vulnerabilities

Hardening reduces exposure through unnecessary-service removal, supported patches, restricted accounts/permissions and appropriate port filtering.

Blocking an unused known port can be hardening; blocking every useful port merely destroys availability.

Weak passwords and unpatched exploitable software are vulnerabilities; attacks exploit weaknesses, and risk combines likelihood with impact.

Preventive controls restrict harmful actions, detective controls expose them and recovery controls restore service.

Account lifecycle, backup procedures, device configuration and traffic analysis can all be security controls under a stated objective.

A vulnerability scan checks known weakness patterns, not bandwidth.

Test/stage patches, prepare rollback, deploy and verify; testing and installation are both legitimate administrator duties.

Data Communication and Computer Networking module

Network and System Administration module

NSA28 | Encryption, hashes and script secrets

Encryption provides confidentiality with correct key/nonce use; authenticated encryption adds integrity/authenticity.

Unspecified encryption alone does not guarantee both properties.

A password hash supports verification without recovering the original password; it is not a reversible credential store.

Encryption can protect a recoverable credential only if its key is separately protected.

Keep reusable secrets out of source code and routine logs; obtain them from an appropriately protected credential service or restricted runtime source.

Environment variables and configuration files are transport/storage choices, not automatic encryption.

A script that must authenticate to another service often needs the actual credential/token, so replacing it with a one-way hash will not work.

State whether the question is about password verification or recoverable service credentials.

Data Communication and Computer Networking module

Network and System Administration module

RFC 5116 Authenticated Encryption

NSA29 | Incidents, evidence and reports

Incident response prepares, detects/analyzes, contains, removes the cause, restores service and learns from the event.

Investigation is reactive response; routine patching and backup are normally preventive/preparatory operations.

Document timestamps, affected systems, evidence, impact, actions, responsible people and recovery results.

An incident report may record policy violations, but its scope is broader than that alone.

DoS harms service availability; DDoS uses distributed sources.

Phishing deceives users, brute force tries credentials and an interception attack reads/alters communications.

Correlate logs using time and identity/context.

Preserve relevant evidence before resetting systems.

A suspicious event is not automatically confirmed compromise; record facts and uncertainty.

Data Communication and Computer Networking module

Network and System Administration module

NSA30 | Segmentation, NAC and remote administration

Segmentation separates groups of systems and enforces allowed flows with routing/filtering policies; VLAN membership alone does not establish complete security.

NAC assesses device/user admission and can reject or quarantine unauthorized/noncompliant devices.

Authentication proves an identity, while access rules decide which resources it may use.

Use authenticated protected remote administration such as SSH or properly secured management services, with restricted source access and auditing.

VPNs provide protected connectivity under their defined security model; they are not packet analyzers.

Firewall filtering permits/blocks traffic according to policy; it is not inherently encryption.

Losing remote connectivity while changing a management route/firewall is an operational risk to plan for.

Data Communication and Computer Networking module

Network and System Administration module

Cisco, Network Access Control

NSA31 | Scripting: inputs, control flow and reusable work

Scripts automate repeatable administration; they are not inherently faster than compiled programs or automatically secure.

Python, shells and PowerShell are common choices; HTML is markup.

Define inputs and outputs, use variables for state, if/elif/else for decisions, loops for repeated work and functions for reusable operations.

Python indentation defines blocks; == compares while = assigns.

Make reruns predictable: compute the desired state, compare with current state and change only what differs.

Validate a small dry run before mutation.

Avoid requiring privilege for read-only reporting.

A good script reports failures and has a meaningful exit status rather than printing success unconditionally.

Example

Worked microexample

- def warn_percent(values, limit): for name, percent in values: if percent >= limit: print(name, "high", percent) warn_percent([("root", 92), ("data", 40)], 90) # Output: root high 92 (with spaces from print). Only the first pair meets the threshold.

Python tutorial, control flow

NSA32 | Launching commands with Python subprocess

Python subprocess is a module, not a universal shell command. run launches a child program and returns its result; capture_output collects stdout/stderr, text=True decodes text, timeout bounds waiting and check=True raises for nonzero exit.

Pass an argument list to avoid accidental shell interpretation; quote handling differs from composing a shell string.

Echo prints, grep searches text and ipconfig/ifconfig reports/configures networking; none is the general Python child-process abstraction.

Specify the scripting language before selecting subprocess for process execution.

Example

Worked microexample

- import subprocess result = subprocess.run(["ip", "addr", "show"], text=True, capture_output=True, check=True, timeout=5) print(result.stdout) # Linux with iproute2 installed. Missing program raises OSError; # nonzero status raises CalledProcessError; timeout raises TimeoutExpired.

Python subprocess documentation

NSA33 | File input, configuration parsing and updates

File I/O obtains/persists configuration; string operations parse values; regex can recognize patterns.

File I/O, string operations and regex can all help manipulate configuration; choose according to the task.

Use the format's parser for structured JSON/INI rather than blindly replacing every matching substring.

Open text with an explicit encoding and use with so files close even when processing fails.

For changes, validate the desired output, preserve a known-good copy, write to a temporary file in the same filesystem, then atomically replace the target where supported.

Check permissions and configuration syntax, reload the affected service, verify and retain a rollback.

Merely appending the same setting on every run is not idempotent.

Example

Worked microexample

- import json with open("settings.json", encoding="utf-8") as handle: config = json.load(handle) port = config["port"] if not isinstance(port, int) or not 1 <= port <= 65535: raise ValueError("port must be 1..65535") print("validated port", port) # This example validates/reports and does not alter a system file.

Python tutorial, input/output

NSA34 | Arguments and targeted error handling

Command-line parsing converts user inputs into named, typed values and displays usage.

In Python argparse, positional arguments supply required inputs and flags enable options.

Validate ranges and paths before work; do not let unchecked input become executable shell text.

Example

Worked microexample

- import argparse from pathlib import Path p = argparse.ArgumentParser() p.add_argument("logfile", type=Path) p.add_argument("--limit", type=int, default=10) a = p.parse_args() if a.limit <1: p.error("limit must be positive") try: print(a.logfile.read_text(encoding="utf-8")[:a.limit]) except (OSError, UnicodeError) as exc: p.exit(1, f"read failed: {exc}\n") # Input limit controls characters here. Exceptions skip the rest of try; # catching specific failures makes unexpected programming errors visible.

Python argparse documentation

Python errors and exceptions tutorial

NSA35 | Regex log analysis and rotation

Regex recognizes textual patterns; it does not encrypt, compress or rotate logs. search finds a match anywhere, while fullmatch requires the entire string to fit.

Use raw Python strings for backslash-heavy patterns, capture only the needed fields and treat unmatched/malformed lines explicitly.

Log rotation bounds file size and preserves historical segments; it is complementary to parsing.

Count the event actually requested, use timestamps/hosts to correlate, and separate absence of matches from failure to read the log.

Logs support security and performance analysis, so neither purpose is universally exclusive.

Example

Worked microexample

- import re pattern = re.compile(r"status=(\d{3})\b") counts = {} for line in ["status=200 ok", "status=503 unavailable", "noise"]: match = pattern.search(line) if match: code = match.group(1) counts[code] = counts.get(code,0)+1 print(counts) # {"200":1,"503":1}; noise is deliberately ignored.

Python re documentation

NSA36 | Collecting CPU and memory measurements

A cumulative CPU counter must be sampled twice to calculate interval use; a single lifetime total is not current utilization.

On Linux /proc/stat exposes cumulative CPU categories; define which categories count as idle and avoid double-counting guest fields.

A simple convention is busy fraction=1−Δidle/Δtotal using idle+iowait as the chosen nonbusy categories.

Iowait interpretation has limitations.

/proc/meminfo exposes MemAvailable, an estimate of memory available without swapping; MemFree alone excludes reclaimable memory.

Windows Performance Counters and OS APIs expose analogous measurements through different interfaces.

System calls, counters and log parsing can all contribute; selecting one as the only monitoring technique is unjustified without a platform/API.

Example

Worked microexample

- If total CPU counters grow by 1000 ticks and chosen idle counters by 250, measured busy≈75%. If MemTotal=8192 MiB and MemAvailable=2048 MiB, approximate unavailable/used fraction is75%, not a prediction that an allocation of exactly 2048 MiB must succeed.

Data Communication and Computer Networking module

Network and System Administration module

Linux proc_stat(5)

Linux proc_meminfo(5)

NSA37 | Scheduling a backup and checking restoration

Schedule recurring work using cron/systemd timers on suitable Unix systems or Task Scheduler on Windows.

A user crontab uses minute,hour,day-of-month,month,day-of-week then command.

Use absolute paths, known environment, protected inputs, logs, bounded runtime and a lock when overlap is unsafe.

Do not assume an interactive PATH, current directory or stored credentials exist in a scheduled job.

Compression packages data; scheduling chooses when the operation runs.

Test exit status and the archive contents, then restore a sample into an isolated destination.

A successful command alone does not prove a usable backup.

Version control of scripts helps review but does not replace data backup.

Example

Worked microexample

- One user-crontab entry (keep the command on one line): 0 2 * * * /usr/bin/python3 /opt/backup.py /srv /bak >>/var/log/bak.log 2>&1 This runs daily at 02:00 in the scheduler's timezone as the crontab owner. The script takes its configured source and destination arguments. Paths, permissions, logging and the backup program must already be configured. Standard user crontabs omit a username field; system crontabs may include one.

Data Communication and Computer Networking module

Network and System Administration module

crontab(5) manual

NSA38 | Construct a bounded backup script

Separate argument parsing, preconditions, copying/archive creation and result reporting.

Require a source directory, avoid putting the backup inside that source, create unique output names and ensure a failed partial file is not mistaken for a completed backup.

Real backup policies also need application consistency, capacity checks, retention and independent protected copies.

The following teaching script packages a directory; it does not silently install a scheduled job.

Example

Worked microexample

- import argparse, shutil from pathlib import Path from datetime import datetime, timezone p = argparse.ArgumentParser() p.add_argument("source", type=Path) p.add_argument("destination", type=Path) a = p.parse_args() src, dst = a.source.resolve(), a.destination.resolve() if not src.is_dir() or not dst.is_dir(): p.error("source and destination must be existing directories") if dst == src or src in dst.parents: p.error("destination must be outside source") name = datetime.now(timezone.utc).strftime("copy-%Y%m%dT%H%M%S%fZ") base = dst /name try: archive = shutil.make_archive(str(base), "gztar", root_dir=src) except OSError as exc: base.with_suffix(".tar.gz").unlink(missing_ok=True) p.exit(1, f"backup failed: {exc}\n") print(archive) # Run against a teaching directory, verify archive members, then restore # into a different empty directory and compare selected files.

Python tutorial, control flow

Python tutorial, input/output

Python errors and exceptions tutorial

Python shutil documentation

NSA-S01 | Operations principles versus development and delivery models

Least privilege limits an account or process to the rights its task needs. Agile development instead organizes frequent delivery and adaptation of software. Object-oriented programming organizes software around objects with state and behavior. Cloud computing supplies on-demand access to a shared pool of configurable computing resources. A development approach or hosting model does not itself define an access-rights principle.

Network and System Administration module · page 43

Agile Manifesto - Principles

Oracle - Object-oriented programming concepts: objects

NIST SP 800-145 - The NIST Definition of Cloud Computing

NSA-S02 | Disaster recovery as a planning objective

Disaster recovery planning prepares for restoring systems and data after disruptive loss. Regular protected backups and tested restoration are part of that plan. Continuing operation despite a component failure is a fault-tolerance objective; possessing extra components alone does not establish that older valid data can be recovered.

Network and System Administration module · page 30

NIST SP 800-34 Revision 1 - Contingency planning

NSA-S03 | Initialization, shutdown and filesystem-check tools

In a traditional SysV-style system, init starts and supervises configured processes; /etc/inittab is its configuration file. The halt command stops the system through the supported shutdown procedure. Neither initialization, shutdown nor their configuration file is a filesystem-consistency check; fsck supplies that separate checking and optional repair role.

Debian sysvinit manual - init

Debian sysvinit manual - halt

NSA-S04 | What an ICMP reachability probe actually tests

Ping uses ICMP echo messages to test IP reachability. A reply shows that the addressed host and path support that exchange; no reply alone does not prove the host is disconnected, because filtering or loss can suppress responses. Traceroute instead discovers responding intermediate hops through packet-lifetime expiry; it is a path-discovery diagnostic, not the same question as an end-to-end echo test.

Network and System Administration module · page 27

NSA-S05 | Distinguish scripting features by their responsibility

Exception handling transfers control from a failing operation to a matching handler. A try block contains the attempted work; an except clause handles specified raised exceptions. A loop repeats operations, a conditional chooses an ordinary branch, and a function packages reusable work; these constructs do not by themselves intercept exceptions.

A module groups definitions and statements that can be imported. A class bundles data with functionality for its objects. These organize a program, while command-line parsing interprets the arguments and flags supplied at invocation. Their implementation can use functions or classes, but organization alone does not parse arguments.

A signal handler responds when a process receives a specified asynchronous event notification. Handling such a notification is distinct from reading CPU or memory measurements through an OS interface, a performance counter or a log that actually records the metric.

Python tutorial - Errors and exceptions

Python tutorial - Modules

Python tutorial - Classes

Python library - Signal handlers for asynchronous events

Automata and Complexity Theory

Complete Study Notes

AUT01 | Symbols, alphabets, words and length

An alphabet Σ is a finite set of symbols.

A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, $L \subseteq \Sigma^*$.

An empty word ε, also written λ, has length zero.

The empty language $\varnothing$ contains no words; $\{\varepsilon\}$ contains one.

Neither ε nor $\varnothing$ is the numeric symbol 0.

Length counts occurrences, not distinct symbols: $|caddba|=6$, so $|caddba|^2=36$.

By contrast $|(caddba)^2|=12$ because the word is concatenated twice.

Distinguish $|Z|^2$, the square of a word's length, from $|Z^2|$, the length of its repeated concatenation; keep the exponent in its stated position.

Automata and Complexity Theory module

AUT02 | Concatenation, powers and reversal

Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. $\varepsilon x = x\varepsilon = x$. Define $x^0=\varepsilon$ and x^n as n copies for $n \geq 1$.

Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R=y^Rx^R$ and $(x^R)^R=x$.

For languages, $LM=\{xy : x \in L, y \in M\}$, $L^0=\{\varepsilon\}$, and $L^{n+1}=L^nL$.

Thus $\{e,fgh\}\{c,d\}=\{ec,ed,fghc,fghd\}$; fghe is absent.

A language power is a set, so $L^0=\varepsilon$ is a type error unless braces are understood.

Symbols are case-sensitive; uppercase symbols do not equal lowercase symbols unless a conversion is explicitly specified.

Automata and Complexity Theory module

AUT03 | Language operations and empty-word edge cases

Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup_{n\geq 0}\Sigma^n$ and $\Sigma^+=\bigcup_{n\geq 1}\Sigma^n$.

Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over $\{a,b\}$ is $\{aa,ab,ba,bb\}$.

For an arbitrary language L, $L^*=\bigcup_{n\geq 0}L^n$ and $L^+=LL^*$.

Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon \in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$.

Union means membership in either set; intersection requires the same word in both, not concatenation.

Complement is $\Sigma^* - L$ and needs a fixed alphabet.

Automata and Complexity Theory module

AUT04 | Grammars generate; automata recognize

Write $G=(V,\Sigma,P,S)$: finite variables V, terminals Σ disjoint from V, productions P and start variable $S \in V$. A production replaces its left side by its right side.

The language is $L(G)=\{w \in \Sigma^* : S \Rightarrow^* w\}$.

A recognizer processes a word and accepts exactly the language members.

Grammars generate languages; automata recognize languages, not grammars.

Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names.

One common word, equal alphabets or one-way inclusion is insufficient.

A regular expression denotes a language but is not itself the set of matched words.

Natural languages such as English are not the intended formal-language examples; give a precisely specified set instead.

Automata and Complexity Theory module

AUT05 | Chomsky hierarchy and the epsilon convention

Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

In an unrestricted production $\alpha \to \beta$, α contains at least one variable; unrestricted does not mean any meaningless rewrite is allowed.

Use $\text{REG} \subsetneq \text{CFL} \subsetneq \text{CSL} \subsetneq \text{DECIDABLE} \subsetneq \text{RE}$, under the usual convention allowing $S \to \varepsilon$ with S absent from right sides when needed.

Strict noncontracting grammars otherwise cannot generate ε.

Stronger families can describe lower-family languages: asking which grammar is capable is often nonunique.

Ask for the least sufficient family or the exact family equivalent to a machine model.

This is an expressiveness hierarchy, not a runtime ranking.

Automata and Complexity Theory module

AUT06 | DFA definition and acceptance

A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition.

F may be empty, recognizing $\varnothing$.

State names $q_0, q_1, \ldots$ do not determine accepting status.

Start at q_0 and consume the entire input.

Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input.

Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Automata and Complexity Theory module

AUT07 | Design and trace a finite-state recognizer

Choose states with invariant meanings: the information about the consumed prefix needed to decide future acceptance.

For binary words ending 01 use q_0=no useful suffix, q_1=last suffix 0, q_2=last suffix 01; only q_2 accepts.

Transitions on (0,1) are $q_0 \to (q_1, q_0)$, $q_1 \to (q_1, q_2)$, $q_2 \to (q_1, q_0)$.

Input 10101 follows $q_0, q_0, q_1, q_2, q_1, q_2$ and accepts.

For length at least two use meanings 0 symbols, 1 symbol, at least 2 symbols.

Every input advances toward the third state, which loops and accepts.

Calling it q_2 is a naming choice: without a diagram or state definitions no numbered-state answer follows.

To verify a design, test ε, shortest accepted/rejected words and overlaps; sample agreement alone is not a proof.

Automata and Complexity Theory module

AUT08 | Minimum states: prefix, suffix and valid alphabets

For a fixed valid word w of length n, a DFA for strings ending in w needs $n+1$ states representing the longest consumed suffix that is a prefix of w. Overlapping prefixes determine fallback transitions.

Distinct prefixes are distinguishable by suitable continuations, proving the lower bound; do not add an unnecessary rejecting sink.

For strings starting with w, a complete DFA ordinarily has n matched-prefix stages, an accepting continuation state and a rejecting sink: n+2 when n≥1 and the alphabet allows mismatches.

Over a unary alphabet the sink may be unreachable.

If w contains a symbol outside Σ, the language is empty, whose minimal complete DFA has one rejecting state.

Thus starting abc over {a,b} needs one state, not five; five requires an alphabet containing a,b,c.

Automata and Complexity Theory module

AUT09 | NFA and epsilon transitions

A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta:Q\times\Sigma\to P(Q)$.

An ε-NFA additionally permits moves without consuming input, using domain $Q\times(\Sigma\cup\{\varepsilon\})$.

Some books include ε moves in the term NFA; state the convention.

An NFA accepts iff at least one finite path consumes the complete word and finishes in F, including permitted ε moves before/after symbols.

Other paths may get stuck.

An NFA is not more expressive than a DFA: both recognize exactly REG.

Nondeterminism can make a description smaller.

A DFA is a special NFA after interpreting its unique successor as a singleton set. ε is a zero-input move, not an ordinary input character or permission to accept with unread input.

Automata and Complexity Theory module

AUT10 | Subset construction with epsilon closure

ε-closure(T) contains T and every state reachable through ε moves.

Start the DFA at ε-closure($\{q_0\}$); for each current subset T and symbol a compute ε-closure($\bigcup q\in T\,\delta(q,a)$).

A subset accepts iff it intersects F. Explore only reachable subsets; the empty subset is the rejecting sink when reached.

An n-state NFA yields at most 2^n DFA states, not necessarily exactly 2^n.

Example over $\Sigma=\{a\}$: states p,r, start p, final r; $\delta(p,a)=\{p,r\}$, $\delta(r,a)=\varnothing$.

The subset machine starts {p}, reaches {p,r} on a, and stays there on further a's. It accepts a^+.

With an ε edge p→r, the starting subset would already contain r and ε would also be accepted.

Automata and Complexity Theory module

Sipser MIT Lecture 2: nondeterminism and closure

AUT11 | Equivalence, minimization and diagram checks

First discard unreachable DFA states.

Partition remaining states into accepting/nonaccepting blocks; repeatedly split a block when two states transition on some symbol into different blocks.

Stable blocks are equivalence classes and can be merged.

Equivalent states have identical acceptance for every continuation; current acceptance alone is insufficient.

For DFA equivalence build pairs of states and search from the start pair.

Any reachable pair with exactly one accepting component supplies a distinguishing word; if none exists, the languages agree.

A partial diagram must first be completed consistently with a sink.

To reject a proposed RE quickly, compare ε and the shortest cycles: an accepting start plus an ab return cycle forces ε and ab into the language.

Every equivalent expression must contain both.

Automata and Complexity Theory module

Cornell CS412/413: DFA minimization

AUT12 | Regular-expression notation and precedence

Classical expressions use ∅, ε, symbols, union, concatenation and star.

A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +.

Parenthesize when conventions might clash.

Star binds tightest, concatenation next, union last. r? means r|ε; r^+=rr*.

(a|b)*a denotes words ending a; 0*10* denotes exactly one 1; a*(b|c) gives zero-or-more a's then exactly one b or c, so aab belongs but aaa and aaabc do not. (0|ε)(0|1)* is simply all binary words because ε permits any suffix.

Programming regex features such as backreferences may exceed classical regular-language power; wildcard dot is an extra notation, not a basic alphabet-independent theorem.

Automata and Complexity Theory module

AUT13 | Build expressions and convert representations

To require at least one a anywhere over {a,b}, use (a|b)*a(a|b)*. b*ab* requires exactly one a. At least two zeros uses (0|1)*0(0|1)*0(0|1)*; the zeros need not be adjacent.

Testing 010 distinguishes it from a requirement for substring 00.

For S→aS|ε, any number of a's is generated, giving a*.

RE→automaton uses ε-NFA fragments: parallel alternatives for union, connect accepting ends to the next fragment for concatenation, and add bypass/repetition ε moves for star.

Then apply subset construction.

For automaton→RE, eliminate a state k while updating edge i→j as $r_{ij} \mid r_{ik}(r_{kk})^*r_{kj}$; include new start/final nodes where required.

These operations preserve the whole language, not just tested examples.

Automata and Complexity Theory module

Sipser MIT Lecture 3: FA/RE and pumping

AUT14 | Regular grammars and closure constructions

Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba.

Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates a^nb^n.

A CFG rule A→aBA is legal but not regular-grammar form.

For right-linear grammar→NFA, variables become states, A→aB becomes an a edge, A→a leads to a fresh final state, and A→ε makes A accepting.

Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Reverse edges and exchange start/final roles using an ε-NFA to obtain reversal.

Complement requires a complete DFA; then swap accepting/nonaccepting states.

Product-state constructions implement intersection/difference.

Automata and Complexity Theory module

Sipser MIT Lecture 2: closure constructions

AUT15 | Regular pumping lemma: use the quantifiers

If L is regular, some p≥1 satisfies: every s∈L with $|s|\geq p$ has a split s=xyz, $|xy|\leq p$, $|y|\geq 1$, and $xy^iz\in L$ for every i≥0.

To disprove regularity, assume an arbitrary pumping length p, choose s, then show every allowed split can be broken by some i. Finding one bad split proves nothing; satisfying this necessary condition alone does not prove regularity.

For $L=\{a^nb^n : n\geq 0\}$, choose a^pb^p.

Every allowed $y=a^k$ with k≥1; pumping i=0 leaves fewer a's than b's. The contradiction excludes every DFA.

For an n-state DFA, an accepted word of length at least n repeats a state along its path, giving an accepted cycle that can be repeated indefinitely.

Thus a fixed DFA's language is infinite iff it has an accepted word of length at least its state count.

Automata and Complexity Theory module

AUT16 | CFG derivations, sentences and parse trees

In a context-free grammar (CFG) each production has one variable on the left, $A \to \alpha$ with $\alpha \in (V \cup \Sigma)^*$.

The left side therefore also belongs to $V \cup \Sigma$ and its string closure; alternatives stating these weaker memberships can all be true.

A sentential form is any string derived from S, including terminal-only sentences and S itself.

A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable.

This is not scanning the completed input in opposite directions.

A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used.

Read the leaves left-to-right for the yielded word.

The whole tree represents derivation structure; a path in a derivation graph is a different object.

Automata and Complexity Theory module

AUT17 | Design and validate context-free grammars

Use a recursive rule to enforce matched structure and a base rule to stop.

$S \to aSb \mid \varepsilon$ generates exactly $a^n b^n$ for $n \geq 0$: n recursive steps add n a's on the left and n b's on the right.

Derive aabb by $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.

For $n \geq 1$ replace ε by base ab.

For $a^n b^m$ with $n > m \geq 0$, use $S \to aSb \mid A$ and $A \to aA \mid a$: each S recursion adds a matched pair, then A adds at least one extra a. For balanced parentheses use $S \to (S)S \mid \varepsilon$.

Validate soundness-each rule preserves the intended property-and completeness-decompose any desired nonempty word into pieces matching a production.

Merely producing several examples does not establish exact language equality.

Automata and Complexity Theory module

AUT18 | Ambiguity and precedence

A CFG is ambiguous when some word has two distinct parse trees, equivalently two different leftmost derivations.

Different expansion orders that give the same tree do not prove ambiguity.

E→E+E|E*E|id allows id+id*id to group as (id+id)*id or id+(id*id).

Separate precedence levels: E→E+T|T, T→T*F|F, F→id|(E).

Multiplication binds inside T before addition at E, and left recursion encodes left association.

Uniqueness follows by the top-level last operator split and induction.

S→SS|ε is already ambiguous for ε.

S→A|B is ambiguous if A→ab and B→abB|ε both produce ab.

Ambiguous grammar and inherently ambiguous language differ: another grammar may remove ambiguity.

Automata and Complexity Theory module

AUT19 | Simplification, CNF and membership

A null production is A→ε; a unit production is A→B with one variable.

Mark nullable variables and add alternatives omitting nullable occurrences, preserving ε through a fresh start when needed.

Remove unit rules by copying nonunit productions reachable through unit chains.

Remove nongenerating variables, then unreachable symbols.

Do not delete a nullable variable's every use and lose valid nonempty words.

Chomsky normal form uses A→BC or A→a, with an optional isolated S_0→ε.

Replace terminals in long right sides by variables and split long variable sequences into binary steps: A→aBC becomes A→XC_1, X→a, C_1→BC.

For CNF membership, CYK fills variables generating each substring, combining splits with A→BC.

For a fixed grammar its time is $O(n^3)$; n>0 derivations contain n terminal rules and n−1 binary rules.

Automata and Complexity Theory module

Virginia Tech OpenDSA: transforming grammars

AUT20 | Greibach form and left recursion

Greibach normal form requires A→aA_1…A_k, k≥0: one leading terminal followed only by variables.

Handle ε separately through a fresh start that never appears on a right side.

A nonempty word of length n takes exactly n leftmost rule applications because each contributes one terminal to the prefix.

A constructive route removes ε/unit/useless rules, orders variables, substitutes rules beginning with earlier variables, removes left recursion, and substitutes until every rule begins with a terminal.

Some algorithms first convert to CNF; this is a method choice, not a universally mandatory first step.

Direct $A \to A\alpha \mid \beta$ can become $A \to \beta R$, $R \to \alpha R \mid \varepsilon$, then further normalization handles ε.

GNF cannot retain direct or indirect left recursion.

A question demanding the first step must name its chosen algorithm.

Automata and Complexity Theory module

Duke CPS140: CNF and Greibach conversion

AUT21 | PDA configuration and acceptance

A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs.

Fix which end denotes the top before tracing.

Replacing X by X leaves the stack unchanged; ε input means consume no symbol, while replacing X by ε pops it.

These are different uses of ε.

A configuration records state, unread input and stack.

Final-state acceptance requires all input consumed and a final state; empty-stack acceptance requires all input consumed and an empty stack.

For NPDAs the modes characterize the same language family, through a construction, not necessarily the same unmodified machine.

Deterministic-PDA acceptance conventions need extra care and are not automatically interchangeable.

Automata and Complexity Theory module

AUT22 | Construct PDA recognizers and compare determinism

For $a^n b^n$, push one X per a, switch to a b-reading phase, pop one X per b, and accept when input ends at the bottom marker.

Reject a after entering the b phase and b when no X remains.

Handle ε explicitly if n=0 is permitted.

A stack counts without storing the full input as random-access memory.

For even palindromes ww^R over {a,b}, push the first half, nondeterministically guess its midpoint, then pop matching symbols.

No center marker makes that language context-free but not deterministic context-free.

With a marker wcw^R the switch point is visible and a DPDA suffices.

Unary even palindromes are merely even-length words and regular.

A PDA cannot compare two arbitrary halves in the same forward order: $\{ww : w \in \{a,b\}^*\}$ is the copy language, not the palindrome language.

Automata and Complexity Theory module

Jeannette Wing, CMU: Formal methods for computer systems

AUT23 | CFL closure and counterexamples

CFLs are closed under union, concatenation, star, reversal and intersection with a regular language.

For CFL∩REG combine PDA control with DFA control, using the original stack; each consumed symbol updates both states, while an ε move leaves the DFA state unchanged.

The result is guaranteed context-free, not necessarily regular: intersect a^nb^n with a^*b^* to get a^nb^n.

CFLs are not closed under arbitrary intersection.

Both $\{a^nb^nc^m\}$ and $\{a^mb^nc^n\}$ are context-free; their intersection $a^nb^nc^n$ is not.

Nor are CFLs closed under complement or arbitrary difference.

CFL minus a regular language remains CFL by intersecting with the regular complement.

A failed construction of one proposed PDA proves only that construction fails, not that every PDA fails.

Automata and Complexity Theory module

Old Dominion University CS390: CFL properties

AUT24 | CFL pumping proof and copy versus mirror

If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \le p$, $|vy| \ge 1$ and $uv^ixy^iz \in L$ for every $i \ge 0$.

The two pieces v and y are pumped together; either may be empty, not both.

A non-CFL proof must defeat every allowed decomposition.

For $s=a^pb^pc^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count.

Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks.

Hence $a^n b^n c^n$ is not CFL.

The copy language ww is also non-CFL over two symbols, whereas ww^R has the explicit grammar $S \to aSa \mid bSb \mid \varepsilon$.

An expression that writes ww and calls it a palindrome is contradictory.

Automata and Complexity Theory module

AUT25 | Context-sensitive rules and linear bounded automata

A context-sensitive rule $\alpha A \beta \to \alpha \gamma \beta$ replaces A in context, with γ nonempty.

Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception.

The condition is on the complete sides, not an unrelated comparison $|\beta| \geq |\alpha|$.

A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs.

For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

All marks stay within the input region.

A fixed input has finitely many configurations, at most $|Q| m |\Gamma|^m$ for m cells.

Reachability search in that finite graph decides acceptance, even if some machine branches loop.

Thus every CSL is decidable.

Automata and Complexity Theory module

Illinois CS373: unrestricted grammars and LBAs

AUT26 | Classify languages with justified boundaries

a*b* and $\{a^n b : n \geq 0\}$ are regular. $a^n b^n$ is CFL but not regular.

Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^n b^n c^n$ is CSL but not CFL, and therefore decidable.

Every finite language is regular.

A stronger grammar or TM can still recognize any of these lower-family languages.

Unary prime length $\{a^p : p \text{ prime}\}$ is nonregular: choose prime q above pumping length p; if pump piece has length $k \geq 1$, using exponent q+1 gives length q(k+1), composite.

Every unary CFL is regular, so this language is not CFL either.

It is nevertheless context-sensitive: count the input in bounded workspace and test divisors.

The balanced-three-count, prime-length, palindrome and a^nb examples above are all recursive languages.

Automata and Complexity Theory module

UMass, David Barrington: unary context-free languages

AUT27 | TM components, transitions and configurations

For a Turing machine (TM), use $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with finite state/tape alphabets, $\Sigma\subseteq\Gamma$ and blank $\square\in\Gamma-\Sigma$.

The blank symbol belongs to the tape alphabet Γ, not the input alphabet Σ.

The standard machine has finite control, a read/write head and an unbounded tape.

Input is initial tape content, not a separate hardware organ; no separate stack is required.

$\delta(q,a)=(p,b,L)$ means read a in q, write b, move one cell left and enter p. Accept/reject halting states have no further moves under this convention.

In ID u q av the head scans a; the left-move successor, when u=u'c, is u' p cbv.

For a right move it is ub p v, showing a blank if the head enters unused tape.

State and transition names must follow the diagram's declared convention.

Automata and Complexity Theory module

AUT28 | TM design, variants and computation limits

To decide $a^nb^nc^n$, repeatedly mark matching triples, resetting the head between scans, then verify no unmatched symbol remains.

Each round marks additional input cells, so finitely many rounds suffice.

To decide ww, reject odd length, locate the midpoint using marks/counters, and compare corresponding symbols of the two halves; writable tape permits access a single stack lacks.

State both the acceptance test and why every input halts.

An off-line TM has a read-only input tape and separate read/write work tape.

Multitape, multihead, two-way-infinite-tape and nondeterministic TMs have the same language-recognition power as standard TMs, though simulation costs differ.

Two unbounded stacks can simulate a tape.

Turing completeness means ability to simulate general TM computation, not polynomial speed or literally using infinitely many cells in a finite run.

The Church-Turing thesis connects effective procedures to this formal model; it is not a proved runtime theorem.

Automata and Complexity Theory module

AUT29 | Deciders, recognizers and complements

A decider halts on every input and accepts exactly its language; these languages are recursive/decidable.

A recognizer must halt and accept every member, but on a nonmember may halt-reject or loop.

Its languages are recursively enumerable (RE), Turing-recognizable or semidecidable.

Looping is nonacceptance, not a rejecting halt.

An undecidable language may still have a recognizer.

Every decidable language is RE, and the containment is strict.

If L and its complement are both RE, simulate their recognizers in dovetailed steps until one accepts, obtaining a decider.

Conversely complementing a decider works because it always halts.

Flipping final states in an arbitrary recognizer fails to handle nonmembers that loop.

Recursive function theory's total computable functions are not merely programs written with recursive self-calls; partial computable functions may be undefined where computation does not halt.

Automata and Complexity Theory module

AUT30 | Halting, diagonalization and Rice’s theorem

HALT={⟨M,w⟩:M halts on w} is recognizable: simulate and accept upon any halt.

Suppose a decider H determines halting.

Construct D(x) that loops if H(x,x) says halt, otherwise halts.

Running D on its own description contradicts H in both cases.

Thus HALT is undecidable and witnesses RE beyond DECIDABLE.

Its complement is not RE, or both recognizers would yield a decider.

Rice's theorem makes every nontrivial semantic property of a TM's recognized language undecidable.

Nontrivial means some recognized languages have it and some do not.

Empty language, contains ε and at least three accepted words qualify.

Number of states and whether a particular symbol is written are machine-behavior/syntactic properties, not automatically language properties; prove such claims separately.

Undecidable means no general decider exists, not merely no algorithm is currently known.

Automata and Complexity Theory module

AUT31 | Which decision questions are solvable?

DFA/NFA word membership is decidable by simulation or reachable state sets.

FA emptiness is reachable-accepting-state search; DFA equivalence searches the product for an acceptance disagreement.

CFG membership is decidable by parsing/CYK and emptiness by marking variables that can generate terminal strings.

These procedures halt and are not the same as deciding arbitrary TM membership.

TM acceptance on input is RE-complete and undecidable; HALT is undecidable.

General CFG ambiguity and CFG equivalence are undecidable even though membership for a given CFG and word is decidable.

A concrete grammar's two parse trees can establish that particular grammar is ambiguous; this does not give an algorithm deciding ambiguity for every grammar.

FORTRAN/programming-language questions must distinguish CFG-style syntax, semantic restrictions and Turing-complete computation; a language's name alone does not choose one formal classification.

Automata and Complexity Theory module

Sipser MIT Lecture 7: decision problems

AUT32 | Computable reductions for undecidability

A many-one reduction $A \leq_m B$ is a total computable transformation f satisfying $x \in A$ iff $f(x) \in B$. If B had a decider, compute f and invoke it to decide A. Therefore known-undecidable A reduced to B proves B undecidable.

Showing $B \leq_m A$ alone does not establish B's hardness.

Example $HALT \leq_m A_{TM}$: given $\langle M,w \rangle$, construct N that ignores its input, simulates M(w), and accepts if M ever halts, including rejecting halts.

Then $\langle N,\varepsilon \rangle$ is accepted iff M(w) halts.

Construction of N's finite description terminates; the simulation inside N may not.

Reductions require both yes and no directions and a computable mapping, not just a resemblance between two problems.

Automata and Complexity Theory module

Sipser MIT Lecture 9: reducibility

AUT33 | Measure resources with the right input size

Time counts computation steps; space counts cells/memory used, specifying whether read-only input is excluded.

For a deterministic decider use the maximum over all length-n inputs; for a time-bounded nondeterministic decider every branch meets the bound.

Big-O is an asymptotic upper bound, not inherently a worst-case label: it can describe a worst-, average- or best-case function. Θ adds a matching lower bound.

Growth typically orders $1, \log n, n, n \log n, n^2, 2^n, n!$.

A fixed polynomial n^k differs from exponent 2^n.

Encoding matters: the number N in binary has Θ(log N) bits, so O(N) work is not polynomial in its binary input length.

Pseudopolynomial algorithms depend on numeric magnitude rather than encoding length.

A polynomial of enormous degree need not be practical.

Automata and Complexity Theory module

AUT34 | P, NP and polynomial verification

Complexity classes here contain decision languages.

P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates.

A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance.

NP means nondeterministic polynomial, not non-polynomial.

For CLIQUE, the certificate lists k vertices; verify they are distinct and every pair is an edge.

This is polynomial in the graph encoding.

$P \subseteq NP$ because a verifier can ignore its certificate and run the P algorithm.

P=NP remains open; NP does not mean definitely hard or definitely outside P. Finding versus verifying is an intuition for witnesses, not a rule that all NP instances need exponential search.

Automata and Complexity Theory module

Sipser MIT Lecture 14: P, NP, certificates

Clay Mathematics Institute: P versus NP

AUT35 | Polynomial reductions: direction and consequences

$A \leq_p B$ means f is computable in polynomial time and $x \in A$ iff $f(x) \in B$. Solving B then solves A, so B is at least as hard under this reduction.

If $B \in P$ then $A \in P$; if A is NP-hard then B is NP-hard.

$A \in P$ does not force $B \in P$, and the reduction alone does not place both problems in the same class.

Polynomial reductions compose.

To prove a new target X hard, reduce a known hard problem Y to X, not X to Y. The mapping and target instance must have polynomial size.

For undecidability, general computable reductions suffice; for NP-completeness, an arbitrarily slow computable transformation is insufficient.

Always state which reduction and resource bound are being used.

Automata and Complexity Theory module

AUT36 | NP-hard versus NP-complete; decision versus optimization

X is NP-hard if every $A \in NP$ satisfies $A \leq_p X$.

It need not belong to NP and can even be undecidable.

X is NP-complete if $X \in NP$ and X is NP-hard.

NP-complete languages lie inside NP and can be decided by finite exhaustive certificate search in exponential time.

They are not known to require exponential time.

SAT asks whether a Boolean formula has a satisfying assignment; Cook-Levin establishes SAT NP-complete.

CNF-SAT and 3-SAT are NP-complete versions.

TSP decision asks whether a tour of cost at most B exists; it is NP-complete for standard finite encoded inputs.

Finding the shortest tour is optimization and NP-hard; call it NP-complete only after specifying a decision formulation.

Fixed 2-colorability is in P, while fixed k-colorability for $k \geq 3$ is NP-complete.

Automata and Complexity Theory module

AUT37 | Work a hardness proof: 3-SAT to CLIQUE

First CLIQUE belongs to NP by the pairwise-edge certificate check.

For a 3-CNF formula with m clauses, make one vertex for each literal occurrence.

Connect two vertices iff they come from different clauses and are not complementary literals.

Ask whether the graph has a clique of size m. Construction takes polynomial time and space.

A satisfying assignment supplies one true literal per clause; they are mutually consistent and form a clique.

Conversely an m-clique selects one occurrence per clause, with no variable chosen both positively and negatively, so assign those literals true and extend remaining variables arbitrarily.

Both implications prove equivalence.

For $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_3$, the assignment (true,true,false) satisfies all clauses because ¬false is true.

Automata and Complexity Theory module

Sipser MIT Lecture 15: NP-completeness

AUT38 | PSPACE, quantifiers and space reuse

PSPACE is $\bigcup_k \text{SPACE}(n^k)$, using deterministic polynomial workspace.

NP⊆PSPACE because candidate polynomial witnesses can be enumerated and checked one at a time, reusing memory; exponentially many candidates do not require retaining them all.

Savitch's theorem gives $\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2)$ for appropriate $s \geq \log n$, hence NPSPACE=PSPACE.

It does not prove P=NP.

TQBF asks whether a fully quantified Boolean formula is true and is PSPACE-complete.

Evaluate ∃x by trying one branch or the other; evaluate ∀x by requiring both.

Depth-first evaluation stores assignments and recursion state using polynomial space despite potentially exponential time.

For example $\forall x \exists y (x \leftrightarrow y)$ is true, but $\exists y \forall x (x \leftrightarrow y)$ is false.

PSPACE-complete means membership plus hardness for every PSPACE language under the stated reduction.

Automata and Complexity Theory module

Sipser MIT Lecture 17: PSPACE and Savitch

AUT39 | Known hierarchy and open separations

Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE.

Here L/NL use logarithmic workspace; EXPTIME is $\bigcup_k \text{TIME}(2^{(n^k)})$, an upper-bound class that also contains P, not a statement that every member requires exponential time.

Suitable hierarchy theorems prove P⊊EXPTIME and PSPACE⊊EXPSPACE.

They do not establish every adjacent containment strictly.

P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; P⊆NP∩coNP, and NP=coNP is open.

If any NP-complete language is in P, all NP collapses to P. This does not make every decidable language polynomial or make undecidable problems decidable.

Avoid blanket claims about all cryptography or all problems becoming easy; the class and encoding matter.

Automata and Complexity Theory module

Sipser MIT Lecture 21: hierarchy theorems

AUT40 | Derive languages from grammar productions

For S→11S|0A|1A and A→0|1|0S, every reachable form has at most one variable, at the right end.

A complete terminal word is still a sentential form.

For example S⇒11S⇒1111S⇒111111S⇒1111110A⇒11111101.

Returning to S appends a pair 11,00 or 10; use that invariant to rule out impossible prefixes rather than guessing from string length.

For A→aB|bA|bBa and B→bB|cA|b, specify a start variable first.

With A assumed, a terminating membership test recursively removes the required a/b prefix or b…a wrapper; B either matches b or removes b before B/c before A. Every recursive residual is shorter.

Apply the membership test to every candidate word; none qualifies in this four-word example under the assumed start A.

Use the membership test to justify whether a candidate word belongs to the language.

Automata and Complexity Theory module

AUT41 | Specify languages, states and start symbols

Use the exact readable language formula and alphabet when testing membership.

State labels are arbitrary: use the transition diagram, initial state and accepting markings, or a definition assigning meanings to states.

Language theory alone cannot identify an arbitrary state label.

An accepting start state and an ab return cycle require a corresponding regular expression to include both ε and ab.

Declare the start variable before testing grammar membership; with A assumed in the grammar above, none of the four candidate words qualifies.

Distinguish the non-CFL copy language ww from the even-palindrome language ww^R.

Generic capable, contains and most difficult questions need a least-family, exact-class or reduction qualifier.

For GNF first-step questions identify the conversion algorithm.

Evaluate each description against the stated language and machine conventions.

Automata and Complexity Theory module

AUT-S01 | Computability, complexity and abstract machines

Computability theory asks whether an effective algorithm can solve a problem at all. For a decision problem, a decision procedure must terminate with the correct yes-or-no result on every input.

Computational complexity theory studies the time and memory resources required to solve problems as functions of encoded input size. Automata theory studies abstract machine models and the languages they recognize; these subjects overlap but ask different questions.

Automata and Complexity Theory module · page 15

Automata and Complexity Theory module · page 2

Automata and Complexity Theory module · page 16

AUT-S02 | The stack discipline and its queue contrast

A stack is last-in, first-out: push adds an item at the top and pop removes the top item, so the most recently pushed item is removed first. A PDA accesses this top rather than an arbitrary interior position.

A queue is first-in, first-out: insertion at the rear and removal at the front returns the oldest waiting item. A linked list can implement a storage discipline, but the implementation container is different from the abstract push/pop or enqueue/dequeue access rule.

Data Structure and Algorithms module · page 6

Data Structure and Algorithms module · page 7

Oracle Java SE 17: Deque stack and queue semantics

AUT-S03 | Why satisfiability was the first NP-complete problem

Boolean satisfiability was the first problem proved NP-complete, through the Cook-Levin theorem. Its hardness proof starts directly from an arbitrary nondeterministic polynomial-time computation and constructs a Boolean formula that is satisfiable exactly when an accepting computation exists.

This starting point matters: once one NP-complete problem is available, later hardness proofs can reduce that known problem to a new target. Membership in NP remains a separate obligation. Checking a satisfying assignment is efficient; whether finding one is always polynomial-time remains unresolved.

University of Toronto CSC373: Complexity, slide 28

AUT-S04 | Finite procedures are different from undecidable tasks

Sorting a finite numeric list is a terminating computation. Quicksort, for example, chooses a pivot, partitions the other entries and recursively sorts the smaller sublists. Each recursive subproblem has fewer entries. A large running time does not make the task undecidable.

A finite linear system with exactly represented rational coefficients can be processed by Gaussian elimination and back substitution. Row exchanges handle unsuitable pivots; row reduction detects inconsistency or leaves free variables describing multiple solutions. These finite algebraic procedures differ from a general halting decider, which does not exist.

Data Structure and Algorithms module · page 11

MIT 18.06: Gaussian elimination

MIT OpenCourseWare - Solving Ax = b: row reduced form R

Compiler Design

Complete Study Notes

COMP01 | Translation, interpretation and compilation time

A compiler translates a source program into an equivalent target representation while preserving defined observable behavior.

The target may be machine code, assembly, bytecode or another source language.

A transpiler is a source-to-source compiler, not a fundamentally separate noncompiler technology.

A compiler preserves language semantics, not the programmer's unstated intention.

An interpreter executes a program representation; it need not work one source line at a time.

Systems can compile to bytecode and then interpret or JIT-compile it.

AOT compilation happens before execution; JIT compilation happens during execution and may use runtime profiles.

Neither is universally faster; startup cost, optimization and workload matter.

Compiler Design module

COMP02 | Tools, host, target and incremental work

A preprocessor handles directives such as file inclusion and macros.

An assembler translates assembly to object code.

A linker combines object files and libraries, resolves external symbols and performs relocation.

A loader maps an executable into memory and starts it.

Compilation alone does not necessarily produce a ready executable.

A cross-compiler runs on a host platform and generates code for a different target.

Multiple targets require appropriate backends, not a special definition of cross-compilation.

A retargetable compiler supports changing target descriptions.

Incremental compilation recompiles affected fragments or dependent units after edits; it is more than literally replacing changed lines.

One-pass versus multipass counts traversals, not supported targets.

Compiler Design module

COMP03 | Compiler phases and shared services

The conventional sequence is lexical analysis → syntax analysis → semantic analysis → intermediate-code generation → optimization → target-code generation.

Scanning turns characters into tokens; parsing builds structure; semantics checks declarations and types; synthesis produces implementation code.

A phase is a logical activity; a pass is a traversal that can combine activities.

The front end handles source-language rules; a middle end analyzes and transforms IR; the backend handles target details.

Symbol-table management and error reporting support multiple phases rather than forming one mandatory seventh final phase.

Optimization can occur at several levels, including after instruction selection. “Final phase” means code generation in the standard six-phase teaching pipeline.

Compiler Design module

COMP04 | Token, lexeme and pattern

A token is a category with optional attribute, such as (ID, symbol-entry) or (INT_LITERAL,42).

A lexeme is the actual character sequence, such as total or 42.

A pattern specifies which lexemes belong to a token class.

The scanner emits an ordered token stream, not an unordered mathematical set.

Keywords, identifiers, literals, operators and punctuation are token classes.

Whitespace and comments are usually skipped while line/column positions are retained.

In indentation-sensitive or newline-sensitive languages, layout can carry tokens.

Parse-tree construction is a parser task; scanning can intern identifier spellings, but declaration scope and types need later analysis.

Example

Scan a statement

- int x = 42; gives keyword(int), ID(x), assignment(=), integer(42), semicolon(;): five tokens.

Compiler Design module

COMP05 | Regular expressions for lexical structure

Regular expressions use concatenation, alternation r|s, Kleene star r* for zero or more, and r+ for one or more.

In a simplified alphabet, identifier = Letter(Letter|Digit)*; it permits a single letter and forbids a leading digit.

A C-like ASCII rule commonly also allows underscore: [A-Za-z_][A-Za-z0-9_]* State the convention instead of claiming this covers every language's identifiers.

Letter(Letter+|Digit*)* denotes the same language as Letter(Letter|Digit)* under ordinary regular-expression semantics: repeated blocks can contain any trailing letter/digit sequence.

These two regular expressions are equivalent.

Regular patterns handle ordinary tokens; arbitrary balanced nesting generally needs grammar/stack structure.

Compiler Design module

COMP06 | Construct and run a scanner automaton

A common generator pipeline is regular expressions → NFA using Thompson construction → DFA using subset construction; minimization is optional.

For concatenation rs, connect r's final state to s's start by ε.

Alternation introduces branching ε paths; star adds a way to skip and repeat the fragment.

The ε-closure of states includes those reachable by zero or more ε edges.

A DFA state is a set of NFA states; its a-transition is ε-closure(move(set,a)).

Start with the NFA start closure, repeatedly add unseen subsets, and mark subsets containing an NFA final state as accepting.

Token identity and rule priority annotate accepting states; the empty subset acts as a dead state.

Compiler Design module

COMP07 | Longest match and counting tokens

At each input position, choose the longest matching lexeme; if equal lengths match multiple flex rules, choose the earliest rule.

Thus ifx is an identifier rather than if followed by x; <= and ++ are each one operator token.

An implementation may recognize an identifier first and then consult a keyword table.

For the ordinary C++ header:
for (int i = 0; i <= 10; i++)
The tokens, split into two display lines, are:
for, (, int, i, =, 0, ;
i, <=, 10, ;, i, ++,)
There are 14 tokens, irrespective of spacing.

The increment operator is one token.

A comment contributes no ordinary parser token after preprocessing.

For the loop header above, 4,5,9 and 10 are incorrect token counts; the count is 14.

Compiler Design module

Flex manual: Matching

COMP08 | Locate the error by evidence

A lexical error prevents forming a legal token, such as an invalid character or unterminated literal.

A syntax error is an illegal token arrangement, such as a missing delimiter.

A semantic error violates name, scope or type rules after grammatical recognition.

A wrong algorithm can compile successfully; compilers do not detect all logical mistakes.

In the malformed C line fro ((I = o, I;, fro and lowercase o are legal identifier lexemes.

Misspelling for does not turn fro into a lexical error; mismatched syntax is definite, and undeclared o can also produce a semantic diagnostic.

Recovery order and exact diagnostics depend on the compiler.

Do not silently replace o by 0 or fro by for.

Compiler Design module

COMP09 | CFGs, derivations and trees

A context-free grammar G=(N,T,P,S) has nonterminals N, terminals T, productions P and start S. A production A→α rewrites one nonterminal.

A sentential form is any string of terminals/nonterminals derivable from S; a sentence contains only terminals.

A leftmost derivation rewrites the leftmost remaining nonterminal; a rightmost one rewrites the rightmost.

A parse tree shows grammar productions: root S, internal nonterminals, and terminal leaves, with ε sometimes shown for empty productions.

Its left-to-right yield is the input.

An AST omits grammar scaffolding and punctuation; expression operands are leaves and operators internal nodes.

The parser can build either representation, depending on implementation.

Compiler Design module

COMP10 | Ambiguity, precedence and associativity

A grammar is ambiguous if at least one terminal string has two different parse trees, equivalently two leftmost derivations.

Merely changing derivation order on the same tree does not establish ambiguity.

E→E+E|E*E|id allows different trees for id+id*id.

Use E→E+T|T, T→T*F|F, F→(E)|id to give * higher precedence and both operators left associativity.

For subtraction, a−b−c then groups (a−b)−c. The dangling-else grammar permits else to attach to either nested if; a nearest-unmatched-if rule or grammar rewrite resolves it.

Taking a leftmost derivation by itself does not resolve ambiguous interpretation.

Compiler Design module

COMP11 | Top-down parsing

Top-down parsing starts with S and expands productions toward the input, usually constructing a leftmost derivation.

Recursive descent typically has one procedure per nonterminal.

A general recursive-descent implementation can backtrack; a predictive parser chooses a production using lookahead without backtracking.

LL(1) scans Left to right, produces a Leftmost derivation and uses one lookahead token.

A nonrecursive predictive parser stores grammar symbols on an explicit stack and consults a table.

It is another implementation of predictive parsing, not a wholly different language class.

Table-driven and recursive predictive implementations can both be linear for a fixed suitable grammar.

Efficiency cannot be ranked from their names alone.

Compiler Design module

COMP12 | Remove left recursion and factor prefixes

Immediate left recursion has A→Aα.

For A→Aα|β, where β does not start with A, introduce a new nonterminal A′ and use: A→βA′ A′→αA′|ε This preserves the generated language; tree shape changes, so semantic actions must preserve intended associativity.

Indirect left recursion follows a cycle through other nonterminals and also needs removal for ordinary predictive descent.

Left factoring rewrites A→αβ|αγ using:
A→αA′
A′→β|γ
It postpones a choice until after the common prefix.

It does not by itself remove ambiguity or guarantee LL(1).

LR parsers can handle suitable left-recursive grammars; ordinary unmodified recursive descent can recurse forever before consuming input.

Compiler Design module

COMP13 | Compute FIRST and FOLLOW

FIRST(α) contains terminals that can begin strings derived from α, plus ε when α can vanish.

For a sequence, add FIRST of each symbol except ε, continue while symbols are nullable, and add ε only if every symbol is nullable.

FOLLOW(A) contains possible next terminals after A, plus end marker $ where appropriate; never ε.

Put $ in FOLLOW(S).

For B→αAβ, add FIRST(β)−{ε} to FOLLOW(A); if β is nullable, add FOLLOW(B).

Repeat all rules until no sets grow.

Example

Small grammar

- S→AB, A→a|ε, B→b gives FIRST(A)={a,ε}, FIRST(S)={a,b}, FOLLOW(A)={b}, FOLLOW(B)={$}.

Compiler Design module

COMP14 | Build and use an LL(1) table

For every A→α, put it in M[A,a] for each terminal a in FIRST(α).

If α is nullable, also enter it under every symbol in FOLLOW(A).

More than one production in a cell is a conflict.

Pairwise SELECT sets must be disjoint; SELECT(α)=FIRST(α)−{ε}, plus FOLLOW(A) when α is nullable.

This also detects two nullable alternatives.

Initialize stack with $S. Match and pop terminals; for a nonterminal consult its row and lookahead, then push the chosen RHS in reverse.

An ε production consumes no input.

Accept when stack and input reach $.

For A→a|ε with b∈FOLLOW(A), choose a on lookahead a and ε on b.

Exam tip

Exam trap

- Ambiguous useful grammars are not LL(1). Check overlapping FIRST sets, FIRST/FOLLOW conflicts and left recursion as well.

Compiler Design module

Allan Gottlieb, NYU Compilers Lecture 6

COMP15 | Bottom-up actions and handles

Bottom-up parsing starts from input and reduces toward S, constructing a rightmost derivation in reverse.

A handle is a production RHS occurrence whose reduction is the next step of that reverse derivation.

Not every substring matching a RHS is a handle in the current context.

Shift pushes the next input token and a state; reduce replaces a recognized RHS by its LHS and consults GOTO; accept recognizes completion; error reports an invalid situation.

For A→XYZ, pop three grammar symbols and their states, expose state q, then push A and GOTO[q,A]. “Go” is not a separate standard shift-reduce ACTION, although GOTO is essential table machinery.

Compiler Design module

COMP16 | Construct LR(0) item states

Augment the grammar with S′→S. An LR(0) item A→α·β records progress through a production.

CLOSURE(I) repeatedly adds B→·γ whenever an item has its dot before nonterminal B and B→γ is a production.

GOTO(I,X) moves the dot past X in every eligible item and takes closure.

Start with CLOSURE({S′→·S}); repeatedly compute nonempty GOTO sets for all grammar symbols until no new state appears.

The resulting set of item sets is the canonical collection, not GOTO itself.

Example

One transition

- For S→a, call the initial closure I0: I0 = {S′→·S, S→·a} GOTO(I0,a) = {S→a·} GOTO(I0,S) = {S′→S·} Each complete set above is one state, not one state per item.

Allan Gottlieb, NYU Compilers Lecture 7

COMP17 | LR families and conflicts

SLR(1) uses LR(0) states and FOLLOW(A) for reduction by a completed A production.

Canonical LR(1) attaches context-specific lookahead to items.

LALR(1) merges canonical states with the same LR(0) core, combining lookaheads; merging can introduce reduce/reduce conflicts.

The standard grammar-class hierarchy is LR(0)⊆SLR(1)⊆LALR(1)⊆LR(1), with increasing capability and often table size.

A shift/reduce conflict offers both consuming a token and reducing; reduce/reduce offers two reductions.

An SLR or LALR conflict does not by itself prove the grammar ambiguous.

Canonical LR(1) handles LR(1) grammars, not every unambiguous CFG.

Precedence and associativity can resolve suitable expression conflicts; changing parser names alone does not repair all ambiguity.

Compiler Design module

GNU Bison manual

COMP18 | Parser limits and operator grammars

LL, LR and recursive-descent parsers can all handle nested parentheses for suitable grammars.

An explicit or implicit stack, not the LR label alone, supports unbounded nesting.

Both LL and LR can use parsing tables.

Deterministic LL/LR grammars must be unambiguous; a general ambiguous grammar requires rewriting, disambiguation policy or a generalized method such as GLR.

An operator grammar conventionally forbids adjacent nonterminals on a RHS and usually ε productions.

Operator-precedence parsing additionally requires conflict-free precedence relations.

Ordinary right-linear regular grammars also lack adjacent nonterminals, so that property alone does not uniquely identify the operator-grammar category.

Regular patterns are chiefly lexical specifications; CFGs chiefly describe nested syntax.

Compiler Design module

COMP19 | Symbol tables and scope

A symbol table maps names or symbol identities to attributes: type, scope, width, storage offset, parameter signature, constant information or linkage.

A hash table gives expected O(1) insertion/lookup with suitable hashing; a stack of scope tables supports lexical nesting.

Search the nearest scope first, insert declarations in the current scope, and remove visibility when leaving it.

Scanning may intern a spelling first; semantic analysis determines the declaration and type.

Implementations differ, so first interaction is conventional rather than universal.

Not every punctuation token needs a symbol-table entry; literals may use a separate literal table.

A hash table is an implementation, not the information's semantic role.

Compiler Design module

COMP20 | Names, types and advanced language features

Distinguish lexical recognition, grammatical structure and semantic constraints.

In obj.method(x), the scanner recognizes identifiers and punctuation; the parser recognizes a member call; semantic analysis resolves the receiver, accessible member and argument types.

Overload resolution chooses among signatures using static types; overriding can select an implementation at runtime.

Nested scopes require declaration identity rather than spelling alone: an inner int x can shadow an outer float x. Closures capture free variables from lexical environments; lifetime can outlive the creating stack frame.

Generic functions need parameter constraints or instantiation checks.

These features require symbol/type information beyond a CFG, even when their surface syntax is context-free.

Example

Specify and check a member call

- For object.resize(3), the token kinds are id, dot, id, left parenthesis, integer, right parenthesis. One small grammar is Call → Expr . id (Args); Expr → id; Args → Expr | integer | ε. Derive Call ⇒ Expr . id (Args) ⇒ id . id (integer), then bind the tokens object, resize and 3. The parser checks this shape. Name and type analysis then require object to be in scope, resize to be an accessible member and an overload accepting one integer-compatible argument. If only resize() exists, the text still parses but fails the argument/type check.

Compiler Design module

COMP21 | Semantic checking: static and dynamic

Static checks occur before execution: type compatibility, duplicate declarations within a restricted scope, valid names, argument counts, return compatibility and control-flow placement.

A break is valid only in constructs allowed by the language; in C/C++ it can exit a loop or switch.

A compiler may annotate an AST or create typed IR; there is no mandatory uniquely named output tree.

Dynamic checks depend on runtime values, such as some bounds and null checks.

Static analysis can prove some accesses safe or faulty; "all memory errors cannot be statically checked" is too strong.

Unreachable code is not universally a semantic error: some languages reject it in specified cases, others merely warn or optimize it away.

Compiler Design module

COMP22 | Type rules, inference and compatibility

A type checker verifies that operations are defined for their operands.

In a simple strict language, integer mod requires two integers; array indexing requires an array and integer index; an if condition requires Boolean; assignment requires equal or convertible types.

Function calls check parameter count and compatibility, and returns match result types.

Type inference derives types from constraints: if x is integer and + expects numeric operands, infer the expression type from numeric promotion rules.

It is a static technique even without explicit annotations.

Operator overloading supplies multiple operation meanings, not a substitute for type checking. int x="hello" is a type mismatch in ordinary C++/Java; division by zero and bounds violations are not automatically operand-type mismatches.

Compiler Design module

COMP23 | Conversions need a named language

Coercion is an implicit conversion permitted by the language; a cast requests an explicit conversion.

Widening often increases representable range, but can still lose precision: Java int→float is widening although large integers round.

Java int→long is exact.

Narrowing can lose range or fractional information.

In C++ ordinary int x=3.14 converts to 3 when representable; it is not inherently a type error. int x{3.14} is rejected as narrowing.

C++ floating-to-integer conversion discards the fraction and has undefined behavior if the truncated value is unrepresentable.

Java ordinary double→int assignment needs a permitted explicit cast.

Mixing char and int is ordinarily valid after promotion in C++/Java; do not import a fictitious universal ban.

C++ working draft, floating-integral conversions

Java Language Specification, conversions

COMP24 | Attributes and syntax-directed definitions

A syntax-directed definition is a CFG with attributes on grammar symbols and semantic rules on productions.

An annotated parse tree displays computed attribute values; it is not itself an inherited or synthesized attribute.

Synthesized attributes flow from children to a parent; inherited attributes carry context from parent or siblings.

For E→E1+T, E.val=E1.val+T.val synthesizes an expression value.

A declaration's type environment passed into its body is inherited.

S-attributed definitions use only synthesized attributes and fit LR reduction-time evaluation naturally; suitable LL parsers can also compute them.

L-attributed definitions restrict dependencies so a left-to-right traversal can evaluate inherited context.

Attribute dependency cycles can prevent evaluation.

Compiler Design module

COMP25 | Translation actions, ASTs and postfix

A syntax-directed translation scheme embeds executable actions in productions; the actions run while translating a source program, not while generating the compiler.

Attributes may compute types, values, AST pointers or generated code.

The scheme defines an input-to-output translation, not merely an input language.

Create an identifier leaf with mkleaf(id,entry), a number with mkleaf(num,value), and an operator node with mknode(op,left,right).

For a−4+c, construct +(−(a,4),c).

Postorder traversal prints postfix: a 4 − c +.

A stack evaluator pushes operands and pops right then left operands for an operator.

Infix-to-postfix is useful for stack machines but not mandatory for all compilers.

Compiler Design module

COMP26 | Intermediate representations and three-address code

IR separates source analysis from target implementation, allowing optimizers and backends to be reused.

It should be easy to generate, analyze, transform and lower.

Portability is an architectural benefit, not proof that every IR omits all target-dependent details.

ASTs, graphs, bytecode and TAC can all be IRs.

Ordinary TAC arithmetic has at most one operation, two sources and one destination: t1=c*d; t2=b+t1; a=t2.

Quadruples store (op,arg1,arg2,result); triples omit an explicit result field and refer to instruction positions.

Indirect triples reference a separate ordering list, easing rearrangement.

Trees describe expressions; DAGs share common subexpressions, while quadruples/triples are instruction-record structures.

Compiler Design module

COMP27 | Control flow, arrays and SSA

A conditional expression can generate if a<b goto Ltrue; goto Lfalse; labels join control paths.

Short-circuit AND evaluates the right operand only after a true left operand; OR only after a false one.

Translating both eagerly can change side effects.

For a zero-based array with element width w and lower bound L, element i uses base+(i−L)*w, assuming fixed-width contiguous storage.

SSA gives each definition a distinct name.

At a join, x3=φ(x1 from B1,x2 from B2) chooses the incoming value according to the predecessor actually taken; it is not an arithmetic average.

SSA is a naming discipline over IR and commonly coexists with a control-flow graph.

Compiler Design module

LLVM Language Reference, phi instruction

COMP28 | Basic blocks, flow graphs and dominators

A basic block is a maximal straight-line instruction sequence entered at its beginning and exited at its end.

Leaders are the first instruction, every jump target, and each instruction after a branch.

Start a block at each leader and continue until the next leader.

A control-flow graph has blocks as vertices and possible next-block transfers as edges, including fall-through.

A node D dominates N when every entry-to-N path passes through D. For an edge N→H where H dominates N, H is a loop header of a natural loop.

This structure supports loop analysis.

An expression DAG records value dependencies, not arbitrary control flow.

Compiler Design module

COMP29 | Data-flow questions and equations

Reaching definitions is forward may-analysis: which assignments can reach this point without a replacement definition?

OUT=GEN∪(IN−KILL), with IN the union of predecessor OUT sets.

Available expressions is usually forward must-analysis: an expression has been computed on every relevant path and its operands have not changed.

Join with intersection.

Liveness is backward may-analysis: a value might be read before redefinition on some future path.

OUT[B]=union of IN over successors; IN[B]=USE[B]∪(OUT[B]−DEF[B]).

Iterate to a fixed point.

Liveness supports register allocation and dead-value removal.

Definite-assignment analysis asks whether initialization occurs on every path before use; reaching definitions alone does not prove this.

Compiler Design module

COMP30 | Folding, propagation and dead code

Constant folding evaluates constant expressions during compilation: 3*4→12.

Constant propagation substitutes known values: x=3; y=x+2 becomes y=3+2 before folding.

Copy propagation substitutes a copied name while its source remains unchanged.

Dead-code elimination removes unreachable code or computations whose values are unused and whose evaluation has no necessary side effects.

For const int x=8,y=16; if(x>y){int z=x+y;}, propagate the constants, fold 8>16 to false, then remove the unreachable branch.

Both compile-time evaluation and dead-code elimination apply.

Compare optimizations by their applicability and effect; several transformations may be appropriate.

Computing an unused volatile read or effectful call may still be required.

Compiler Design module

COMP31 | Reuse, strength reduction and peepholes

Common-subexpression elimination reuses an earlier value only if operand values and relevant memory remain unchanged.

A DAG can represent sharing.

Algebraic simplification removes identities when valid: integer x+0→x. Strength reduction replaces an operation with an equivalent cheaper form, such as unsigned x*8→x<<3 for a valid width and shift count.

Target costs determine whether it is actually cheaper.

Peephole optimization examines short instruction windows.

It can remove redundant loads/stores, replace machine idioms and shorten jump chains: jump L1 where L1 immediately jumps L2 can become jump L2. Local optimization is a wider category; peephole optimization is a concrete method within the relevant scope.

Compiler Design module

COMP32 | Loops: invariants, induction and fusion

A loop-invariant expression has unchanged inputs during a loop.

Hoist it to a preheader only when evaluation is safe on the paths where it will newly run.

Induction variables change by a loop-invariant increment, such as i=i+1.

For repeated t=4*i, maintain a running t increased by 4 when equivalent, reducing repeated multiplication.

Loop fusion or jamming combines loops with compatible bounds and safe dependence order.

Unrolling replicates a body to reduce branch overhead and expose parallelism; handle remaining iterations and increased code size.

Loop-invariant code motion, fusion, unrolling and strength reduction are different transformations.

A finite taxonomy needs a criterion: machine dependent/independent versus local/global/loop are different classifications.

Compiler Design module

COMP33 | Optimization legality and aliasing

Optimization preserves defined observable behavior, including required side effects and exceptions.

Alias analysis asks whether references can denote the same memory.

After *p=5, a previous load through q cannot be reused unless p and q are known not to alias or an equivalent proof applies.

Function calls can also invalidate memory facts.

Do not move a potentially trapping division outside a loop that might execute zero times.

Floating-point reassociation can change rounding and NaN behavior.

Signed C/C++ overflow and shift rules differ from unsigned modular arithmetic; a mathematical identity is not automatically a legal transformation.

Dependence analysis must show that fusion or scheduling preserves the required order of memory accesses.

Compiler Design module

COMP34 | Target code generation

Code generation lowers checked IR to a target form.

Instruction selection chooses target operations; register allocation chooses where values reside; scheduling orders instructions while respecting dependencies; storage layout assigns offsets and follows ABI requirements.

Register assignment names actual physical registers after allocation decisions.

For t=a+b, a simple target could emit LOAD R1,a; LOAD R2,b; ADD R3,R1,R2; STORE t,R3.

Other targets may combine loads with arithmetic or keep t in a register.

Machine idioms, register availability and instruction order affect execution time.

Absolute machine code, relocatable object code, assembly and VM bytecode can be legitimate compiler targets; TAC is normally intermediate, but “cannot ever be target code” is an unjustified universal.

Compiler Design module

LLVM target-independent code generator

COMP35 | Descriptors and getReg

A register descriptor records which current values are held in a register.

An address descriptor for a name records all current locations containing its value, possibly memory and multiple registers.

A memory copy can become stale after a register-only assignment.

These are code-generator bookkeeping structures, not source-language types.

A simple code-generation algorithm calls getReg for a TAC instruction to select operand/result registers, emits necessary loads, emits the operation, then updates descriptors.

Prefer an already suitable register or a reusable dead operand; otherwise choose a spill victim and store any live value lacking another current copy.

At a block exit, preserve values needed by successors.

Allan Gottlieb, NYU Compilers Lecture 14

COMP36 | Interference graphs and spilling

A value is live where it can be used before redefinition.

In the interference model, values that must coexist in distinct registers are connected by an edge; each register is a color.

Nonoverlapping live ranges may share a register.

If a valid assignment cannot fit the available registers, spilling places selected values in memory and inserts loads/stores.

The general graph k-colorability decision problem is NP-complete for fixed $k \geq 3$; optimal register assignment/spill minimization is a hard optimization problem in general, so practical allocators use heuristics.

Restricted interference graphs or SSA formulations can have different properties.

Do not describe every possible register-allocation algorithm as NP-complete merely because a general model is difficult.

Cornell CS4120, Register Allocation notes

COMP37 | Runtime storage and calling conventions

Static storage lasts throughout execution; stack frames support nested calls and recursion; heap storage supports lifetimes beyond a call.

A frame may contain parameters, return address, saved registers, locals and temporaries.

Sizes, alignment and offsets are target-dependent; a two-byte int is an example, not a C++ guarantee.

A calling convention specifies argument and result locations, caller/callee-saved registers, stack alignment and frame responsibilities.

Escape analysis asks whether an object/reference can outlive or be accessed outside a region such as its allocating call.

Proven nonescaping allocations may use stack storage or be eliminated by scalar replacement, provided identity and lifetime semantics remain correct.

Compiler Design module

COMP38 | Inlining and dynamic dispatch

Inlining replaces a call with a suitably renamed copy of its body, preserving argument evaluation, returns and side effects.

It can remove call overhead and expose optimization opportunities, but may increase code size and instruction-cache pressure.

A recursive function cannot simply be expanded without limit.

Virtual dispatch chooses an overridden method based on a receiver's runtime class.

A common implementation uses a per-class virtual-method table and an object pointer to the table; the call loads the selected entry.

This is an implementation strategy, not a universal language requirement.

Name mangling distinguishes linker symbols, often for overloads; it is not itself runtime method selection.

Proven receiver types may permit devirtualization.

Compiler Design module

COMP39 | Recovery and compiler-construction tools

Panic-mode recovery skips input until a synchronizing token such as ; or } permits parsing to resume.

Phrase-level recovery makes a local insertion/deletion.

Report the location and cause while avoiding cascades of misleading diagnostics.

Lexical, syntax and semantic phases each detect their own errors; error handling is a shared service.

Lex/flex generates scanners from patterns and actions.

Yacc means Yet Another Compiler Compiler; Yacc/Bison generates parsers from grammar rules and semantic actions.

Typical specification files have definitions, %% rules, and %% user code.

Classic Yacc uses LALR; Bison supports additional LR and generalized modes.

Scanner generation does not replace the CFG parser.

Compiler Design module

COMP40 | Trace one expression through the compiler

For a=b+c*d;, the scanner recognizes names and operators, the parser forms an assignment with +(b,*(c,d)), and semantic analysis resolves declarations and operand/result types.

A declaration such as float a; int b,c,d; can require a conversion before storing the numeric result, according to language arithmetic rules.

Generate t1=c*d; t2=b+t1; a=convert_if_needed(t2).

Constant operands can fold; repeated unchanged expressions can share values.

Finally choose target instructions and registers.

The same spelling a can denote different declarations in nested scopes.

At each step ask what representation enters, what new information is added, and which errors can now be justified.

Compiler Design module

COMP41 | Distinguish tokens, lexemes and lexical analysis

The primary output of lexical analysis is a token stream, rather than a symbol table, AST or assembly; some are side structures. Distinguish a token category from the lexeme text that matches it.

Counting the loop header gives 14 tokens, even if no option matches.

Grouping characters into token classes is lexical analysis; grouping tokens into expressions and statements is syntax analysis.

Identify whether an analysis groups characters into tokens or tokens into syntactic constructs.

Symbol tables store identifiers and their attributes, not obligatorily every kind of token.

Trees store hierarchical constructs, while a symbol table stores declaration metadata.

Compiler Design module

COMP42 | Compare parser methods and grammar conditions

Shift-reduce and LR are both bottom-up rather than top-down methods.

Recursive and nonrecursive predictive parsers can both be efficient.

Nested parentheses and parsing tables do not uniquely identify LR.

Ambiguity does not disappear merely by choosing LR, and leftmost derivation is not an ambiguity-resolution policy.

Overlapping FIRST sets, left recursion and ambiguity can each prevent a useful grammar from being LL(1); do not restrict the check to only the first two.

Check all applicable conditions and identify any conflicting table cell.

Precedence versus operator-precedence parsing needs a concrete conflict and policy, not a blanket claim that every shift-reduce conflict has one solution.

Compiler Design module

COMP43 | Distinguish semantic analysis and translation

Type checking and symbol tables are both important to semantic analysis: one is an activity and one a data structure.

Semantic rules generally compute attributes, including but not limited to types, values and code; no listed specialized purpose is automatically exhaustive.

Both names and types belong in symbol records.

Distinguish division by zero, bounds violations, indentation mistakes and syntax errors from type errors; a type-error example should specify the language and incompatible operands.

Code generation is conventionally last in a simplified pipeline even if later target optimization exists.

A compiler may emit an annotated AST or typed IR, so distinguish representation, annotation and phase before forcing one tree name.

Compiler Design module

COMP-S01 | Parser direction, expression parsing and coverage

The common two-way classification of parsing is by construction direction: top-down parsing expands from the start symbol, while bottom-up parsing reduces the input toward that symbol. This counts broad directions, not the number of parser algorithms or implementations.

Operator-precedence parsing is a restricted bottom-up expression-parsing method using a stack and precedence relations. With a suitable expression grammar, the stack retains unfinished surrounding expressions while nested parentheses are processed.

In the standard comparison by grammar coverage, canonical LR(1) is more general than the restricted operator-precedence method. Together with SLR(1)⊆LALR(1)⊆LR(1), this makes canonical LR(1) the broadest of those named deterministic parsing methods. Broader coverage does not mean faster execution or smaller tables.

Compiler Design module · page 10

Compiler Design module · page 11

NIT Warangal Compiler Design Lab: operator-precedence parsing

NYU Compiler Construction: simple LR parsing and grammar coverage

Crespi Reghizzi and Pradella: Higher-order Operator Precedence Languages

COMP-S02 | Grammar notation and general CFG parsing algorithms

Backus-Naur form, or BNF, is a notation for context-free grammar production rules. It describes how syntactic constructs are formed; it is different from an abstract syntax tree representing one particular program.

CYK and Earley are context-free grammar parsing algorithms. They determine grammatical structure or membership of an input sequence. They are different tasks from Thompson construction, which builds a token-recognizing NFA from a regular expression, and subset construction, which turns an NFA into a DFA.

GNU Bison: Languages and Context-Free Grammars

Lark official documentation: parsing algorithms

Compiler Design module · page 8

COMP-S03 | Equivalent specifications for ordinary token languages

Classical regular expressions, consistently right-linear regular grammars and finite automata describe the same regular-language family. Thus an ordinary token pattern specified by a regular expression also has a regular grammar and an equivalent finite-state recognizer.

This equivalence concerns expressive power, not spelling of the specification or implementation size. Context-free grammars can also describe regular languages, but their larger family is needed for syntax with unbounded nested structure rather than ordinary regular token patterns.

Automata and Complexity Theory module · page 7

Automata and Complexity Theory module · page 8

COMP-S04 | Target-aware instruction scheduling

Instruction scheduling orders selected target instructions while preserving dependencies. A target-aware scheduler also considers instruction latencies and the processor resources needed by each operation. This is a backend optimization whose performance choices depend on the target machine, even when a reusable scheduling framework serves many targets.

By contrast, constant folding, dead-code elimination and safe loop-invariant motion can be expressed on a target-neutral IR using values, control flow and side-effect rules. Target costs can still influence whether a legal transformation is worthwhile. Machine independence therefore describes the abstraction of a pass, not freedom from all language semantics or implementation costs.

LLVM - The Target-Independent Code Generator

LLVM - TargetSchedModel reference

Compiler Design module · page 18