CS Exit Ethiopia

Revision Outlines

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	3
Fundamental of Database Systems	23
Advanced Database Systems	50
Object Oriented Programming	80
Computer Organization and Architecture	102
Data Communication and Computer Networking	128
Data Structure and Algorithms	150
Web Programming	180
Operating System	207
Software Engineering	234
Design and Analysis of Algorithms	261
Introduction to Artificial Intelligence	293
Computer Security	318
Network and System Administration	341
Automata and Complexity Theory	369
Compiler Design	394

Computer Programming

Revision Outlines

Problem-to-program workflow | Exam focus

- Problem-to-program workflow
  - Inputs, outputs, legal ranges, edge cases
  - Manual small-case solution
  - Algorithm → pseudocode or flowchart
  - Implement → test → maintain
  - Largest-of-three: best=first input
  - Compare second, then third
  - Zero initialization fails all-negative inputs
- Programming responsibilities
  - Writing, testing, debugging, maintaining
  - Actual versus required result
  - Execution alone insufficient

Algorithms and paradigms | Exam focus

- Algorithm properties
  - Precise executable steps
  - Termination for every legal input
  - Definiteness: unambiguous meaning
  - Effectiveness: performable steps
  - Optional inputs, specified result
  - Correctness against specification
  - Efficiency: desirable, measurable
  - Inefficient terminating algorithms still qualify
  - Module's desirable-efficiency listing
- Programming paradigms
  - Procedural: procedures organize operations
  - Structured: sequence, selection, iteration
  - Object oriented: state and behavior

- Functional: function evaluation
- Conditional: construct, not paradigm

Language generations and portability

- Language generations
  - First: machine instructions
  - Second: symbolic assembly
  - Third: C++, Java, general high-level
  - Fourth: SQL, queries, application generators
  - Fifth: loose historical category
  - Declarative logic, constraints, AI/expert systems
  - Generation independent of interpretation
- Language portability and conventions
  - Machine/assembly: instruction-set dependence
  - Readable high-level source
  - Recompile without platform assumptions
  - Humans write assembly and high-level
  - Python, Perl, PHP: conventional scripting
  - C++: conventionally compiled
  - Usage labels, not technical prohibitions

Compilation, linking and loading | Exam focus

- Conventional C++ toolchain
  - Source → preprocessing → compilation proper
  - Assembly → object files → linking
  - Executable → loading → execution
  - #include: preprocessing directive
  - Compiler may emit assembly
  - Assembler: object-file machine instructions
  - Linker: external-symbol resolution
  - Object files and libraries combined
  - Loader: memory placement and startup
- Translation caveats

- GCC: first four stages
- Compiler output needn't be executable/native
- Java javac → bytecode
- JVM: interpretation or runtime compilation
- Interpreters execute program representations
- Line-by-line: beginner approximation
- Multiple compilation passes
- Whole-program one-step claim overbroad

Identifiers, keywords and comments

• ASCII identifier rules
- Initial letter or underscore
- Subsequent letters, digits, underscores
- Keywords forbidden
- Case sensitivity: exitExam ≠ ExitExam
- delete, float: keywords
- FLOAT, variable_1234: valid
- 4thInput, T-3, spaces: invalid identifiers
- Reserved underscore patterns avoided

• Comments and entry point
- // line comments
- /* ... */ block comments
- Intent, assumptions, maintenance
- Comments don't execute
- No automatic speed/reliability/portability improvement
- main: entry function, not keyword

Declarations, constants and types | Exam focus

• Declaration versus initialization
- type name;
- type name = value;
- int age = 20;
- Declaration: name and type

- Initialization: initial value
- Later assignment replaces value
- Scalar a=b copies value
- Later b changes leave a unchanged
- Pointer assignment copies address
- Java object-reference assignment copies reference

• Types, constants and aliases

- int: whole numbers
- float/double: fractional approximations
- char: character
- bool: logical values
- Implementation-dependent C++ sizes
- Java widths don't transfer
- short n ≡ short int n
- int short n: same signed short
- const int ≡ int const
- Ordinary const int requires initialization
- typedef: alias, not distinct type

Literals, conversions and sizeof

• Literals and storage sizes

- 'A': character literal
- "A": string literal
- Terminator storage beyond visible characters
- sizeof: bytes
- sizeof(char)==1
- Byte width and sizes implementation-dependent
- sizeof(array): total before pointer conversion

• Conversion timing

- int x=5,y=2: x/y=2
- Double assignment after division: 2.0
- static_cast<double>(x)/y=2.5
- Cast before fractional operation

- In-range float→int: toward-zero truncation

Arithmetic and compound updates

- Arithmetic grouping and validity
  - *, /, % before +, -
  - Equal-precedence arithmetic: left grouping
  - Integer division: toward-zero truncation
  - a == (a/b)*b + a%b
  - Identity requires valid division
  - %: integral operands only
  - (8.0*3)%5: invalid
  - Division by zero: no ordinary output
  - Signed overflow: no ordinary output
- Compound assignment traces
  - +=, -=, *=, /=, %=
  - Operation then conversion to x's type
  - Simple x op= y: x=x op y
  - p=q=n=8: p+=n → 16
  - p/=q → 2
  - x=10: +=5, /=2 → 7
  - x=x+x doubles, not generally increments

Increment operators and sequencing

- Prefix versus postfix
  - Expression value versus stored value
  - ++x: increment, yield new
  - x++: yield old, increment
  - a=8,b=5: a++ + --b=12
  - Final a=9,b=4
  - x=1,y=10,z=-3: (x++*y--)+ ++z=8
  - Final x=2,y=9,z=-2
- Grouping versus evaluation order
  - Repeated unsequenced modifications unsafe

- C++17 operator-form stream sequencing
- cout<<x++<<" "<<++x: 5 7
- Before C++17: undefined behavior
- Bank options omit correct C++17 output

Boolean, bitwise and precedence rules

• Truth-value operations
- Relational and equality tests
- && conjunction, || disjunction
- ! negation
- C++ zero=false, nonzero=true
- Default bool output: 0/1
- boolalpha changes Boolean formatting
- a=3,b=5: (a==b) prints 0

• Short circuit versus bitwise
- &: bits, no short circuit
- && right only after true left
- || right only after false left
- false && dangerousCall(): skipped
- x=1,y=0,z=4: (x&y)||++z
- Final z=5, Boolean result=1
- Grouping/postfix → unary → multiplicative
- Additive → shifts → relations → equality
- Bitwise & → ^ → |
- && → || → conditional
- Assignment → comma

Conditional and comma expressions

• Conditional and comma operators
- condition ? yes : no
- Condition then exactly one alternative
- Comma: left then right
- Comma result: right operand value

- Argument separators differ from comma operator

• Conditional-comma trace

- a=6,b=8,x=2,y=4
- x>y false: (b--,y)
- Final b=7,c=4,a=6
- Unselected alternative unexecuted
- Internal semicolon: invalid syntax
- Comma restoration requires labeled interpretation

Decisions and flowcharts | Exam focus

• Decision grouping

- Independent consecutive if statements
- else: nearest unmatched if
- else-if: first true branch
- Nested S2: c1 && !c2
- Braces determine grouping

• Decisions and flowchart evidence

- ph=15.3: warning plus Alkaline
- Separate range check and classification
- else/early return creates exclusivity
- Diamond: decision
- Rectangle: processing
- Parallelogram: input/output
- Terminator: start/end
- Pre-test: check before body
- Post-test: check after body
- Missing arrows/condition placement: indeterminate loop

Switch and fall-through

• Traditional C++ switch

- Integral or enumeration selector
- case: constant expressions
- Matching label starts execution

- Fall-through until break/return/end
- default: unmatched value
- default needn't be last

• Switch fall-through trace

- Input 2: Bus.Truck.
- Case 2 enters, case 3 continues
- break: nearest switch or loop
- break doesn't terminate whole program
- case label: entry point
- No automatic branch isolation

Loop mechanics | Exam focus

• Pre-test versus post-test loops

- while: condition before each body
- while: possibly zero executions
- do-while: body before condition
- do-while: at least once
- Repeat while condition true
- Menu/input attempt use case

• for execution order

- Initialization once
- Condition → body → update → condition
- for(;;): absent condition=true
- Known-count convenience, condition-control flexibility
- Indices 0..n-1: i=0,i<n,++i
- i=0,i<5,i+=2: 0 2 4

Control transfers

• Control-transfer scope

- break: nearest loop or switch
- continue: skip current iteration remainder
- for continue → update
- while continue → condition

- return: leave current function
- Optional return result
- main return: normal program end

• goto label tracing

- Transfers within same function
- Follow reached labels
- a → b → end
- Graduating fourth
- Year unreachable from entry a
- Structured alternatives ease reasoning

Iteration counts and nontermination

• Loop state tables

- Condition inputs, body output, updated values
- i=0,j=8: i<j, i++,j--
- Entries (0,8),(1,7),(2,6),(3,5)
- Odd-i continue: two prints
- Outer i=1,i<5,i+=5: once
- Inner j=10,j>i,--j: nine prints

• Progress and early exit

- n=1,n<=18,n+=2
- Break before multiples of seven
- Output: 1 3 5
- Fixed a=6>b=5: infinite while
- data[0]=5: %10 guard false
- Conditional-only increment: no progress
- No output, infinite loop
- <=10→<10 alone insufficient

Functions and reusable modules | Exam focus

• Functions and call syntax

- Named task grouping
- int add(int a,int b){return a+b;}

- add(x,y): no call-site parameter types
- Parameters: definition names
- Arguments: supplied expressions/values
- void: no return value
- Side effects still possible
- return; permits early void exit

• Reusable organization
- Method: class/object-associated function
- Methods can return values
- Modules: responsibilities behind interfaces
- Reuse reduces duplication
- Independent functions: understandable, testable
- Libraries: reusable components
- Frameworks: organizing structure
- Frameworks often invoke callbacks

Value, reference and pointer parameters | Exam focus

• Pass-by-value isolation
- Separate parameter object from argument
- func: a*=5,b+=5,return a+b
- func(9,3)=53
- Caller values remain 9,3

• Aliasing and copied addresses
- int& aliases caller variable
- Reference swap changes originals
- int*: copied address
- *p=value mutates pointed object
- p=anotherAddress changes local pointer
- int a[] parameter adjusts to int*
- Element mutation possible
- Array length passed separately
- Java always passes by value
- Object reference copied, caller variable unaliased

Overloads, defaults and inline

- Overload resolution and defaults
  - Same name, different parameter count/types/order
  - Return type alone insufficient
  - Argument types then required conversions
  - No unique best match: ambiguity
  - sum(x,y,z=0): two/three arguments
  - Ordinary defaults occupy trailing positions
- inline meaning
  - No compulsory call-site expansion
  - Small frequent functions: optimization candidates
  - Optimizer decides expansion
  - Linkage/definition rules also matter
  - Not simply a speed command

Scope, lifetime and static locals

- Scope versus lifetime
  - Scope: name usability
  - Lifetime: object existence
  - Local name: block restriction
  - Inner names may shadow outer
  - Global availability requires visible declaration
  - ::name selects hidden global
- Automatic versus static local
  - Automatic: fresh on entry
  - Static: initialized once
  - Static value persists, name stays local
  - Even +1, odd +2, static x=0
  - Five calls: 1,3,5,7,9
  - Sum=25
  - Namespace/static integers: zero-initialized
  - Uninitialized local int: no assumed zero

Recursion and call tracing | Exam focus

- Recursive termination
  - Direct or indirect self-call
  - Base case plus progress
  - Every supported input must reach base
  - factorial(0)=1
  - factorial(n)=n*factorial(n-1), n>0
  - factorial(3)=3*2*1*1=6
  - Negative inputs: reject/separately define
  - n-1 doesn't reach zero from negative
- Recursive call tracing
  - Conditions before expansion
  - if(n<m) return 0;
  - f(1,4): immediate zero
  - Unreachable recursion contributes nothing
  - Record local parameters and pending operations
  - Unwind return values
  - Smaller instances commonly used
  - Base branch alone doesn't prove reachability

Arrays and bounds

- Built-in array model
  - Fixed count, same type
  - Contiguous storage
  - int a[4]={10,20,30,40};
  - Valid indices: 0..3
  - a[4]: undefined C++ access
  - No predicted nearby-memory output
  - Java: ArrayIndexOutOfBoundsException
- Array declarations and initialization
  - Standard bound: compile-time constant
  - Variable-length locals: compiler extension
  - int a[5]={1,2,3}: remaining zeros

- Uninitialized automatic int array differs
- a[i] ≡ *(a+i)
- Brackets not required for element access

Array algorithms and matrix indexing | Exam focus

- Array traversal algorithms
  - Sum: total=0, add each once
  - Maximum/minimum: initialize from a[0]
  - First verify nonempty array
  - Fifty inputs: count-controlled loop
  - Positive-only sum: if(x>0) total+=x
  - Skip nonpositive: continue for x<=0
  - Counting versus filtering responsibilities
- Two-dimensional indexing
  - int a[R][C]: R rows, C elements
  - Zero-based row then column
  - Row-major offset: r*C+c
  - Bank 4×4: a[1][1]=6
  - a[2][3]=6, a[3][3]=2
  - Java length belongs to each array
  - arr.length-2 endpoint: incomplete traversal

C strings and std::string

- C-string termination and storage
  - Character array ending '\0'
  - "Hello": six char positions
  - strlen: five visible characters
  - Terminator excluded from strlen
  - sizeof fixed array: allocated bytes
  - Capacity may exceed strlen+1
  - cout<<a: terminated sequence, not address
- Text APIs and capacity
  - <cstring>: strcpy(destination,source)

- strcat(destination,source), strlen, strcmp
- Sufficient destination capacity
- strcmp equality: zero
- Otherwise sign, not necessarily ±1
- <string>: owning resizable std::string
- +: concatenation, ==: content
- size()/length(): stored-character count
- substr(start,count): segment
- No std::string toUpperCase()/trim() members

Pointers and dereferencing

• Pointer identity versus pointed value
- int x=10, int* p=&x
- *p: read/write x
- &p: pointer variable's address
- p=&a,q=&b: p=q redirects
- No change to a or b
- *p=*q: pointed-value copy
- Uninitialized/null/dangling dereference forbidden

• Arrays and pointer levels
- Common array-to-first-element pointer conversion
- Array isn't assignable pointer variable
- int** pp=&p
- **pp reaches x
- void*: holds object address
- Convert appropriately before C++ dereference

Pointer arithmetic and allocation

• Scaled pointer arithmetic
- p+k: k elements
- k*sizeof(*p) byte displacement
- Assume sizeof(float)==4
- Address 7000: p+3 → 7012

- cout<<p++: old address, advance
- cout<<*p++: old element, advance
- ++*p: change element
- p+q: invalid
- Same-array subtraction: element distance

• Allocation and pointer boundaries
- Within-array or one-past arithmetic
- One-past dereference forbidden
- new T ↔ delete
- new T[n] ↔ delete[]
- Prefer resource-owning standard containers
- Primitive initialization must be requested

Structures and member access

• Structures and member access
- Named members, possibly differing types
- House definition: price and rooms
- House h{150000,3}: object creation
- h.rooms: dot access
- p->rooms ≡ (*p).rooms
- Pointer must reference live object
- Type definition alone creates no instance

• Aggregate and recursive structure rules
- Initialization in declaration order
- Default member initializer: int rooms=3
- C++11 list initialization rejects narrowing
- Exactly representable floating constant exception
- 25.00→float not necessarily narrowing
- 25.0f clarity, language-version check
- Structures copied, passed, returned
- No full self-instance member
- Self-type pointer allowed

Console streams and input state

- Console streams and spelling
  - <iostream>, std::cin/std::cout
  - Explicit qualification or introduced names
  - cout <<: output insertion
  - cin >>: formatted input extraction
  - std::endl: newline plus flush
  - Cout, Int: incorrect case
  - std: :cout: invalid printed spelling
- Input state and output formatting
  - getline(cin,text): full line
  - Numeric extraction leaves pending newline
  - Consume newline before line read
  - Check input failure before value use
  - Exact double 4 may print 4
  - Text, spaces, newlines affect traces

File streams and safe reading

- File stream types and modes
  - <fstream>: file-stream header
  - ofstream writes, ifstream reads
  - fstream: both directions
  - Default ofstream argument: ios::out
  - Ordinary effect: create/truncate
  - Signature versus effective out|trunc
  - ios::app: every write at end
  - ios::ate: initial seek to end
  - ios::binary: binary mode
- Safe file-reading lifecycle
  - if(!file): opening failure
  - while(file>>record)
  - while(getline(file,line))
  - EOF discovered by attempted read

- while(!file.eof()): possible invalid extra record
- Close or rely on destruction
- ofstream class versus <fstream> header

Math functions and iterative powers | Exam focus

• Mathematical functions
- <cmath>: sqrt, pow, floor, ceil
- <cstdlib>: integer abs also
- sqrt(16)=4, pow(3,4)=81, abs(-7)=7
- ceil(x): least integer-valued result ≥x
- floor(x): greatest integer-valued result ≤x
- ceil(-2.7)=-2, floor(-2.7)=-3
- Negative truncation differs from floor

• Iterative exponentiation
- No built-in C++ exponentiation operator
- ^: bitwise XOR
- **: not power operator
- Nonnegative integer e: r=1
- Repeat r*=b exactly e times
- 4^3 → 64
- Powers/factorials: fixed-width overflow risk
- Specify valid input range

Proofs and loop invariants | Exam focus

• Proof methods
- Direct P⇒Q: assume P, derive Q
- Contrapositive: ¬Q⇒¬P
- Contradiction: false claim assumption → impossibility
- Counterexample refutes universal claim
- Induction: base case
- P(k) assumption → P(k+1)

• Loop-invariant summation proof
- Initial sum=0,i=1

- Guard: i<=n
- Invariant: sum=(i-1)*i/2
- Initialization establishes invariant
- Add i → i*(i+1)/2
- Increment i restores invariant
- Exit: i=n+1
- Result: sum=n*(n+1)/2
- Termination measure: n-i+1
- Nonnegative, decreasing while guard holds
- Assumptions: n>=0, no overflow

Debugging, exceptions and maintenance | Exam focus

- Error categories and debugging
  - Syntax: grammar violation
  - Type: semantic rejection despite valid syntax
  - Runtime: during execution
  - Logic: wrong behavior
  - Bug: defect
  - Debugging: locate and correct
  - First useful diagnostic, nearby tokens/declarations
  - Repair and rebuild
  - Debugger requires runnable program
- Testing and maintenance tools
  - Ordinary, boundary, invalid inputs
  - First actual/expected state divergence
  - Exceptions: matching-handler control transfer
  - No arbitrary undefined-behavior recovery
  - Refactoring: structure improved, behavior preserved
  - Version control: records, comparisons, rollback
  - Documentation: use, design, maintenance assumptions

Static typing and Java distinctions

- Static versus dynamic typing

- Static: types checked before execution
- C++ auto: inferred static type
- Dynamic: operations checked using runtime values
- No mandatory explicit static declarations
- Static typing allows type inference

• Java versus C++ APIs
- Java boolean, fixed-width numeric primitives
- Object references
- Java String: immutable
- substring(begin,end): exclusive end
- equals: content, ==: references
- Java arrays: length field
- Scanner nextInt() → pending nextLine() newline
- C++ std::string: mutable
- substr(start,count): different API

Malformed-code interpretation

• Evidence before output prediction
- Complete code required
- Diagram edges, conditions, labels required
- Post-test loop
- Entry body before decision
- Branch returns to body
- Post-test do-while: answer B
- Literal invalid versus labeled intended trace
- Documented OCR correction only

• Malformed-code reference cases
- Pointer members: p->m, (*p).m
- Cout/Cin ≠ cout/cin
- C1 ≠ c1, car_mode ≠ car_model
- goto skips unreached output statements
- Assignment trace: z=5, a=1
- Intended z=5 then a=1

- Case matters: Z ≠ z
- Negative-value test: x < 0
- C++ entry point: int main()
- C++ spelling: int and cout
- Quotes and comma punctuation repair
- No invented unique literal output

Interpret programming questions precisely

• Ambiguous question options
- Linking versus loading
- Machine instructions versus executable files
- Data members and initialization
- Equivalent short declarations
- Language version and narrowing conversions
- Equivalent pointer-member expressions

• Incorrect-key and wording traps
- Static typing versus explicit annotations
- Scalar declaration: int x;
- Void function definitions
- int myFunction(){}: may compile
- Reaching int function end: undefined behavior
- break exits the innermost loop
- Const spelling and required initialization
- SS167: counting versus skipping
- Loop exit versus iteration skip
- Inspect, rebuild, and retest
- Rebuild and retest complete repair

Fundamental of Database Systems

Revision Outlines

Database approach, benefits and costs

- Database versus DBMS
  - Coherent related application facts
  - DBMS defines/stores/retrieves/updates
  - DBMS protects/shares
  - Content versus management software
  - Database system applications/people/infrastructure
- File approach versus DBMS
  - Application-specific formats/access procedures
  - Repeated facts/update disagreement
  - Data isolation
  - Shared integrity difficulties
  - Concurrent update difficulties
  - Centralized metadata
  - Multiple views/authorization/transactions
  - Backup/recovery
  - Controlled redundancy ≠ zero duplication
  - Complexity/administration/storage/licensing costs
  - Tiny fixed application tradeoffs
  - Lightweight embedded DBMSs
  - Useful concurrent access
  - Uncontrolled concurrency → anomalies
- Database personnel
  - DBA access authorization
  - Backup/recovery/availability
  - Monitoring/tuning/routine maintenance
  - Coordinating database use
  - Designers → schema
  - Developers → user-facing programs

Data models and product names

- Data model choices
  - Structure/relationships/constraints/operations
  - Conceptual ER → entities/relationships
  - Relational → tables
  - Hierarchical → parent-child tree
  - Network → pointer-based owner/member links
  - Network permits multiple parents
  - Objects → identity/state/behavior
  - Physical → layouts/access paths
  - Representations ≠ query stages
- Products versus vendor/language
  - Relational PostgreSQL/MySQL/Oracle Database
  - Relational Microsoft SQL Server/IBM Db2
  - MongoDB → document DBMS
  - Java → programming language
  - IBM → vendor
  - IBM answer needs product name

Relations, schemas, instances and metadata

- Relation terminology
  - Relation → set of tuples
  - Tuple/row/record → attribute values
  - Attribute/column/field → property
  - Degree → columns
  - Cardinality → rows
  - Classical distinct tuples
  - Atomic cells/attribute domains
  - Meaningless row order
  - Named attributes → meaningless column order
  - SQL duplicates unless constrained
- Schema versus instance
  - Schema → names/structures/constraints

- Instance/state/snapshot → current data
- Student insertion → state change
- Column addition → schema change
- Metadata → data describing data
- Catalog/data dictionary
- Definitions/types/constraints
- Privileges/optimizer statistics
- Schema diagram → tables/key references
- Database diagram terminology overlap

Three-schema architecture and data independence | Exam focus

- Three-schema abstraction
  - External → user-group views
  - Conceptual → whole logical database
  - Entity types/relationships/types/constraints
  - Internal → files/placement/access paths
  - Abstraction levels ≠ deployment tiers
- Data independence
  - Physical → internal changes
  - Index addition preserves conceptual schema
  - Logical → conceptual changes
  - Preserved external views/application interfaces
  - Table split with compatible view
  - Logical independence harder
  - Applications depend on logical meaning
  - Abstraction hides irrelevant detail
  - Program-data insulation separates definitions/code

Requirements through implementation | Exam focus

- Requirements through implementation
  - Business facts/transactions/reports
  - Volumes/privacy/quality targets
  - Conceptual design → ER/EER

- Logical design → relations/keys/constraints
- Logical normalization check
- Physical → organization/indexes/storage
- Implementation → create/load objects
- Testing → requirements

• Design methodology qualifications
- Logical design tied to model
- Potential DBMS independence
- Later vendor-specific refinement
- Universal dependence claims need methodology
- Conceptual enterprise view → downward
- Normalization dependencies → upward
- Useful approaches ≠ mandatory directions

Entities, attributes and ER notation | Exam focus

• ER concepts and Chen notation
- Entity → distinguishable real/conceptual thing
- Student/course instances
- Entity type → common properties
- Entity set → current collection
- Relationship → entity association
- Rectangle → entity type
- Diamond → relationship
- Ellipse → attribute
- Double ellipse → multivalued
- Dashed ellipse → derived
- Inspect diagram legend

• Independent attribute dimensions
- Simple versus composite
- Address → City/Street
- Single-valued versus multivalued
- Student → several phone numbers
- Composite single-valued address

- Age derived from BirthDate
- Derived can also be single-valued
- Names alone cannot establish multiplicity

Superkeys, candidate keys and primary keys | Exam focus

- Key hierarchy
  - Superkey → tuple uniqueness
  - Candidate key → minimal superkey
  - Remove component → uniqueness lost
  - Minimal ≠ globally fewest columns
  - Primary → selected candidate
  - Alternate → remaining candidate
  - Composite → multiple attributes
  - Composite need not be primary
- Key assumptions and traps
  - Unique ID → {ID} superkey
  - {ID,City} nonminimal superkey
  - Several candidates/foreign keys
  - At most one primary-key constraint
  - Multicolumn primary key allowed
  - String keys allowed
  - Prefer stable compact identifiers
  - Email/SSN/EmployeeID uniqueness needs rules
  - Single-column options ≠ composite example
- Worked example Eligible versus preferred primary keys
  - CustID/Email/ContactNo candidate keys
  - All eligible
  - CustID preference needs stability/size policy
  - Uniqueness alone cannot choose

Weak entities and identifying relationships | Exam focus

- Weak entity identification
  - No independent complete key

- Owner normally strong entity
- Owner/strong entity-set terminology overlap
- Partial key = discriminator
- Distinguishes same-owner weak entities
- Owner key + partial key

• Dependent weak-entity mapping
- Names unique within employee
- Key (EmployeeID,DependentName)
- EmployeeID foreign key
- Total identifying participation
- Chen double rectangle
- Double identifying diamond
- Partial-key dashed underline
- Foreign key alone ≠ weak entity

Cardinality, participation and direction | Exam focus

• Relationship cardinality and degree
- Maximum → 1:1/1:N/M:N
- Minimum zero → optional/partial
- Minimum ≥1 → mandatory/total
- Maximum one ≠ compulsory participation
- Degree → entity roles
- Unary recursive/binary/ternary

• Direction and notation assumptions
- Employee→Department N:1
- Department→Employee 1:N
- Building→Location possibly N:1
- Names alone insufficient
- Advisor arrows to both entities
- Common convention → maximum-one both ends
- Resulting 1:1
- Unknown convention → unsafe arrow inference

Map entities, composite and multivalued attributes | Exam focus

- Strong-entity mapping
  - One relation per entity type
  - Preserve simple attributes
  - Select candidate key
  - Expand composite components
  - Flat atomic relational design
  - Calculate derived values
  - Avoid stale stored copies
- Multivalued-attribute mapping
  - Separate owner-key/value relation
  - StudentPhone(StudentID,Phone)
  - Primary key (StudentID,Phone)
  - StudentID foreign key
  - Separate phone rows
  - No comma-separated lists
  - No Phone1/Phone2 repetition
  - Explicit omitted mapping step
  - Entity-based table names valid

Map 1:1 and 1:N relationships | Exam focus

- 1:N relationship mapping
  - One-side key → many-side FK
  - Relationship attributes on many side
  - Department/Employee DeptID reference
  - HireDate relationship attribute
  - Total participation → NOT NULL
  - Partial → nullable unassigned reference
- 1:1 relationship mapping
  - Foreign key plus UNIQUE
  - Prefer totally participating side
  - Reduce NULLs
  - NOT NULL → mandatory reference

- Merge only compatible lifecycles/participation
- A→B many-to-one reference in A
- Total or partial participation allowed
- Total-only mapping claim overstated

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

- M:N association table
  - Enrollment(StudentID,CourseID,Grade)
  - References Student and Course
  - One enrollment/pair → composite key
  - Repeated attempts → Term/AttemptNo key
  - Ordinarily three tables
- Recursive and ternary mapping
  - UserLink(UserID,OtherUserID)
  - Two User foreign keys
  - Directed versus undirected links
  - Self-link policy
  - Employee nullable SupervisorID
  - Recursive 1:N supervision
  - Supply(SupplierID,PartID,ProjectID,Quantity)
  - Preserve participating references/triples
  - Binary replacements can lose triples
  - Association key from actual cardinalities
  - All references not automatically necessary

EER inheritance and category constraints | Exam focus

- EER inheritance constraints
  - Superclass/subclass
  - Inherited attributes/relationships
  - Specialization → Undergraduate/Graduate
  - Generalization → common Student properties
  - Disjoint → no overlap
  - Overlapping → shared members

- Total → every superclass member
- Partial → membership not compulsory

• Inheritance and category mapping
- Person(PersonID,Name)
- Student(PersonID,Program)
- Subclass PersonID both PK/FK
- Preserved inherited identity
- Category/union → Person or Company
- Intersection → both superclasses
- Object inheritance also supported
- ER answer requires EER scope
- Structural inheritance ≠ executable methods

Fan traps and chasm traps

• Fan trap
- Common entity fans into two 1:N
- Department→Employee and Department→Project
- Path cannot determine WorksOn
- Department join → every employee/project pair
- Record WorksOn explicitly

• Chasm trap
- Optional relationship breaks path
- Unstaffed branch property disappears
- Add required direct relationship
- Or revise model
- Multiple 1:N relationships legitimate
- Eliminating all fans ≠ universal repair
- Missing path → chasm
- Fan → association ambiguity

Integrity and constraint taxonomy | Exam focus

• Integrity categories
- Domain → type/allowed values

- Key → uniqueness
- Entity → every PK component non-NULL
- Referential → non-NULL FK matches key
- Eligible unique/candidate referenced key
- Self-references allowed
- Parent need not have child

• Constraint mechanisms and interpretation

- Inherent → data model
- Schema-based → DDL
- Semantic/application → triggers/application logic
- DDL expressiveness limits
- Integrity → validity
- Authorization → allowed access
- Uniqueness/domain validation both integrity
- NULL Department PK → entity violation
- Manager=107 missing employee
- Declared FK → referential violation
- One INSERT → multiple violations

Write and enforce SQL constraints | Exam focus

• SQL constraint declaration

- Enrollment StudentID/CourseID INTEGER NOT NULL
- REFERENCES Student(StudentID)/Course(CourseID)
- Grade INTEGER CHECK
- Grade BETWEEN 0 AND 100
- PRIMARY KEY (StudentID,CourseID)
- Column name then type
- Commas separate declarations
- CHECK clause ≠ standalone GRANT-like statement
- POSITIVE not general SQL keyword
- CHECK (Amount > 0)

• Constraint NULL and reference behavior

- PRIMARY KEY → unique/non-NULL

- UNIQUE need not imply NOT NULL
- UNIQUE NULL treatment DBMS-dependent
- CHECK unknown can pass
- NOT NULL forbids missing values
- NO ACTION/RESTRICT
- CASCADE
- SET NULL
- Referenced keys can change
- Children/action determine legality

Functional dependencies and attribute closure | Exam focus

- Functional dependencies
  - X→Y: agreeing X implies agreeing Y
  - Every legal tuple pair
  - Business constraint ≠ sample correlation
  - Full → all determinant attributes
  - Partial → proper determinant subset
  - Prime → some candidate key
  - Nonprime → no candidate key
- Closure and Armstrong axioms
  - Start X^+ with X
  - Applicable FD → add RHS
  - Repeat until fixed point
  - X^+ covers relation ↔ superkey
  - Remove/recompute → minimality
  - Reflexivity: $Y \subseteq X \Rightarrow X \rightarrow Y$
  - Augmentation: $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
  - Transitivity: $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

Find all keys before deciding normal form | Exam focus

- Find every candidate key
  - R(A,B,C,D)
  - C→D,C→B,B→C

- A absent from RHS
- Every key needs A
- (AB)+=ABCD via B→C→D
- (AC)+=ABCD
- AB/AC minimal
- BC cannot derive A
- Prime A/B/C
- Nonprime D

• Highest normal form with partial dependencies
- C→D and B→D
- Proper subsets of AB/AC
- 2NF fails
- Atomic values → highest 1NF
- Read functional dependencies accurately
- C3D,C>PB,BOC alone insufficient
- Request actual FD set
- Never calculate guessed symbols

• Worked example Transitive dependency key test
- R(A,B,C), F={A→B,B→C}
- A+=ABC → key A
- Nonkey B → 3NF violation
- Candidate-key B changes conclusion
- Chain alone insufficient

Insertion, deletion and update anomalies

• Normalization anomalies
- Repeated Enrollment StudentName
- Update → every copy must change
- Missed copies → disagreement
- Insertion → student needs enrollment
- Deletion → last enrollment loses student
- Revoke ≠ normalization anomaly

• Normalization benefits and limits

- Separate facts by determinants
- Reduce repetition/anomalies
- Declared FDs suffice
- Bad current data unnecessary
- Generated row ID ≠ dependency repair
- Repeated facts persist
- Potential integrity/storage improvement
- Extra keys/joins
- Storage savings not guaranteed
- Faster retrieval not guaranteed

First and second normal forms | Exam focus

• First normal form
- Atomic chosen-domain values
- No repeating cell groups
- Primary-key declaration insufficient
- Correct composite/multivalued mapping → 1NF
- ER drawing ≠ higher-form guarantee

• Second normal form
- 1NF prerequisite
- No nonprime partial candidate-key dependency
- Enrollment key (SID,CID)
- SID→SName
- CID→CTitle
- Student(SID,SName)
- Course(CID,CTitle)
- Enrollment(SID,CID,Grade)
- Grade → whole pair
- Every candidate single-attribute → no partials
- No partials alone needs 1NF assumption
- Higher forms inherit 2NF

Third normal form, BCNF and higher forms | Exam focus

- 3NF versus BCNF
  - Every nontrivial X→A
  - 3NF: X superkey or A prime
  - BCNF: X superkey
  - Whole-key mnemonic limited
  - Overlapping keys need full tests
  - Remove nonprime/nonkey-determinant dependencies
- Sale normalization diagnosis
  - Key (Date,Customer,Product)
  - Product→Vendor
  - Vendor→VendorCity
  - Product proper key subset
  - Vendor/VendorCity violate 2NF
  - Atomic → highest 1NF
  - Transitivity does not imply highest 2NF
  - 4NF → nontrivial MVDs
  - Nonsuperkey MVD determinants
  - 5NF → join dependencies
  - MVD ≠ list inside cell
- Worked example Transitive Employee decomposition
  - EID→DeptID
  - DeptID→DeptName
  - Employee(EID,DeptID)
  - Department(DeptID,DeptName)
  - Department name stored once

Decomposition: lossless and dependency preserving | Exam focus

- Lossless decomposition
  - Join reconstructs exact legal tuples
  - Lossy → spurious combinations
  - Binary FD losslessness test
  - Common attributes determine R1 or R2

- Shared column alone insufficient

• Preservation and decomposition algorithms
  - Dependency preservation → local constraint enforcement
  - No join required
  - Separate from losslessness
  - 3NF synthesis → minimal FD cover
  - Relation per determinant/dependents
  - Remove contained relations
  - Add candidate-key relation if absent
  - BCNF → offending X→Y
  - Separate XY and remainder with X
  - BCNF may sacrifice dependency preservation

Choose data types and storage levels | Exam focus

• Data types by meaning
  - CHAR(n) fixed length
  - VARCHAR(n) variable bounded length
  - DATE → date
  - BLOB → binary/images
  - INTEGER → whole numbers
  - DECIMAL(p,s) → exact decimal
  - Money needs specified scale
  - FLOAT/REAL/DOUBLE approximate
  - Names/widths/precision DBMS-dependent
  - Reject universal numeric-width claims

• Storage hierarchy and type selection
  - Primary → RAM/processor cache
  - Usually volatile
  - Secondary → persistent online SSD/HDD
  - Pages copied to CPU buffers
  - Tertiary → tape/removable archives
  - Types from meaning/range/accuracy
  - Integers not universally faster than strings

Records, pages and primary file organization

- Pages and blocking calculations
  - Page/block → I/O unit
  - Several records/page
  - Fixed unspanned record size r
  - Usable page space B
  - Blocking factor = floor(B/r)
  - Pages = ceil(n/blocking_factor)
  - Headers/slots reduce usable space
- File organization tradeoffs
  - Heap → unordered/simple inserts
  - Heap search → possible full scan
  - Sorted → range scans/binary search
  - Ordering-field search
  - Costly inserts/overflow handling
  - Hashed → hash(key) bucket
  - Equality lookup/collisions/overflow
  - Hash order unsuitable for ordinary ranges
  - Choose from reads/writes/predicates
  - Partitioning → split
  - Replication → copy
  - Migration → move
  - Archiving → older data leaves active service

Ordered, dense, sparse, and multilevel indexes

- Index purpose and costs
  - Search keys plus pointers/row identifiers
  - Selective retrieval benefit
  - Space overhead
  - Write maintenance
  - Tiny table → scan can win
  - Most rows → scan can win
  - Index ≠ referential-integrity enforcement

- Index varieties and assumptions
  - Primary ordered → file ordering key
  - One entry/page → sparse
  - Clustering → nonunique ordering field
  - Often one entry/distinct value
  - Secondary → alternative access paths
  - Every searchable value represented
  - Record pointers/pointer lists
  - Dense → each search value
  - Sparse → ordered intervening data
  - Multilevel → indexes on indexes
  - B/B+ trees → ordered/range search
  - Hashing → equality
  - Bitmap → some analytic workloads
  - No universally best index

SQL categories and exact scope

- SQL command categories
  - DDL → structure
  - CREATE/ALTER/DROP
  - Textbook RENAME/TRUNCATE often DDL
  - DML → retrieve/change data
  - SELECT/INSERT/UPDATE/DELETE
  - SELECT sometimes DQL
  - DCL → GRANT/REVOKE privileges
  - TCL → COMMIT/ROLLBACK/SAVEPOINT
  - Classify by effect, not length
- Deletion and dialect scope
  - DELETE → rows removed/table retained
  - DROP TABLE → object removed
  - ALTER → definition changed
  - TRUNCATE → table emptied
  - Implementation-specific mechanisms

- PostgreSQL TRUNCATE rollback allowed
- No-rollback claim not universal
- SQL Server DROP DATABASE Sales, suppliers
- Comma list vendor-specific
- SQLite shell .databases → names/files
- .database undocumented command

SELECT and its result columns

• SELECT output semantics
- SELECT Name,Address FROM Customer
- Mailing-label fields only
- SELECT * → all columns
- DISTINCT → duplicate result removal
- AS → alias
- ALL → duplicate permission
- ALL ≠ all columns
- Default SQL preserves duplicates
- Column reads require appropriate FROM
- Literal SELECT 1 can omit FROM

• Written versus conceptual query order
- SELECT/FROM/WHERE/GROUP BY/HAVING/ORDER BY
- FROM/JOIN → WHERE
- GROUP BY → HAVING
- SELECT expressions → ORDER BY
- SELECT normally preserves state
- WHERE alone ≠ executable query
- Clause/statement distinction

Predicates, strings and NULL

• SQL predicate syntax
- Strings → single quotes
- WHERE ID = 'B001'
- Standard double quotes → identifiers

- Vendor string modes differ
- E.Department resolves ambiguity
- Comparisons =/<>/</>
- AND/OR/NOT
- Parentheses clarify precedence
- LIKE 'Ab%' → prefix
- BETWEEN includes endpoints

• NULL and timestamp boundaries
- NULL → missing/unknown
- Salary = NULL → unknown
- IS NULL/IS NOT NULL
- WHERE retains true only
- BirthDate >= DATE '2000-01-01'
- BirthDate < DATE '2001-01-01'
- Half-open timestamp year interval
- No December 31 time omissions
- Expression indexes support functions
- Functions not universally unindexable

INSERT, UPDATE and DELETE

• INSERT column/value correspondence
- INSERT INTO Hotel(HotelID,HotelName,SubCity)
- VALUES (1,'Hilton','Yeka')
- Counts/types must match
- Keys/constraints still apply
- No column list → declared order

• UPDATE and DELETE scope
- UPDATE Book SET Price = 600
- WHERE ID = 'B001'
- Matching rows only
- No UPDATE WHERE → every row
- DELETE FROM Publisher WHERE Name='XWZ'
- Reference policies still apply

- DELETE FROM r → all rows
- Table definition remains
- Clear entries/delete rows synonymous
- Several UPDATEs can be valid
- Appropriateness needs requested change/dialect

CREATE and ALTER tables

• CREATE TABLE and primary keys
- Employee EmpID INTEGER PRIMARY KEY
- Name VARCHAR(80)
- Name/type stay together
- Commas between column declarations
- ALTER TABLE Employee ADD PRIMARY KEY (EmpID)
- Existing values unique/non-NULL
- Second PK requires replacing existing constraint

• ALTER and malformed syntax
- ADD COLUMN Email VARCHAR(120)
- PostgreSQL-style syntax
- DROP COLUMN Email
- ADD CONSTRAINT uq_email UNIQUE (Email)
- Named constraint
- DALTER malformed token
- Column-name/type comma malformed
- Repair ≠ printed statement executes

Join behavior and preservation

• Join matching semantics
- Inner → matching ON pairs
- Equijoin → equality
- Theta join → other comparison
- Natural → every same-named column
- Ordinarily inner equality join
- Natural merges duplicate join columns

- USING (DeptID) explicit shared columns
- CROSS JOIN → every pair
- No matching condition

• Outer joins and join arity
- LEFT preserves unmatched left
- RIGHT preserves unmatched right
- FULL preserves both
- Missing-side columns NULL-padded
- Inner/natural-inner discard unmatched rows
- Self-join → aliases/base-table occurrences
- Operator combines two inputs
- Inputs may be earlier joins
- Query may join many tables
- Base-table count versus occurrence count
- Binary arity distinction

Aggregation, GROUP BY and HAVING

• Aggregation and group filtering
- COUNT(*) → rows
- COUNT(x) → non-NULL x
- SUM/AVG → non-NULL numeric values
- AVG, not AVERAGE
- GROUP BY → distinct-value groups
- WHERE filters individual rows
- HAVING filters aggregate groups

• Grouped query validity
- Department/AVG(Salary) output
- WHERE Department <> 'Sales'
- GROUP BY Department
- HAVING AVG(Salary) > 35000
- Ordinary aggregate predicate invalid in WHERE
- EmpID/FirstName/LastName not grouped
- Unless DBMS establishes functional dependence

- Grouping column need not be selected
- Damaged query defect ≠ omitted grouping output

Subqueries and views

• Subquery forms and NULL traps
- Nested SELECT
- WHERE/HAVING/FROM locations
- IN → membership
- EXISTS → qualifying row existence
- Correlated → outer-row reference
- Scalar → appropriate single value
- NOT IN with possible NULL
- NOT EXISTS often safer absence test

• Views and rewrite tradeoffs
- CREATE VIEW CSStudent AS SELECT
- StudentID/Name WHERE Major='CS'
- Stored query definition/virtual table
- Repeated-query simplification
- Derived values
- Configured access restriction
- Ordinary view need not store result
- Materialized view stores result
- Subquery→join can change duplicates
- Rewrite not guaranteed faster

Set operations and SQL conversion

• SQL set operations
- UNION → compatible results/deduplication
- UNION ALL → duplicates retained
- UNION DISTINCT dialect-dependent
- PostgreSQL accepts explicit DISTINCT
- Not separate algebra rule
- INTERSECT → common rows

- EXCEPT → first-only rows
- Same column count
- Compatible corresponding types

• Conversion and UNION repair
- CAST('42' AS INTEGER)
- Value must permit conversion
- CONVERT vendor-specific conversion
- Both options need dialect
- UNION xyz FROM table2 malformed
- Missing second SELECT
- SELECT expressions versus table names
- xyz ≠ table name

Relational algebra: calculate results

• Relational algebra operators
- Selection σ → tuple filtering
- Projection π → attributes
- Set projection removes duplicates
- Rename ρ → relation/attribute names
- Union/difference/intersection → compatible schemas
- R−S not commutative
- Union/intersection commutative
- Inner joins commutative with adjusted predicates
- Up to attribute order
- LEFT OUTER not generally commutative
- Vague join needs variant

• Product counts and algebra basis
- $|R \times S| = |R| \times |S|$
- Product degree = degree(R)+degree(S)
- Rename conflicting attributes
- 200×100=20,000
- 5×7=35
- Join = selection over product

- Intersection derivable from difference
- Basis σ/π/union/difference/product/rename
- Assignment useful, not indispensable
- Textbook primitive lists vary

Relational calculus: state the desired facts

• Algebra versus calculus
- Algebra → operation sequence
- Procedural at logical level
- Calculus → desired-result predicate
- Declarative/nonprocedural
- Tuple variables → tuples
- Domain variables → attribute values
- Safe/range-restricted expressions
- Results bounded by active domain

• Calculus forms and quantifiers
- {s | Student(s) AND s.Major='CS'}
- Tuple result → CS students
- Domain result → names n
- ∃i,m (Student(i,n,m) AND m='CS')
- Existential → some matching tuple
- Universal → every required case
- SQL declarative
- Optimizer chooses procedural execution plan

Transactions and ACID

• Transactions and ACID
- Transaction → logical operation unit
- Atomicity → all effects or none
- Consistency → correct invariant preservation
- Isolation → concurrent interference control
- Serializable → equivalent serial execution
- Durability → committed crash survival

- Durable logging typical
- Immediate page forcing unnecessary

• Transfer and transaction control
- Subtract/add 100 commit together
- Lost half → atomicity violation
- Possible total-balance invariant violation
- BEGIN → start
- COMMIT → complete
- ROLLBACK → undo uncommitted work
- SAVEPOINT → intermediate rollback point
- Other committed transactions remain
- Syntax/error-handling DBMS-dependent
- @@ERROR/IF SQL Server-specific

Choose techniques from requirements | Exam focus

• Match technique to requirement
- Normalization → dependency anomalies
- Indexing → search paths
- Denormalization → repeat/precompute facts
- Lower selected read costs
- Higher maintenance cost
- Partitioning → rows/columns divided
- Replication → copies
- Locality/availability benefits
- Update complexity
- Techniques can coexist
- E-commerce/gaming label insufficient

• Security and quality requirements
- Privacy → threat model
- Authentication → identity
- Role authorization → records/actions
- Encryption → at rest/in transit
- Auditing → activity records

- Patient data → encryption/access control
- Scalability/availability/retrieval speed
- Redundancy control/security
- Functional → behavior
- Nonfunctional → quantified quality
- 99.9% uptime/response time
- Mechanisms alone cannot classify requirement

• CMS meaning versus integrity
- Tags/categories → meaning
- Foreign keys → valid references
- ContentTag(ContentID,TagID)
- Multiple tags/item
- Complementary functions

Registration and dormitory design | Exam focus

• Registration/dormitory business rules
- Student → one department
- Student/course → many-to-many
- Student → at most one current dorm
- Dorm → at most six students
- Department/Student/Course/Dorm entities
- Enrollment relationship with Grade

• Integrated schema and capacity enforcement
- Department(DeptID,Name)
- Student(SID,Name,DeptID,DormID)
- Course(CID,Title)
- Enrollment(SID,CID,Grade)
- Dorm(DormID,FloorNumber)
- Entity IDs → primary keys
- Student DeptID/DormID → parent FKs
- Enrollment PK (SID,CID)
- Enrollment → both parent references
- Nullable DormID until assignment

- StudentPhone(SID,Phone) → multiple contacts
- Department/course facts in owning tables
- FK alone cannot enforce six capacity
- Concurrency-safe trigger/allocation transaction
- Or six unique slots/dorm

• Worked example Dorm reference direction
- FOREIGN KEY (DormID) REFERENCES Dorm(DormID)
- STUDENT → referencing child
- DORM → referenced parent
- Consistent names
- Student.DormID required before execution

Check SQL syntax and question assumptions

• Check supplied answer keys
- Owner/strong entity both possible
- Row/record synonyms
- Difference/set-difference synonyms
- Negative DROP question → multiple answers
- DELETE/UPDATE/invalid REMOVE
- Highlighting does not establish correctness

• Resolve question defects
- Damaged token → corrected form
- Printed malformed SQL remains invalid
- Unsupported cardinality → rules/direction
- Best/efficient/important → workload/goal/baseline
- Diagram conclusions need symbols/description
- Question index → exceptions/minimal clarification
- Unsupported unique answer ≠ useless teaching

Advanced Database Systems

Revision Outlines

Objects, identity, classes and persistence | Exam focus

- Object identity, state, behavior
  - Class → attributes/operations
  - Object → class instance
  - OID independent of attribute values
  - Lifetime-stable OID
  - Equal values/different identities
  - Name change ≠ identity change
- Persistence and database approaches
  - Object survives creating program
  - OODBMS durable storage/transactions
  - Querying/controlled sharing
  - Objects not primarily rows/tables
  - Object-relational → richer relational types
  - ORM → objects mapped to tables
  - Three distinct approaches
  - Complex CAD/multimedia suitability
  - M:N alone needs no relational abandonment

Encapsulation, abstraction and object associations | Exam focus

- Encapsulation versus abstraction
  - Encapsulation → state plus operations
  - Controlled implementation access
  - Account.withdraw(amount) → balance rule
  - Hidden storage details
  - Abstraction → relevant operation contract
  - What versus implementation how
  - Overlapping information hiding
  - Combining state/methods → encapsulation

- Object associations and collaboration
  - References link objects
  - Department.employees → Employee objects
  - Association ≠ inheritance
  - Inheritance → is-a types
  - Methods invoke other-object operations
  - Class → structure/behavior
  - Attribute → state
  - Method → operation implementation
  - Part names ≠ whole object

Inheritance, polymorphism and safe substitutability | Exam focus

- Inheritance versus association
  - Subtype from superclass
  - Reuse applicable attributes/behavior
  - GraduateStudent is-a Student
  - Student belongs-to Department
  - EER inherited attributes/relationships
  - Programming object model adds behavior
- Subtype polymorphism
  - Superclass/interface reference → compatible subclasses
  - Common operation/different implementations
  - Shape.area() dispatch
  - Actual Circle.area()/Rectangle.area()
  - No arbitrary unrelated-class conversion
  - No dynamic class mutation
  - Inheritance → structural subtype prerequisite
  - Polymorphism → runtime common-interface use
  - Ambiguous MCQ needs requested distinction

Object type constructors | Exam focus

- Object type constructors
  - (OID,type constructor,state)

- Atom → basic value
- Integer/real/Boolean/string
- Tuple → named component values
- (Street:'A',City:'B')
- Structured record ≠ record collection

• Collection constructors and nesting
- Set → unordered/no duplicates
- Bag → duplicates allowed
- List → ordered
- Array → indexed positions
- Nested constructors → complex objects
- Tuple with phone-number set
- One variable ≠ atomic constructor
- Type constructor → value shape
- Executable constructor → object creation

ODL, OQL and inverse relationships | Exam focus

• ODMG object languages
- ODL → Object Definition Language
- Object database schema definition
- Interfaces/classes/attributes/operations/relationships
- OQL → Object Query Language
- Host-language bindings → Java/C++
- Definitions/operations connected to host languages
- Logical schema ≠ storage implementation

• Inverse relationship declaration
- Department.employees ↔ Employee.department
- Opposite directions/same relationship
- Employee in department D
- Employee appears in D collection
- Relationship consistency
- Not inheritance keyword/scalar attribute
- ODL DDL/logical-schema descriptions overlap

- Exact MCQ wording matters

Transactions, ACID and state transitions

- Transactions and ACID
  - Logical database-work unit
  - Read-only transactions included
  - Atomicity → all effects or none
  - Consistency → correct integrity-preserving work
  - Isolation → concurrent interference control
  - Durability → committed effects survive failure
  - DBMS cannot repair incorrect business logic
- Transaction states and responsibilities
  - Active → executing operations
  - Partially committed → last operation finished
  - Durability not yet assured
  - Committed → commitment succeeded
  - Failure → Failed
  - Rollback → Aborted
  - Terminate or restart new attempt
  - Concurrency control → isolation
  - Recovery → atomicity/durability
  - Transaction management broader
  - Broad responsibility answers can overlap
- Worked example Requested transfer versus invariant
  - Initial A=5,000/B=8,000
  - Transfer 750 A→B
  - Final A=4,250/B=8,750
  - Total=13,000
  - Unchanged/reverse balances also preserve total
  - Correct debit/credit required

Read/write operations, buffers and log records

- Item operations and buffer policy

- read_item(X) → locate page
- Fetch absent page into buffer
- Copy X into program variable
- write_item(X) → variable into buffered page
- Mark page dirty
- Disk flush may occur later
- write_item ≠ immediate disk flush
- Insert/delete modify data
- Sorting → query processing
- Sorting ≠ primitive item update

• Log records and images
- START/UPDATE(T,X,old,new)
- COMMIT/ABORT/END
- Before-image → undo
- After-image → redo
- Physical/logical record alternatives
- Not every read needs logging
- COMMIT → successful completion
- ABORT may only start rollback
- ABORT ≠ completed undo proof

Diagnose concurrency anomalies from a trace

• Lost update versus dirty read
- Both read 500
- T2 +200 → writes 700
- T1 stale +100 → writes 600
- Correct serial result 800
- Dirty read → other's uncommitted write
- Writer abort → reconsider dependent work
- Own changes ≠ dirty read

• Read and summary anomalies
- Nonrepeatable → same-row committed change
- Phantom → changed predicate result set

- Matching insert/delete/update
- Incorrect summary → inconsistent versions
- Transfer 10: A=100/B=100
- Old A=100 plus new B=110
- Observed 210 versus invariant 200
- Single committed B=20→50 insufficient
- Invalid-summary 90 needs more evidence

Serial schedules, conflicts, and precedence graphs

• Serial versus serializable
- Serial → consecutive transaction operations
- Serializable → equivalent serial order
- Interleaving permitted
- Same final value insufficient
- Accidental coincidence possible
- Conflict → different transactions
- Same item/at least one write
- Read/read nonconflicting

• Precedence graph test
- Vertex per transaction
- Earlier/later conflict → Ti→Tj
- Acyclic → conflict-serializable
- Topological order → valid serial order
- Cycle → not conflict-serializable
- View serializability weaker test
- Read-from relationships/final writers
- Some blind-write schedules permitted

• Worked example Worked conflict graphs
- r1(X),w2(X),r2(Y),w1(Y)
- X → T1→T2
- Y → T2→T1
- Cycle → not conflict-serializable
- r1(X),w1(X),r2(X),w2(X)

- Only T1→T2
- Acyclic → conflict-serializable

Recoverable, cascadeless and strict schedules

- Commit-dependency schedule hierarchy
  - Recoverable → reader commits after writer
  - Avoid committed dependency on aborted writer
  - Cascadeless → read committed writes only
  - No cascading aborts
  - Strict → no other's uncommitted-item reads
  - No other's uncommitted-item overwrites
  - Strict ⇒ cascadeless ⇒ recoverable
- Recoverability worked trace
  - w1(X),r2(X),c2,a1
  - Dirty dependency committed → unrecoverable
  - c2 after c1 → recoverable dependency
  - Earlier dirty read remains
  - Not cascadeless
  - Serializability → equivalent order
  - Recoverability → safe commit/rollback dependencies
  - Separate tests

Locks, compatibility, conversion and managers

- Shared/exclusive lock compatibility
  - S → read permission
  - Multiple S locks coexist
  - X → holder reads/writes
  - X conflicts with other's S/X
  - Different-item locks need not conflict
  - Basic lock-based rules
  - Multiversion snapshot readers may differ
- Lock ownership, conversion, management
  - Appropriate lock before access

    - Release only own locks
    - S→X upgrade
    - Conflicting holders must leave
    - X→S downgrade
    - Lock manager → request decisions
    - Lock table → granted/waiting requests
    - Already locked request allowed
    - Compatible → granted
    - Incompatible → wait/abort by policy
    - No-double-locking ≠ shared-reader prohibition
- Held S compatibility
    - Requested S → compatible
    - Requested X → conflict
- Held X compatibility
    - Requested S → conflict
    - Requested X → conflict

Two-phase locking variants

- Basic two-phase locking
    - Growing → acquire/upgrade
    - Shrinking → release/downgrade
    - Any release → no new acquisition
    - Lock point → final lock obtained
    - Well-formed 2PL → conflict serializability
    - Deadlock possible
    - Basic 2PL may allow cascading aborts
- 2PL variants versus 2PC
    - Strict → X until commit/abort
    - Rigorous → S/X until commit/abort
    - Conservative/static → all locks beforehand
    - No execution with partial lock set
    - Prevents hold-and-wait deadlock
    - Requires known lock set

- May reduce concurrency
- 2PL → data-access ordering
- 2PC → distributed commit decision

Deadlock detection, prevention and starvation

- Deadlock detection and repair
  - T1 holds X/wants Y
  - T2 holds Y/wants X
  - Wait-for Ti→Tj → held-lock wait
  - Cycle → ordinary lock-wait deadlock
  - Select victim/abort/undo/release locks
  - Repeat for remaining cycles
  - Aborting all also works
  - Aborting all wastes more work
- Deadlock prevention versus starvation
  - Fixed resource ordering
  - Conservative 2PL prevents cycles
  - No-wait → abort incompatible request
  - No age timestamp needed
  - Timeout → abort long waits
  - Slow progress mistaken for deadlock
  - Starvation → indefinite postponement
  - Other work continues/no required cycle
  - Fair queues/sensible victim selection
  - Retain original retry timestamps
  - Deadlock freedom ≠ fairness

Wait-die and wound-wait requester/holder rules

- Wait-die versus wound-wait
  - Smaller timestamp → older
  - Identify requester/holder first
  - Wait-die older requester → wait
  - Wait-die younger requester → self-abort

- Wound-wait older requester → abort holder
- Wound-wait younger requester → wait

• Timestamp priority and correction
- Restart with original timestamp
- Relative priority grows with new arrivals
- Starvation avoidance needs finishing holders
- Fair scheduling also required
- Guide reverses wait-die incorrectly
- Younger waits/older dies incorrect
- Prevention priorities ≠ timestamp-ordering validity tests

• Older requester/younger holder
- Wait-die → requester waits
- Wound-wait → abort younger holder

• Younger requester/older holder
- Wait-die → abort requester
- Wound-wait → requester waits

Basic timestamp ordering tests

• Basic timestamp-ordering tests
- Assign TS(T)
- read_TS(X) → largest successful reader timestamp
- write_TS(X) → latest successful writer timestamp
- Read abort: TS(T)<write_TS(X)
- Otherwise read/update read_TS
- Write abort: TS(T)<read_TS(X)
- Or TS(T)<write_TS(X)
- Otherwise write/set write_TS
- Preserve timestamp order

• Timestamp example and protocol limits
- TS(T)=10/write_TS(X)=20
- Read aborts → logically later write
- Older transaction can abort
- Younger-always-aborts claim false

- Thomas rule → obsolete write ignored
- Different protocol must be stated
- No lock-wait deadlocks
- Repeated restarts possible

Optimistic control, versions, and lock granularity

- Optimistic and multiversion control
  - Private reads/computation
  - Validate read/write sets
  - Check overlapping transactions
  - Validation success → publish
  - Failure → retry
  - Uncommon conflicts favorable
  - High contention → wasted aborted work
  - MVCC → multiple versions
  - Reader uses appropriate snapshot
  - Snapshot isolation ≠ automatic serializability
  - Different-row writes can violate shared invariant
- Lock granularity tradeoffs
  - Fine row locks → independent operations
  - Many-lock tracking overhead
  - Coarse table/database → simpler management
  - Unrelated rows blocked/lower concurrency
  - Intention → planned descendant locks
  - Hierarchy
  - Escalation → many small to coarse
  - Small granularity ≠ more logical conflicts
  - Choose contention versus management overhead

Query processing before execution | Exam focus

- Query-processing phases
  - Parsing → SQL grammar
  - Semantic analysis → names/types/permissions

- Translation → logical representation
- Relational algebra/logical query tree
- Optimization → equivalent operations
- Join order/physical access methods
- Evaluation → execute plan/return results
- Phase boundaries implementation-dependent

• Query representation and scope
- Tree leaves → base relations
- Internal nodes → operators
- Graph → relations/predicates
- Physical plan → scans/indexes/algorithms
- Query processing → whole activity
- Optimization → efficient strategy selection
- DDL unnecessary before each SELECT
- Authentication→DDL→DML permutations incorrect

Heuristic and semantic query rewrites | Exam focus

• Heuristic rewrites
- Push selective filters toward bases
- Product plus join predicate → join
- Project unneeded columns away
- Retain join/filter columns
- Reorder equivalent inner joins
- Reuse common expressions when worthwhile
- Split conjunctive selections
- Apply each predicate to own input

• Semantic versus cost-based optimization
- Trusted constraints drive semantic rewrites
- Key → join cannot multiply matches
- Cost-based → estimate candidate plans
- Outer joins/NULLs/duplicates/aggregation restrict equivalence
- LEFT JOIN right filter ON→WHERE
- Preserved unmatched rows may disappear

- Fewer intermediates ≠ minimum-runtime proof

Costs, cardinalities, and selectivity | Exam focus

- Cost and cardinality estimation
  - Page I/O/CPU/memory
  - Temporary storage/network traffic
  - Catalog row/page counts
  - Distinct values/distributions
  - Selection rows ≈ s×N
  - Equijoin ≈ |R|×|S|/max(V(R,k),V(S,k))
  - Non-NULL join-key counts
  - Uniform-value assumption
  - Smaller distinct domain contained in larger
  - Adjust for NULLs
  - Empty non-NULL input → zero
  - Skew/correlation/incomplete overlap invalidate estimate
  - Known disjoint domains → zero
- Disk-time model and objective
  - Cost = seeks×seek_time + blocks×block_transfer_time
  - 5×5ms + 10×1ms = 35ms
  - Textbook local disk → I/O often dominates
  - In-memory → CPU may dominate
  - Distributed → network may dominate
  - State objective before ranking plans

Join algorithms and access paths | Exam focus

- Nested-loop page costs
  - Tuple loops → inner scan/outer tuple
  - Page loops ≈ B(R)+B(R)×B(S)
  - M buffers
  - Block loops ≈ B(R)+ceil(B(R)/(M−2))×B(S)
  - Reserve inner/output buffers
  - Try both outer relations

- Indexed, merge, and hash joins
  - Indexed loops → probe/qualifying outer tuple
  - Small outer/useful inner index
  - Sort-merge → sorted join keys
  - Include sorting cost
  - Hash → partition/build/probe
  - Equality suited/ranges generally unsuited
  - Build side fits → one-pass hash
  - Each input read once
  - Include required output/materialization costs

Compare actual query strategies | Exam focus

- Fair strategy comparison
  - Same result/same assumptions
  - Add scans/probes/sorts
  - Intermediate writes/communication
  - Consistent cost inclusion
  - Record memory limits
  - Already-ordered inputs?
  - Lowest estimated specified-objective cost
  - Compare estimates with actual execution
- Worked join-cost comparison
  - B(R)=100/B(S)=1,000/M=12
  - M−2=10 block pages
  - R outer: 100+ceil(100/10)×1,000=10,100
  - S outer: 1,000+ceil(1,000/10)×100=11,000
  - Selective R index=3 reads
  - 20 qualifying outer tuples
  - S probe including matches=4 reads
  - Indexed loops: 3+20×4=83
  - Indexed wins supplied assumptions
  - No universal join ranking

Pipelining, parallelism and caching | Exam focus

- Materialization versus pipelining
  - Materialize → store intermediate before consumption
  - Pipeline → tuples as available
  - Producer/consumer overlap
  - Reduced temporary I/O
  - Sort normally blocking
  - Hash build before probe output
  - Later probe/output can stream
  - Separate CPUs unnecessary
- Parallelism and cache distinctions
  - Multiple workers → parallel execution
  - Partitioned scans/joins
  - Independent subplans/operator pipelines
  - One-CPU interleaving → concurrency
  - Simultaneous cores → parallelism
  - Result cache → answers/invalidation
  - Plan cache → strategy, not rows
  - Buffer cache → data pages
  - Materialization → intermediate rows
  - Query-caching question needs cache kind

Indexes, statistics and re-optimization | Exam focus

- Index and storage choices
  - B/B+ trees → ordered/range
  - Hash → equality
  - Bitmap → some analytic selections
  - Clustered storage → nearby related rows
  - Index read savings
  - Space/update overhead
  - Most rows qualify → scan may win
  - Common usefulness ≠ unique index property
- Re-optimization and measurement

- Statistics/parameters/observed cardinalities trigger reconsideration
- Choose suitable plan again
- Statistics update ≠ query execution
- EXPLAIN plus measured execution
- Estimates/spills/scans/join choices
- Joins not universally faster than subqueries
- Functions can use indexes
- More indexes not always better

Security, integrity and inference control

- Security versus integrity mechanisms
  - Authentication → identity
  - Authorization/access control → permitted actions
  - Encryption → stored/transmitted content
  - Integrity constraints → valid data
  - Auditing → actions
  - View-only/no-change → restricted privileges
  - Views → visible row/column restriction
  - Privilege/execution rules must support restriction
- Inference control
  - Protected facts from allowed aggregates
  - SUM(group)−SUM(group excluding person)
  - Individual value disclosed
  - Row suppression alone insufficient
  - Query restrictions/minimum aggregate sizes
  - Appropriately designed noise
  - Aggregates-visible/details-hidden also fits views
  - Indirect-disclosure prevention uniquely identifies inference control

GRANT, REVOKE, and privilege scope

- GRANT versus REVOKE scope
  - GRANT SELECT ON Catalogue TO Mohammed
  - Read permission

- REVOKE DELETE, UPDATE (Price,Year)
- Catalogue FROM Mohammed
- Dialect must support form
- DELETE → rows/table, not columns
- UPDATE can be column-scoped
- REVOKE removes/GRANT gives permission

• Roles and effective privileges
- Roles → privilege groups
- WITH GRANT OPTION → onward delegation
- Upstream removal → CASCADE/RESTRICT possibilities
- Chosen command/DBMS determines behavior
- Other roles/grant paths may retain access
- Direct-grant removal ≠ no effective access
- View → stored query, not encryption
- Least privilege → only needed access

Failures and recovery starting resources | Exam focus

• Failure classes and starting resources
- Transaction → invalid operations/constraint violations
- Concurrency-control aborts
- Concurrency enforcement cannot prevent every failure
- System crash → volatile memory lost
- Disks may survive
- Media failure → persistent storage damage
- Restore backup/replay available archived logs
- Human error → point-in-time restoration
- Restore before bad committed operation

• Cooperating recovery mechanisms
- Restart → valid post-crash database
- Preserve durable commits/remove incomplete effects
- Rollback → failed/uncommitted work
- Redo → missing committed page effects
- Checkpoints → less log searching

- Backups → another stored copy
- Not mutually exclusive complete alternatives
- Buffer management ≠ durable backup
- Generic manager ≠ exact MySQL component
- Full backup → separate-storage database copy
- Scheduled backups plus archived logs
- Chosen recoverable-point restoration

WAL, undo and redo | Exam focus

- Write-ahead logging ordering
  - Recovery update log durable first
  - Then changed data page durable
  - Durable commit acknowledgment → required logs forced
  - Commit record included
  - Buffer change may precede log flush
  - WAL orders durable writes
  - Not every CPU assignment
- Undo versus redo models
  - Undo → incomplete-effect before-images
  - Usually backward transaction traversal
  - Redo → logged after-images as needed
  - Usually forward traversal
  - Set X to after-value → idempotent
  - Repeat X=X+10 twice → non-idempotent
  - Simple model → undo losers/redo winners
  - ARIES → repeat history then undo losers

Steal/force recovery requirements | Exam focus

- Steal and force independence
  - Steal → uncommitted page reaches disk
  - May need undo
  - No-steal → uncommitted effects stay off disk
  - Force → updated pages at commit

- No-force → committed dirty pages buffered
- May need redo

• Recovery-policy matrix assumptions
- Conventional page/log model
- Steal/no-force → undo+redo
- Steal/force → undo only
- No-steal/no-force → redo only
- No-steal/force → neither data-page operation
- After system crash
- Logging/atomic commit may remain necessary
- Independent of overwrite/separate-location updates
- Immediate update alone insufficient

• Steal recovery requirements
- Force → UNDO
- No-force → UNDO+REDO

• No-steal recovery requirements
- Force → neither data-page UNDO/REDO
- No-force → REDO

Deferred, immediate, in-place, and shadow updates | Exam focus

• Deferred versus immediate updates
- Deferred → uncommitted outside persistent database
- Usual no-force → NO-UNDO/REDO
- Redo-only = NO-UNDO/REDO
- Synonyms cannot be unique alternatives
- Immediate → persistent changes before commit
- May require undo
- write_item need not flush immediately

• In-place versus shadow updates
- In-place → overwrite original location
- Shadow/copy-on-write → new location
- Committed version preserved
- Unchanged committed page map

- Modified page map
- Commit → durable atomic root/map-pointer installation
- Required pages durable first
- Precommit crash → old map
- Alternative to conventional log recovery
- Allocation/fragmentation/concurrency costs
- Not solely recovery-time action

Checkpoints and recovery boundaries | Exam focus

- Checkpoint boundaries
  - Durable state avoids all-history scan
  - Simple quiescent/checkpoint assumptions
  - Specified flushing may remove earlier redo
  - Active transactions can span checkpoint
  - Earlier active updates may need undo
  - Before checkpoint ≠ no processing
- Fuzzy checkpoints
  - Transactions continue
  - Active-transaction/dirty-page information recorded
  - No simultaneous all-page forcing
  - Analysis may start at checkpoint
  - Redo may start earlier
  - Checkpoint ≠ full backup
  - Checkpoint ≠ active-transaction commit
  - No checkpoint/truncation → expensive log scans
  - Deferred update does not avoid scans

ARIES and repeated crashes during recovery | Exam focus

- ARIES phases and structures
  - WAL/pageLSN
  - Transaction table/dirty-page table
  - Analysis → active/loser transactions
  - Reconstruct dirty-page information

- Redo → repeat history
- Earliest relevant dirty-page update start
- pageLSN skips installed actions
- Undo → reverse losers

• CLRs and repeated-crash recovery

- Compensation log records → undo actions
- Continuation information
- CLRs redone, never undone
- Restartable recovery
- Durable undo progress retained
- No-checkpoint alone ≠ identical undo/redo lists
- Specify algorithm/durable logs/page state
- Specify recovery progress

Execute a small recovery problem | Exam focus

• Worked undo/redo recovery

- Steal/no-force/durable WAL
- Simple undo/redo algorithm
- Initially X=100/Y=50
- T1 X:100→120, durable COMMIT
- T2 Y:50→70, no commit
- Crash → T1 winner/T2 loser
- Redo X=120 if needed
- Undo Y=50 if reached disk
- No-steal → Y never persisted
- No Y undo under no-steal

• Recovery-solving procedure

- List durable transaction commit status
- Identify checkpoint type/buffer policy
- Select recovery algorithm
- Apply required redo/undo order
- Recompute final values
- Active-at-failure ≠ automatically restart/commit

- Business retry → controlled new attempt
- Uncertain commit → duplicate-effects risk
- Idempotency prevents duplicate effects

Distributed, homogeneous, and heterogeneous systems | Exam focus

- Distributed database distinctions
  - Logically related data/networked sites
  - Coordinated management
  - Homogeneous → compatible DBMS technology
  - Heterogeneous → differing DBMS/model/schema/interface
  - Operating systems do not define distinction
  - Same DBMS ≠ identical local schemas
  - No automatic replication/immediate synchronization
- Heterogeneous reconciliation and distribution costs
  - Naming/types/units/identifiers
  - Query capabilities/transaction semantics
  - ODBC/JDBC → connectivity interfaces
  - No automatic semantic reconciliation
  - No automatic distributed transaction
  - Potential locality/expansion/replicated availability
  - Communication/coordination/security costs
  - Outage without usable replica → unavailable data

Architectures and local autonomy | Exam focus

- Distributed architecture responsibilities
  - Client → requests
  - Server → shared-data management/operations
  - Supported integrity/transaction enforcement
  - Collaborating servers → cooperative subqueries/transactions
  - Middleware → request translation/data location
  - Operation coordination/result combination
  - Independent servers

- Responsibilities ≠ fragmentation types

• Integration versus local autonomy

- Tight integration → global schema/shared control
- Federation → autonomous databases/mappings
- Loose multidatabase → more independence
- May lack complete global schema
- Specify schema/transaction/administrative autonomy
- Heterogeneous ≠ surrender all autonomy
- No requirement to copy every row

Distribution transparency | Exam focus

• Distribution transparency types

- Location → site hidden
- Fragmentation → fragments/reconstruction hidden
- Query whole logical relation
- Replication → chosen copy/maintenance hidden
- Single logical database may combine all

• Visible fragments versus hidden locations

- Explicit fragment name → fragmentation visible
- Unnamed site → location possibly transparent
- Specific-fragment stem needs clarification
- Column subsets → vertical fragmentation
- Split itself ≠ transparency type
- Hidden column split → fragmentation transparency

Fragmentation: correctness and reconstruction | Exam focus

• Fragmentation correctness

- Horizontal → predicate row subsets
- Union reconstructs relation
- Completeness → every original row represented
- Disjointness if overlap forbidden
- Explicit NULL/other-value fragment when needed
- Vertical → projected column subsets

- Repeat key/row identifier
- Key join reconstructs original rows
- No spurious matches

• Hybrid reconstruction and availability
- Combine row/column splits
- Either suitable split order
- Reconstruct by reversing operations
- Vertical pieces → key joins
- Horizontal pieces → unions
- FULL OUTER JOIN not universally required
- Partitioning → locality/parallelism
- Partitioning alone ≠ spare copy
- Only-fragment-site loss → fragment unavailable
- Other sites may continue

Replication policies and their costs | Exam focus

• Replication extent and backup distinction
- Full → whole database/relevant sites
- Partial → selected fragments/selected sites
- Nonredundant → one copy/fragment
- Read locality/possible availability gains
- Storage/update-propagation costs
- Mistakes can propagate
- Replication ≠ independent backups

• Replication timing and transfer policies
- Eager/synchronous → propagation before commit/ack point
- Higher update latency
- Lazy/asynchronous → later propagation
- Stale reads/conflict resolution
- Snapshot → periodic copy
- Incremental → changes only
- Policy dimensions can coexist
- Commonly-used stem not unique

- Full replication ≠ simultaneous updates
- State synchronization rule

Distributed plans and semijoin arithmetic | Exam focus

- Distributed query planning
  - Logical query decomposition
  - Localize relations → fragments/replicas
  - Global placement/order
  - Optimize local subplans
  - Bytes/messages/latency/bandwidth/local work
  - Equivalent filters/projections before shipping
  - Semijoin → matching rows from one input
  - Only that input's attributes returned
- Worked semijoin communication
  - R at A=100MB
  - S at B=20MB
  - Result required at A
  - Ship all S → 20MB
  - Send distinct R keys → 0.2MB
  - Return matching S → 1MB
  - Semijoin bytes=0.2+1=1.2MB
  - Other costs excluded
  - Extra messages/high latency may dominate
  - Nearly all S matches → little benefit

Distributed atomic commit and Oracle roles | Exam focus

- Two-phase commit protocol
  - Prepare → durable commit promise
  - All yes → coordinator durably records commit
  - Then inform participants
  - Otherwise decide abort
  - Prepared participant retains decision-following state
  - Coordinator/communication failure → in-doubt blocking

- Wait until decision learned
- Post-prepare timeout ≠ invented commit/abort

• Atomic commit versus isolation
- 2PC → atomic outcome
- Locking/other concurrency protocols → isolation
- Oracle COMMIT_POINT_STRENGTH
- Higher eligible strength → commit point site
- Oracle-specific 2PC role selection
- Not lock phase/checkpoint setting
- Coordinator → decision coordination
- Not every site's physical row storage

Construct homogeneous and heterogeneous designs | Exam focus

• Homogeneous three-branch bank design
- Compatible DBMS
- Horizontal accounts by BranchID
- Home branch owns account
- Replicate selected read-mostly reference data
- Cross-branch → global identifiers
- Coordinated commit/stated isolation
- Replica placement meets specified failure service
- Verify reconstruction/capacity/latency/recovery

• Heterogeneous merger design
- Different branch DBMSs retained
- AccountID/currency/time-zone/precision mappings
- Middleware/gateways for supported operations
- All engines support atomic distributed commit?
- Otherwise explicit compensating workflow
- State intermediate-state limits
- Autonomy versus coordination/availability
- Fact ownership/copy placement
- Failed-link permitted behavior
- Conflicting-write resolution

Warehouses, OLTP, OLAP, and columnar storage

- OLTP, OLAP, and warehouses
  - OLTP → frequent operational transactions
  - Account updates
  - OLAP → large-data scans/summaries
  - Warehouse → integrated historical analysis
  - Extraction/transformation/loading
  - Workload tendencies, not absolute restrictions
  - OLTP may retain history
  - Warehouses may be updated
- Warehouse schemas and storage layouts
  - Star → declared fact grain
  - Fact measurements linked to dimensions
  - Time/product/store dimensions
  - Snowflake → normalized dimensions
  - Columnar → values grouped by column
  - Fewer unnecessary column reads/compression
  - Row-oriented → record fields grouped
  - Physical layout ≠ relational-model determination
  - Indexes/denormalization depend on workload

NoSQL families, wide columns and CAP

- NoSQL families
  - Key-value → retrieve by key
  - Document → structured documents
  - Wide-column/column-family → sparse rows
  - Related column families
  - Graph → nodes/edges/relationship traversal
  - Cassandra wide-column example
  - Not interchangeable with analytic relational columnar engines
  - Flexible schema ≠ absent schema rules
  - NoSQL ≠ no transactions
- CAP under network partitions

- Cannot guarantee both linearizable consistency/availability
- Every request at nonfailed nodes
- Partition condition essential
- Not casually pick-any-two forever
- Failure behavior choices
- May vary by operation
- CAP consistency → single-copy visibility/order
- ACID consistency → application invariant preservation

Data sources and association rules

• Structured versus unstructured data
- Structured → explicit fields
- Registrar relational tables
- Unstructured → free text/images/audio
- Web/search collections may mix both
- DBMS can store text/BLOBs
- Storage product ≠ all-structured contents

• Association rules and mining contrasts
- X⇒Y → co-occurrence
- Support=count(X and Y)/N
- Confidence=count(X and Y)/count(X)=P(Y|X)
- 75/100 shoe purchases include socks
- Confidence(shoes⇒socks)=0.75
- No causation proof
- Support needs total transaction count
- Clustering → similarity/no target labels
- Classification → categories
- Regression → numeric outcomes
- Purchasing co-occurrence → association mining
- Store-layout use does not change task

Data-mining workflow and evaluation

• CRISP-DM workflow

- Business understanding first
- Decision/success criteria
- Data understanding → preparation
- Modeling → evaluation → deployment
- Iterative process
- Preparation → cleaning/integration/feature selection
- Business problem precedes preparation

• Model evaluation and leakage
- Appropriate data split
- Train on training data
- Tune on validation
- Evaluate once on held-out test
- Before deployment
- Testing/evaluation labels can overlap
- Training→testing→evaluation→deployment needs convention
- Model-selection evaluation may precede final testing
- Deployment monitoring
- Return to preparation/modeling when needed
- No future/test outcome leakage

Repair wording before trusting answer keys

• Overlapping answer mechanisms
- ARIES/deferred update both log-based
- Restart → checkpoints/rollback/redo cooperation
- NO-UNDO/REDO = redo-only
- Best/important/common/faster → criterion/workload
- Literal scope over familiar-term uniqueness

• Minimum missing assumptions
- Vendor behavior → exact DBMS/version
- Constructors → OID/type convention
- Waiting → requester/holder ages
- Recovery → durable log/page state/policy
- Distribution → fragments/network cost

- Multiple true descriptions → genuine ambiguity
- Index → question-specific clarifications/corrections
- No whole-question reproduction needed

Object Oriented Programming

Revision Outlines

Objects, classes and responsibilities | Exam focus

- Object versus class
  - Identity, state, behavior
  - Fields store state
  - Methods provide behavior
  - Rectangle: width, height
  - getArea(), getPerimeter()
  - Equal fields, distinct identities
  - Class: common structure and behavior
  - Object: particular instance
- Responsibility-driven classes
  - Account controls balance
  - Transaction records payment
  - Service coordinates objects
  - Messages: method calls, obj.add()
  - Delegation to other objects
  - Objects manage own state
  - Procedural and OO reuse
  - Procedural global shared data optional

Encapsulation, hiding and abstraction | Exam focus

- Encapsulation, hiding and abstraction
  - Encapsulation: state/operations grouped, access controlled
  - Information hiding: implementation behind stable interface
  - Abstraction: essential operations, irrelevant details omitted
  - private balance plus deposit(amount)
  - Overlapping principles, question emphasis matters
- Inheritance and polymorphism
  - Inheritance: subtype relationship, possible implementation reuse

- Common type, concrete-specific behavior
- Fields/methods wrapping: encapsulation
- What versus how: abstraction
- Precise mechanism over vague slogan

References, instances and value passing | Exam focus

- Reference declaration versus object creation
  - Student s;: reference only
  - new Student(): allocate and initialize
  - s.name, s.study(): visibility-dependent access
  - t=s: reference copy
  - Shared object aliasing
  - Mutation through t visible through s
- Java value passing and reachability
  - All arguments passed by value
  - Copied reference permits shared-object mutation
  - Parameter reassignment leaves caller reference unchanged
  - Unreachable objects reclaimable
  - Collection timing unspecified
  - Reachable cycles may retain objects
  - Unreachable cycles collectible

UML class and instance notation | Exam focus

- UML class box
  - Name, attributes, operations
  - + public, - private, # protected
  - name:String
  - getName():String
  - Student(String,String,String,int)
  - Three strings plus integer
  - Age "25": wrong argument type
- UML instance notation
  - Underlined instanceName:ClassName

- sara:Student
- Class notation: type
- Instance notation: particular object
- Optional slot values
- Static-feature underlining: different meaning
- Whole notation determines interpretation

Variables, defaults and initialization

• Local, instance and static variables
- Locals: methods, constructors, blocks
- Definite assignment before use
- Locals: no public/private/protected
- Fields: class, outside method bodies
- Instance fields: per object
- Static fields: per class
- Location alone doesn't distinguish field kinds

• Java defaults and initialization
- Fields and array elements defaulted
- Numeric zero, char '\u0000'
- boolean false, references null
- Unassigned local int t: compilation error
- int a=10,b=10: initial values
- final local permitted
- final isn't access modifier

Primitive types and conversions

• Java primitive and reference types
- byte, short, int, long
- Signed widths: 8,16,32,64 bits
- byte: -128..127
- short: -32768..32767
- float, double, char, boolean
- char: unsigned 16-bit UTF-16 unit

- Digit, letter, symbol, not whole String
- enum and String: reference types

• Conversions and arithmetic promotion
- Compatible widening often implicit
- Narrowing usually explicit, possibly lossy
- int→float widening may lose precision
- Widening-losslessness claim false
- char/byte/short often promoted to int
- boolean ↔ numeric conversion forbidden
- if(0), numeric-zero loop condition: invalid

Static members and final

• Static class members
- One field per loaded class
- Shared across instances
- Three constructor increments: count=3
- Prefer ClassName.count, ClassName.utility()
- Static method: no current instance
- Explicit object permits instance-member access

• Static initialization and final
- Class initialization before relevant active use
- Not merely file loading
- final variable: no reassignment
- final method: no overriding
- final class: no extension
- final reference may target mutable object
- PI: static final candidate
- Student name: normally instance state
- Parameter-only calculation: possible static method

Access control and packages

• Java access boundaries
- private: declaring class/nest

- No modifier: package access
- protected: package plus qualified subclass access
- public: widest ordinary access
- Declaring type must be accessible
- Top-level class/interface: public or package-private
- Top-level private/protected forbidden
- Member types: additional visibility possibilities

• Protected access across packages
- Own inherited context allowed
- Appropriate subclass-typed receiver allowed
- Arbitrary superclass receiver disallowed
- Same-package unrelated classes also allowed
- protected ≠ subclasses only
- Own main accesses own instance's private fields

Constructors and generated defaults | Exam focus

• Constructor identity and purpose
- Exact class name
- No return type, including void
- Initialize new object
- Overloaded/private constructors permitted
- Private: factory or controlled singleton
- Internal object creation still possible

• Default-constructor conditions
- No declared constructor → compiler default
- Circle(double r) suppresses generated no-arg
- new Circle(): explicit no-arg then required
- Constructors neither inherited nor overridden
- return; allowed, return expression forbidden
- Field defaults precede ordinary constructor initialization
- Defaults independent of generated constructor

Constructor chains and initialization order | Exam focus

- Java17 constructor delegation
  - this(args)/super(args): first statement
  - this: same-class constructor
  - super: immediate parent constructor
  - Cannot explicitly invoke both
  - Absent explicit delegation: implicit super()
  - Accessible matching no-arg parent required
- Construction order and version conditions
  - Parent construction first
  - Subclass field initializers/instance blocks in order
  - Relevant subclass constructor body last
  - FinalExam parent: num1=4
  - Subclass initializer: num=3
  - this(5): num=25
  - No-arg body: num1=num=25
  - Java25: restricted pre-invocation statements
  - First-statement rule requires version qualification

Methods, accessors and mutators | Exam focus

- Method structure and invocation
  - Modifiers, return type, name
  - Parameter list and body
  - public int getAge(){return age;}
  - void: no returned value
  - Parameters: declared inputs
  - Arguments: call-supplied values
  - object.method(): instance call
  - Implicit this in suitable instance context
- Accessors, mutators and validation
  - Getter: return information
  - Setter: change state
  - Useful operations over blanket setters

- Validate state changes
- deposit: reject negative amounts
- Separate subclass: private parent field inaccessible
- this.field doesn't bypass access
- Use permitted parent operation

Overloads and compile-time selection | Exam focus

- Method overload distinctions
  - Same name, different types/count/type order
  - Return type alone insufficient
  - Java and C++ rule
  - Inherited methods can form overloads
  - Same-class-only claim simplified
  - f(int), f(double), f(int,int)
  - Parameter-name change: no overload
- Compile-time overload selection
  - Declared argument types and conversions
  - Exact int preferred over double widening
  - Return type outside ordinary Java signature
  - Different signature versus inherited-signature override

Inheritance and composition | Exam focus

- Class inheritance forms
  - Dog extends Animal
  - Subclass/derived/child, superclass/base/parent
  - One direct superclass
  - Multiple implemented interfaces
  - Single: one parent
  - Multilevel: inheritance chain
  - Hierarchical: siblings share parent
  - extends Animal,Mammal: invalid Java
- Contract-based inheritance versus composition
  - Subtype satisfies parent contract

- Dog is an Animal
- Pet: common name and speak contract
- Dog: specialized fetch
- Dog/Cat: appropriate speak implementations
- Composition: has-a
- Car contains Engine
- Reuse alone doesn't justify is-a

Overriding rules | Exam focus

- Override eligibility and restrictions
  - Inherited instance method, same signature
  - Compatible return type
  - Covariant reference return subtype
  - Visibility cannot narrow
  - Checked exceptions cannot broaden incompatibly
  - @Override: compiler-checking annotation
  - No Java override keyword
  - Constructors/final/static/private methods: no overriding
- Independent declaration versus override
  - Private-parent same signature: independent child declaration
  - No runtime overriding
  - No ordinary static-method hiding
  - Concrete subclass implements inherited abstract method
  - Unrelated same-name methods: no overriding

Declared types and dynamic dispatch | Exam focus

- Declared type versus runtime class
  - Animal a=new Dog()
  - Animal: declared, Dog: actual
  - Declared type controls compilable operations
  - Dog-only fetch: a.fetch() fails
  - Eligible sound override: Dog.sound() runtime
- Dispatch-solving sequence

- Declaration and visibility first
- Compile-time signature second
- Runtime class and most-specific override third
- Animal[]{Dog,Cat}: common speak loop
- New subclasses supply behavior
- Generic loop unchanged: extensibility

this, super and hiding

- this, super and field selection
  - this: current object
  - this.name=name: field versus parameter
  - super.name: hidden parent field
  - super.sound(): parent implementation
  - No second parent object
  - Unqualified instance call may dispatch dynamically
  - Explicit super call differs
- Field and static-method hiding
  - Fields aren't dynamically overridden
  - Same-signature static methods hide
  - Animal a=new Dog(): static resolves Animal
  - Prefer Animal.staticMethod()
  - Static context: no current this/super

C++ this and object rules

- C++17 this pointer
  - Ordinary non-static member function
  - Pointer to called object
  - this->field: member selection
  - Implicit object argument explanation
  - Member variables don't receive argument
  - Static member function: no this
- C++ versus Java object rules
  - Base-pointer/reference dispatch requires virtual

- C++: multiple class inheritance
- Java: multiple interface types
- C++ object-value assignment may copy object
- Java class-variable assignment copies reference
- Check dispatch and memory-management assumptions

Abstract classes and implementations | Exam focus

• Abstract class capabilities
- No direct instantiation
- Fields, constructors, concrete methods allowed
- Abstract methods allowed, not required
- Abstract method: no body
- abstract plus final: conflicting requirements
- Required versus forbidden overriding

• Concrete implementation obligations
- Provide/inherit all required abstract implementations
- Otherwise subclass must remain abstract
- Required: String welcome()
- int welcome(): incompatible return
- welcome(int): overload only
- Valid implementation plus unrelated overload allowed

Interfaces and modern Java | Exam focus

• Interface contract and implementation
- Payable: double getPaymentAmount()
- Implicit public abstract operation
- Invoice implements Payable
- Required public double getPaymentAmount()
- getQuantity()*getPricePerItem()
- Class: superclass plus zero/multiple interfaces
- Interface extends multiple interfaces
- No instance fields or constructors
- Fields: public static final

- Modern interface method rules
  - No private/default/static: public abstract
  - default/static methods may have bodies
  - Java9+: private helpers
  - All-interface-methods-bodyless claim outdated
  - All inherited abstract contracts satisfied
  - Suitable public implementation may be inherited
  - No direct interface instantiation
  - References hold implementing objects
  - Anonymous implementations
  - Suitable functional-interface lambdas

Defaults, conflicts and class choice

- Abstract class versus interface choice
  - Abstract class: shared state/construction/implementation
  - Interface: unrelated-class capability
  - Multiple type inheritance
  - Concrete class method beats interface default
  - Inherited abstract class declaration may prevail
  - Concrete subclass may need explicit implementation
- Interface conflicts and diamond ambiguity
  - Unrelated conflicting defaults: explicit resolution
  - Interfaces don't eliminate all diamond ambiguity
  - Same compatible abstract signature: one public implementation
  - Conflicting return types/contracts may be impossible
  - Abstract versus implemented status matters
  - Signature compatibility determines answer

Casts and instanceof

- Upcasting versus downcasting
  - Implicit upcast leaves object unchanged
  - Downcast: compile-time acceptance, runtime risk
  - Car a=new Car()

- (Ferrari)a: ClassCastException
- Later drive()/brake() calls unexecuted
- Cast doesn't transform object

• Safe narrowing and null
- instanceof before uncertain downcast
- null instanceof Ferrari: false
- Casting null yields null
- Dereferencing null fails
- Object obj="Hello": (String)obj succeeds
- (Integer)obj: ClassCastException
- 'Hello': invalid Java literal
- Clarify single-quoted bank version first

Identity, equality and immutable strings

• Identity, equality and representation
- Reference ==: identity
- equals: class equality policy
- Object.equals: identity unless overridden
- String.equals: character content
- Equal objects require equal hashCode
- toString: textual representation, not equality

• Immutable String and text APIs
- text += " World": new concatenated result
- Hello World without original String mutation
- final String class: no subclassing
- final alone doesn't ensure arbitrary immutability
- charAt(index): char
- length(): UTF-16 units
- substring(begin,end): exclusive end
- Character.toUpperCase(ch): character conversion
- matches(String regex): regular-expression parameter

Numeric, bitwise and Boolean expressions

- Java expression tracing
  - Left-to-right operand evaluation with precedence
  - n=6,m=15,p=3: n-=m,n*=p → -27
  - 6-2+10%4+7=13
  - 8-2*5*(3+9/3)+6/3=-50
  - Integer 6/4=1
  - Double 6.0/4=1.5
  - Java % accepts floating operands
  - C++ % differs
- Bitwise and Boolean operations
  - Binary-column method
  - 11&9=9
  - 9^3=10
  - 10|12=14
  - &&, ||: Boolean short circuit
  - Boolean &, |: no short circuit
  - Printed true: lowercase
  - Neither 1 nor TRUE

Increment order and concatenation traps

- Increment and character-space trap
  - x=5: x++ then ++x → 5,7
  - y=++x*x--: 6*6=36
  - Final x=5
  - x + ' ' + y: arithmetic
  - Space char value=32
  - println result: 73
  - x + " " + y: separation
- String concatenation grouping
  - Left-to-right + groups
  - String operand starts concatenation
  - 1+2+"x" → "3x"

- "x"+1+2 → "x12"
- Single-quoted space doesn't separate values
- String concatenation and evaluation order

Control-flow traces and exact output

• Control flow and exact printing
- while/for: pre-test, possibly zero
- do-while: post-test, at least once
- break: exit loop
- continue: remaining body skipped
- for continue then update
- Colon switch: fall-through without break
- Choice 1: Hello then you
- break then outside Goodbye
- default skipped

• Execution counts versus printing
- i=0..4: conditional even-i print
- Unconditional print(i+1)
- Exact output: "0 122 344 5"
- Sum j=1..10: 55
- if(a) then else if(a&&b)
- Second condition unreachable as true
- Reached else implies a=false

Array creation, bounds and ragged rows

• Array declarations and component types
- int[] a=new int[5]: valid
- int a[]={1,2,3}: valid
- int[] a={}: zero-length
- Declaration initializer needn't use new
- int a[],b: only a array
- int[] c,d: both arrays
- Object length fixed after creation

- Reference may be reassigned
- Common declared component type
- Superclass/interface component type permitted

• Bounds and ragged arrays
- Valid indices: 0..a.length-1
- Invalid access: ArrayIndexOutOfBoundsException
- Two-dimensional: arrays of arrays
- Ragged {{12,-9,8},{7,14},{-32,-1,0}}
- Row lengths: 3,2,3
- No portable physical-contiguity guarantee
- No raw-address assumptions

Array algorithms, sorting and copying

• Array sum and average
- Four elements: a[i]=i*3
- Sum: 0+3+6+9=18
- Average: sum every element
- Floating division by length
- Zero length handled separately
- Representation/printing versus rounded result
- Bank mean: 98.8/6≈16.46666667
- Not exact printed-string assertion

• Array sorting, copying and aliasing
- java.util.Arrays.sort(a): in-place
- System.arraycopy(src,srcPos,dst,dstPos,length)
- Existing destination array
- arraycopy: lowercase c
- Arrays.copyOf(a,n): new copy
- a=b: same-array alias
- Random access: reliable advantage
- Size bounded by resources/implementation

Packages, imports and Java tools | Exam focus

- Packages and import order
  - package p; before imports
  - Imports before top-level declarations
  - import p.*: accessible p types
  - No automatic p.sub imports
  - Name organization and access control
  - Subpackages gain no special access
  - java.lang: implicit import
  - java.util: utilities
  - java.io: I/O
  - java.awt: Button, Checkbox
- Compilation, launch and identifiers
  - javac Test.java → bytecode
  - java Test → class launch
  - Modern java Test.java: source-file launch
  - Unrestricted run question: multiple answers possible
  - Public top-level type: matching filename
  - Class/method capitalization: conventions
  - Identifiers case sensitive
  - $ allowed, @ forbidden

Exception hierarchy and checked obligations | Exam focus

- Throwable hierarchy and source errors
  - Throwable → Exception, Error
  - RuntimeException/Error subclasses: unchecked
  - Other throwable classes: checked
  - IOException, FileNotFoundException, SQLException: checked
  - ArithmeticException, NullPointerException: unchecked
  - ClassCastException, array bounds exceptions: unchecked
  - Missing semicolon/undeclared variable: source errors
  - Source errors aren't checked exceptions
- Checked obligations versus runtime throwing

- Compiler checks catch-or-declare obligation
- Actual exception thrown at runtime
- Custom extends Exception or RuntimeException
- Error: normally serious conditions
- Routine recovery usually inappropriate
- Catching Error technically possible

throw, throws and catch selection | Exam focus

• throw versus throws
- throw new IOException("message"): active throw
- throws IOException: signature declaration
- Caller catches or propagates checked obligations
- try: guarded operations
- catch: matching thrown type/subtype
- Throw skips remaining try statements

• Catch selection and propagation
- Source-order handler selection
- FileNotFoundException before IOException
- IOException before Exception
- General-first subtype catch: unreachable, compilation failure
- No match: outward propagation
- Uncaught exception terminates its thread
- Multiple catch blocks allowed
- catch(IOException | SQLException e)
- Multi-catch alternatives cannot subclass each other

finally, cleanup and diagnostics

• finally and pending control transfer
- Normally after try/selected catch
- Return and propagation included
- JVM/process exit or crash: exception
- Never leaving try: exception
- Return expression evaluated before finally

- finally return overrides pending return
- Potential exception suppression
- Avoid returning from finally

• Resource cleanup and diagnostics

- try-with-resources: closes AutoCloseable
- Exceptional exit cleanup
- getMessage: message
- toString: normally type and message
- printStackTrace: exception and recorded call frames
- Test success, caught failure, propagation
- Handling doesn't replace validation
- No automatic logic correction

Streams and GUI events | Exam focus

• Streams and GUI events

- InputStream/OutputStream: bytes
- Reader/Writer: text characters through encoding
- Deterministic resource closure
- read()=-1: end-of-stream
- Event source notifies registered listeners
- Button → ActionListener.actionPerformed
- Swing updates: event dispatch thread
- Short handlers preserve responsiveness

JDBC queries and transactions | Exam focus

• JDBC query lifecycle

- Connection → prepare SQL → bind values
- Execute → process results → close
- PreparedStatement: SELECT name FROM Student WHERE id=?
- ps.setInt(1,id): first parameter
- executeQuery → ResultSet
- rs.next() before column reads
- executeUpdate → affected-row count

- SQL binding, transactions and responsiveness
  - Bound values separate from SQL syntax
  - Catch/declare SQLException
  - Multiple writes: disable auto-commit
  - Success: commit
  - Failure: rollback
  - Slow database work off GUI thread
  - Display through appropriate UI mechanism

Historical applet lifecycle

- Historical applet environment and lifecycle
  - Java embedded in HTML
  - Compatible JVM host/plugin or applet viewer
  - init: initialize once
  - start: after initialization, resume
  - stop: host deactivation, suspend
  - destroy: final cleanup before unloading
  - paint: drawing, not constructor
- Applet historical interpretation limits
  - Historical bank concepts
  - Not modern browser deployment instructions
  - destroy not exclusive to browser shutdown
  - Host may destroy on unloading
  - Abrupt termination: cleanup unguaranteed
  - JVM alone insufficient for browser embedding

Android Activity and Intent

- Android interaction and communication components
  - Activity: user-interaction entry point
  - Often corresponds to screen
  - Intent: requested action/explicit target
  - Extras carry inter-activity values
  - putExtra("id",17): destination reads ID

- Content provider: structured-data sharing
- Broadcast receiver: broadcast handling
- Neither substitutes parameterized activity launch

Android declarations and packaging

- Manifest and layout containers
  - AndroidManifest.xml: essential app metadata
  - Components, permissions, intent filters
  - Manifest isn't executable code
  - Not all source/layout details
  - LinearLayout, RelativeLayout derive ViewGroup
  - android.view.ViewGroup contains child views
- Android packaging distinctions
  - .apk: Android package archive
  - Installation and runtime contents
  - Android Package Kit: common exam expansion
  - Documentation: Android package
  - Manifest describes package
  - Layout resources describe views
  - Old Android questions outside Java module

Malformed syntax and trace corrections

- OCR repair before literal tracing
  - extends misread as Extends
  - println misread as printIn/printin
  - Constructor names, braces, <=, == corrupted
  - Case-sensitive Java may fail compilation
  - Intended trace requires stated repair
  - Even i=2,4,6,8,10; i%4==0
  - Only 16 and 64 print
  - Extra offered values incorrect
- SS118 fragments and numeric output
  - Recovered public String welcome() implementation

- Additional welcome(int) overload
- Current A-D fragments unreliable
- Required signature over fabricated letter
- Rounded mean choice ≠ exact printed output

Ambiguous bank interpretations

- Overlapping single-choice options
  - Static constant and parameter-only utility suitable
  - Multiple valid array declarations/initializers
  - Abstract class may implement complete interface
  - Encapsulation/abstraction overlap
  - Specific requested emphasis needed
- Interpretation-dependent bank claims
  - Interface: multiple type inheritance plus abstraction
  - Broad purpose choices overlap
  - All-public-abstract: restricted/legacy interpretation
  - protected includes same-package access
  - No overriding keyword
  - Compatible declarations establish overriding
  - @Override: annotation
  - C++ hidden object: non-static member functions
  - Not arguments passed to member variables

Language classifications and portability

- Language taxonomy and portability
  - C++, Java, Python, Perl: third generation
  - Taxonomy separate from object orientation
  - Taxonomy separate from compilation/interpretation
  - Java source → JVM bytecode
  - Compatible JVM plus libraries: cross-system portability
- Portability qualifications and language properties
  - Compatible APIs, versions, environment assumptions
  - No unconditional any-device guarantee

- Java objects/classes plus primitives
- Learning difficulty: subjective judgment
- Syntax conventions versus bytecode portability
- Runtime implementation distinguished

Computer Organization and Architecture

Revision Outlines

Architecture and system structure | Exam focus

- Architecture versus organization
  - Programmer-visible instructions and registers
  - Data types and address spaces
  - I/O conventions
  - Implementation: signals, buses, pipelines, memory
  - Same architecture, different organizations
  - No definition-imposed maximum lifetime
  - Microarchitecture: internal processor design
- System structural components
  - CPU, memory, I/O, interconnection
  - CPU: ALU, control unit, registers
  - Microcontroller: integrated control system
  - Microcontroller ≠ mandatory CPU subunit
  - Motherboard: physical connections
  - Gates: fundamental logic building blocks
  - Missing gates option: ambiguous answer

Stored programs and software | Exam focus

- Von Neumann versus Harvard
  - Shared addressed instructions and data
  - Sequential execution unless control changes
  - Harvard: separate memories or paths
  - Unified address space, split caches
  - Modern architecture: nonexclusive labels
  - Separate regions ≠ Harvard
- Software roles
  - OS: resources and services
  - Applications: user tasks

- Compiler: source→target code
- Assembly: symbolic machine instructions
- JIT: compilation during execution
- Intermediate-code translation
- Before execution ≠ uniquely JIT
- Machine language: registers, types, addressing

Gates and truth tables

- Logic gate behavior
  - AND: all inputs 1
  - OR: any input 1
  - NOT: bit complement
  - NAND: complemented AND
  - NOR: complemented OR
  - XOR: two inputs differ
  - XNOR: two inputs agree
  - Enabled buffer: input copied
  - Output bubble: inversion
  - AND shape plus bubble: NAND
- Truth-table construction
  - n inputs → 2^n rows
  - Complements before products before sums
  - Boolean + means OR
- Worked example Boolean evaluation example
  - F=x+y′z; x=0,y=1,z=1
  - y′=0 → F=0
  - Four inputs → 16 rows

Boolean reduction and universal gates

- Boolean identities
  - A+0=A; A·1=A
  - A+A=A; AA=A
  - A+A′=1; AA′=0

- A+AB=A; A(A+B)=A
- De Morgan transformations
- (AB)′=A′+B′
- (A+B)′=A′B′
- Truth-table equivalence check

• Universal gates and Karnaugh maps
- NAND alone; NOR alone
- Tied inputs construct NOT
- Complemented gates construct AND/OR
- Gray-code ordering
- Adjacent power-of-two rectangular groups
- Wraparound and overlap allowed
- Cover every required 1
- Eliminate changing variables
- No diagonal groups or required 0s

• Worked example Reduction and NAND construction
- AB+AB′=A(B+B′)=A
- A NAND A=A′
- NAND then inversion → AND

Combinational and sequential circuits

• Combinational versus sequential
- Present inputs plus propagation delay
- Adders, muxes, demuxes
- Encoders and decoders
- Combinational: no stored logical state
- Sequential: stored state plus inputs
- Registers, counters, flip-flops
- Shift register: sequential criterion

• Powered state retention
- Flip-flop: one bit
- Retention ≠ nonvolatility
- Flip-flops/SRAM lose unpowered state

- Clocked updates: designated clock events
- Combinational logic: no inherent clock

Flip-flops and counter traces

- Latch and flip-flop transitions
  - Active-high NOR SR latch
  - SR=10: set; 01: reset
  - SR=00: hold; 11: forbidden
  - Active-low NAND conventions differ
  - D: active-edge input copy
  - JK=00: hold; 01: reset
  - JK=10: set; 11: toggle
  - T=1: toggle; T=0: hold
- Sequential state tracing
  - Given initial state
  - One clock at a time
  - k-bit counter: 2^k states
  - Modulo 2^k wraparound
  - Extra control may alter sequence
  - Ripple: asynchronous propagation
  - Synchronous: shared clock
- Worked example Counter and T examples
  - 3-bit: 110→111→000
  - Initial T-flop state: 0
  - T inputs 1,0,1 → 1,1,0

Adders, selectors and converters

- Adder equations
  - Half-adder sum=A XOR B
  - Half-adder carry=AB
  - Full-adder includes Cin
  - Sum=A XOR B XOR Cin
  - Carry=AB+Cin(A XOR B)

- Cascading: word-wide carry propagation

• Data selection and encoding
  - 2^k-to-1 mux: k selects
  - Demux: one input→selected output
  - k-to-2^k decoder
  - Decoder enable and active level
  - Ordinary encoder: one active input
  - Priority encoder: multiple-input winner
  - Registers: storage, not combinational converters

• Worked example Adder and mux examples
  - A=B=Cin=1 → binary 11
  - Sum=1; carry=1
  - 8-to-1 mux: 3 selects

Bases and binary codes

• Positional representation
  - Digit value=digit×b^i
  - Binary digits: 0-1
  - Octal digits: 0-7
  - Hex digits: 0-9,A-F
  - F=15
  - Integer: repeated division by 2
  - Remainders read upward
  - Fraction: repeated multiplication by 2
  - Integer parts read downward

• Grouping and encodings
  - Octal: groups of three bits
  - Hex: groups of four bits
  - Group outward from radix point
  - BCD: four bits per decimal digit
  - BCD 12: 0001 0010
  - Binary 12: 1100
  - ASCII: character code, not I/O interface

- Worked example Base-conversion examples
  - 1011_2=8+2+1=11
  - $2F_{16}$=47
  - 101.101_2=5.625
  - 13: remainders 1,0,1,1
  - Reverse remainders → 1101_2

Signed arithmetic and overflow

- Fixed-width integer representations
  - Unsigned: 0…2^n−1
  - Two’s complement: −2^(n−1)…2^(n−1)−1
  - Negation: invert n bits, add one
  - A−B=A+two’s-complement(B)
  - Retain n result bits
  - Signed magnitude: two zeros
  - One’s complement: two zeros
  - Two’s complement: one zero
- Carry versus signed overflow
  - Unsigned carry ≠ signed overflow
  - Same-sign inputs, opposite-sign result
  - Fixed width throughout
- Worked example Four-bit arithmetic examples
  - 7−3: 0111+1101=1 0100
  - Discard carry → 4
  - 0111+0001=1000: signed overflow
  - +8 outside −8…+7

Floating point and arithmetic pipelines

- Binary floating-point representation
  - Sign×significand×2^exponent
  - Binary32 normal: (−1)^s(1.f)2^(e−127)
  - Binary64: 11 exponent bits
  - Binary64 bias: 1023

- Special patterns: zero/subnormal
- Special patterns: infinities/NaNs
- Implicit leading 1: not universal

• Floating addition sequence
- Compare exponents
- Shift smaller significand: alignment
- Add/subtract, normalize, round
- Alignment loses low-order bits
- Overflow: exponent range
- Rounding error: finite precision
- Arithmetic pipeline: overlapping operand pairs
- All steps ≠ one cycle

• Worked example Floating alignment example
- $1.10_2 \times 2^3 + 1.00_2 \times 2^1$
- Align: $(1.10_2 + 0.01_2) \times 2^3$
- Normalize: $1.11_2 \times 2^3 = 14$

Register transfers and memory cycles | Exam focus

• Register transfer semantics
- R2←R1: copy, preserve R1
- P:R2←R1: conditional clocked transfer
- Simultaneous transfers: old values
- Sequential transfers: newly written values
- Common bus: one selected source
- Load signals: selected destinations

• Memory read/write microoperations
- Read: address→MAR/AR
- Assert read; wait for memory
- Capture→MDR/MBR/DR
- Write: address plus data
- Assert write
- ALU: computation
- Control unit: sequencing and enables

- Microoperation: elementary register operation

- Worked example Simultaneous versus sequential example
  - Initially R1=3,R2=7
  - R1←R2,R2←R1 simultaneously → 7,3
  - Same transfers sequentially → 7,7

Address, data and control buses

- Bus signal roles
  - Address: select location
  - Data: carry values
  - Control: read/write and timing
  - Requests and acknowledgements
  - CPU-master addresses: CPU→memory
  - Data: either direction
  - DMA controller: alternate bus master
  - Control: CPU outputs/device inputs
  - Bidirectional: data and control ambiguity
- Addressability versus widths
  - a address bits → 2^a locations
  - Byte capacity=locations×bytes/location
  - Word/data-bus/instruction/address widths may differ
- Worked example Address-space example
  - 32-bit byte addressing: 2^32 bytes
  - 4 GiB; bank wording 4 GB
  - 16-bit data bus ≠ 16-bit addresses

Bit masks and shifts

- Mask operations
  - Set: A OR mask
  - Clear: A AND NOT(mask)
  - Toggle: A XOR mask
  - Retain: A AND mask
  - A XOR A: entire register cleared

- Arbitrary XOR mask ≠ clearing

• Shifts, rotates and shift registers
  - Logical shifts: zero fill
  - Arithmetic right: sign replication
  - Left shift: zero fill, overflow possible
  - Left shift may change sign
  - Rotate: wrap shifted-out bit
  - SISO: serial delay
  - SIPO: serial→parallel word
  - PISO: parallel→serial word
• Worked example Masking and shift examples
  - A=1010; mask=1100
  - Set→1110; clear→0010; toggle→0110
  - Arithmetic-right 1100→1110: −4→−2
  - Logical-right 1100→0110

CPU registers and control words | Exam focus

• CPU register roles
  - PC: next instruction to fetch
  - IR: fetched instruction
  - MAR: current memory-access address
  - MDR: transferred data
  - Accumulator: operands/results
  - SP: stack tracking
  - Status flags: conditions
  - Instruction address: PC/MAR phase ambiguity
  - Registers: explicit execution operands
  - Cache: copied memory blocks
  - Both: temporary storage
• Control-word construction
  - Select ALU sources, destination, operation
  - Machine encoding table required
  - Module control word: 14 bits

- SELA/SELB/SELD: 3 bits each
- Operation field: 5 bits

• Worked example Encoded subtraction example
- R1←R2−R3
- SELA=010; SELB=011; SELD=001
- Subtraction=00101
- 010 011 001 00101
- Source/destination order matters

Instruction cycle and next PC | Exam focus

• Conceptual instruction cycle
- Fetch; decode opcode/addressing fields
- Obtain operands; execute
- Write required results
- Check permitted interrupts
- Example stages: IF, ID, EX, MEM, WB
- Five stages: not architecturally mandatory

• Fetch and next-PC distinctions
- MAR←PC; MDR←M[MAR]; IR←MDR
- PC←PC+instruction_length
- Later branch replaces sequential PC
- Read PC versus calculate branch target
- Control unit: fetch/data-flow coordination
- ALU: arithmetic and logic
- Write-back ≠ another fetch

Instruction fields and operand counts | Exam focus

• Instruction field interpretation
- Opcode: operation
- Operand fields: data/register locations
- Mode field: interpretation
- 4-bit opcode: ≤16 encodings
- Encodings ≠ implemented instructions

- 6-bit reference: 64 encodings
- Machine determines referenced objects

• Explicit address-count formats
- Three-address ADD X,Y,Z: X←Y+Z
- Destination-first convention
- Two-address ADD X,Y: X←X+Y
- Accumulator ADD X: AC←AC+M[X]
- Zero-address ADD: implicit stack operands
- Count explicit operand fields
- INC X ≠ necessarily accumulator instruction

Stacks and operand order

• Stack organization and conventions
- LIFO; PUSH adds; POP removes
- Register stack versus RAM stack
- SP: top or next free
- Address growth: upward or downward
- SP update: before/after access
- Overflow: push full stack
- Underflow: pop empty stack

• Stack arithmetic order
- Second operand popped first
- First operand popped second
- Subtraction: operand order matters
- Stack PUSH/POP: explicit addresses possible

• Worked example Stack expression example
- (A+B)×(C+D)
- PUSH A; PUSH B; ADD
- PUSH C; PUSH D; ADD; MUL
- Downward full stack PUSH: decrement, store
- POP: read, increment

Effective addresses and addressing modes | Exam focus

- Operand addressing rules
  - A: field; R: register; M[x]: contents
  - Immediate: value A
  - Direct: EA=A; operand=M[A]
  - Indirect: EA=M[A]; operand=M[EA]
  - Register: operand R
  - Register-indirect: EA=R
  - Indexed/base displacement: EA=R+A
  - PC-relative: defined PC+signed displacement
  - Auto-increment/decrement: operand-size register update
- Transfer versus addressing mode
  - Register↔memory transfer: nonunique mode
  - Direct, indexed or indirect possible
- Worked example Addressing example
  - A=100; R=20; M[100]=300; M[300]=9
  - Direct operand: 300
  - Indirect operand: 9
  - Indexed EA: 120
  - Immediate 100 ≠ M[100]

Instruction classes and interrupts

- Instruction functions
  - Data transfer: value copying
  - Arithmetic/logic: computed values
  - TEST/CMP: condition information
  - Arithmetic result often discarded
  - Branch: conditional/unconditional target
  - CALL: save return point
  - RETURN: restore return point
  - SKIP: conditional next-instruction bypass
  - Jump-to-next wording ≠ clear skip
- Interrupt and exception origins

- External: device/timer
- Internal synchronous: executing instruction
- Divide errors; illegal opcodes
- Access violations
- Software interrupt: deliberate instruction
- Overflow traps: architecture/instruction dependent
- Overflow flag ≠ automatic interrupt

RISC, CISC and control | Exam focus

- RISC/CISC design tendencies
  - RISC: simple operations, regular encodings
  - Load/store access; plentiful registers
  - Pipeline-friendly control
  - CISC: complex operations, richer addressing
  - Variable instruction lengths
  - Hardwired: direct control signals
  - Microprogrammed: stored control-word sequences
  - Tendencies ≠ universal guarantees
- RISC/CISC interpretation traps
  - One completion/cycle: throughput, not latency
  - RISC instructions: multiple cycles possible
  - CISC: simpler internal operations possible
  - Labels ≠ power/cost/time guarantees
  - Semantic gap: high-level→machine distance
  - Richer instructions: reduced semantic gap
  - Gap reduction: not universal shared objective

x86 register names and classes

- x86 register map
  - 16-bit: AX/BX/CX/DX, BP/SP, SI/DI
  - Conventional special roles
  - AL: AX low byte
  - AH: AX high byte

- BX/CX/DX: analogous byte names
- EAX: 32-bit; RAX: 64-bit
- FLAGS/EFLAGS/RFLAGS: status
- IP/EIP/RIP: instruction position
- CS: code segment; SS: stack segment
- DX: programmer-visible
- IR/MDR: conceptual internal registers

• x86 classification caveats
- SP: general-purpose despite stack role
- EAX/EIP: 80386, before Pentium
- R8-R15: 64-bit extension
- R8-R15 ≠ original Pentium
- Narrow multipurpose category requires definition

x86 legality and implicit operands

• Legacy x86 operand legality
- Intel destination-first syntax
- MOV AL,BP: invalid 8/16-bit mismatch
- INC CX; DEC CX: valid words
- Segment MOV: register/memory forms
- Immediate→segment: no direct form
- Immediate→AX; MOV DS,AX

• Implicit MUL and instruction classes
- Byte MUL: AL×operand→AX
- Word MUL: AX×operand→DX:AX
- AL=8,BH=10; MUL BH → AX=80
- CL irrelevant
- MOVS/LODS: string transfers
- XCHG: operand exchange
- INC: arithmetic
- Different: transfer-versus-arithmetic criterion

Memory hierarchy and timing | Exam focus

- Hierarchy and timing measures
  - Speed/cost order: registers→L1/L2/L3→DRAM
  - Then SSD→HDD→archival tape
  - Faster: smaller, costlier per bit
  - Access time: read/write latency
  - Cycle time: minimum operation interval
  - Recovery may extend cycle time
  - Transfer rate: data/time, not delay
- Locality and cache metrics
  - Temporal: reuse recent data
  - Spatial: access nearby locations
  - Cache: nearby memory-block copies
  - Hit ratio=hits/accesses
  - Miss ratio=1−hit ratio
  - Capacity versus latency versus bandwidth
  - Hit ratio: separate comparison dimension
- Worked example Hit ratio and AMAT example
  - 90/100 hits → 0.9
  - Hit time: 2 ns
  - Additional miss penalty: 50 ns
  - AMAT=2+0.1×50=7 ns

RAM, firmware and chip capacity

- Memory technologies
  - SRAM: powered bistable storage
  - SRAM: fast, costly, no refresh
  - DRAM: charge storage, refresh required
  - DRAM: dense main memory
  - SRAM/DRAM: ordinarily volatile
  - ROM/flash: retain unpowered contents
  - Erasable/programmable ROM variants
  - ROM never modifiable: overbroad

- Firmware: low-level system code
- Commonly nonvolatile storage

• Memory-chip organization
- 2^a×w: words×width
- a address inputs; w data bits
- Parallel chips: increased word width
- Banks: increased word count
- Address decoding versus data selection

• Worked example Chip-combination example
- Eight 1K×1 parallel → 1K×8
- 10 shared address lines
- 8 data bits
- Four banks → 4K×8
- Overall address bits: 12

Cache placement and index bits | Exam focus

• Cache mapping equations
- B: block bytes; L: lines
- Block=floor(address/B)
- Offset=address mod B
- Direct: line=block mod L
- Fully associative: any line
- W-way: S=L/W sets
- Set=block mod S
- Any W ways within set

• Cache address decomposition
- Power-of-two sizes assumed
- Offset bits=log2 B
- Index bits=log2 S
- Remaining address bits: tag
- Hit: matching tag and valid bit
- Same direct index: mutual conflict
- Associativity: fewer placement conflicts

- More tag comparisons

- Worked example Direct-mapped conflict example
  - B=16 bytes; L=4 lines
  - Addresses 0,64 → blocks 0,4
  - Both index 0
  - Alternating accesses: conflict misses

Cache replacement and write policies | Exam focus

- Replacement trace rules
  - FIFO: earliest inserted victim
  - LRU: least recently accessed victim
  - Track hit/miss, set, policy state
  - Hits update LRU order
  - Hits preserve FIFO insertion order
  - Direct mapping: no victim choice
- Write policies and cache classifications
  - Write-through: lower-level update each write
  - Write-back: cache update, dirty bit
  - Dirty eviction: lower-level write
  - Write-allocate: fetch missed block
  - No-write-allocate: write below, no installation
  - L1/L2: hierarchy levels
  - Split: separate instruction/data caches
  - Unified: combined instruction/data cache
  - Internal/external: separate classification
  - Location ≠ split/unified
- Worked example FIFO versus LRU trace
  - Two fully associative lines
  - A,B,A,C: second A hits
  - Remaining accesses miss
  - LRU evicts B
  - FIFO evicts A

Virtual addresses and page faults

- Virtual-memory mechanisms
  - Translated process address spaces
  - Protection
  - Nonresident pages: backing storage
  - MMU: virtual/logical→physical translation
  - Page tables
  - TLB: recent translations
  - Virtual memory ≠ merely disk RAM
- Paging equations and fault distinctions
  - Page=floor(virtual_address/P)
  - Offset=virtual_address mod P
  - Physical_address=frame×P+offset
  - Valid nonresident access: page fault
  - Invalid/forbidden accesses can fault
  - Cache miss ≠ TLB miss
  - TLB miss ≠ page fault
  - Page fault ≠ mandatory disk read
  - Zero-fill; resident shared pages
- Worked example Address-translation example
  - P=1024; virtual_address=2053
  - Page=2; offset=5
  - Page 2→frame 7
  - Physical_address=7×1024+5=7173

Persistent storage and access methods

- Storage media mechanisms
  - HDD: magnetic tracks and sectors
  - Addressed blocks; rotating platters
  - Reading: head over track
  - Seeking: radial head movement
  - HDD bits survive power removal
  - SSD: nonvolatile semiconductors, no mechanical seek

- Optical media: lasers
- Tape: sequential streaming and archives
- Ethernet cable: communication, not disk/RAM

• Access-method taxonomy
- Tape: sequential access
- Disk: direct, positioning delay
- RAM: random, broadly address-independent timing
- Associative: content matching
- Informal disk random-access usage
- Direct/random options require taxonomy
- Optical 100-300 ms: historical example
- Device timings: not universal

I/O interfaces and commands | Exam focus

• Peripheral and interface roles
- Peripheral: input/output/storage beyond execution core
- CPU ≠ peripheral
- Input units: external information encoding
- Output units: processed-result presentation
- Controller: device protocol and timing
- Interface: data/status/control registers
- Buffers: speed mismatch absorption
- Error/status bits: condition reporting

• I/O command categories
- Control: start actions
- Test/status: readiness/errors
- Read: device→system
- Write/output: system→device
- Issue: sending, not command category
- Controller: hardware
- Driver: controlling software

Polling, interrupts and DMA | Exam focus

- I/O transfer mechanisms
  - Polling: repeated status checks
  - CPU moves each item
  - Interrupt-driven: other work until event
  - Handler still consumes CPU time
  - DMA: controller device↔memory block transfer
  - CPU setup first
  - Completion/errors may interrupt CPU
- DMA responsibilities and modes
  - CPU: configuration and ownership
  - Address mapping and completion
  - Burst: bus held for block
  - Cycle stealing: individual transfer opportunities
  - Transparent: idle opportunities
  - Reduced per-word CPU work
  - Competes for memory bandwidth
  - Substantial blocks favor DMA
  - Device address mapping: separate dimension
- Worked example Thousand-word transfer comparison
  - Polling/interrupts: CPU work per word
  - DMA: setup plus completion
  - Actual memory transfers still take time

I/O spaces, priorities and handshaking | Exam focus

- Memory-mapped versus isolated I/O
  - MMIO: device registers in memory space
  - Suitable load/store operations
  - Isolated: separate port space
  - Special IN/OUT instructions
  - Device access widths matter
  - Ordering and side effects
  - Status read ≠ harmless RAM read

- Handshaking and interrupt priority
  - Asynchronous request/ready plus acknowledgement
  - Sender: data then request
  - Receiver: accept then acknowledge
  - Withdraw signals after acceptance
  - Unequal-speed tolerance
  - Single strobe: no feedback
  - Priority: simultaneous interrupt selection
  - Polling, daisy chains, priority hardware
  - Serial: bits over time
  - Parallel: multiple bits/lines
  - Parallel ≠ guaranteed faster

Interfaces by connection role | Exam focus

- Named interface roles
  - USB: serial peripherals, keyboards/storage
  - RS-232: point-to-point serial electrical/interface standard
  - Serial alone: USB/RS-232 ambiguity
  - HDMI: digital video/audio
  - Ethernet: networking, not memory
  - SATA: storage interface
  - Legacy IDE/PATA and SCSI
  - iSCSI: SCSI commands over IP
- Interconnect versus software standards
  - PCI: peripheral expansion interconnect
  - PCI ≠ disk protocol
  - Unlike buses: bridge/controller logic
  - Processor-bus extension needs topology
  - PCI/Multibus: ambiguous without topology
  - POSIX: software interface standard
  - POSIX open/read/write/close
  - ASCII: character encoding
  - Standard disk interface requires era/requirements

Analog interfaces and component identity

- Conversion direction and named components
  - ADC: analog input→digital code
  - DAC: digital code→analog output
  - ADC080x: 8-bit successive-approximation ADC
  - DAC0830: DAC
  - 82C55: programmable parallel I/O
  - 82C55 ≠ ADC
  - Component identity versus conversion direction
- Converter resolution
  - Ideal n-bit voltage span V
  - Approximate code step=V/2^n
  - Endpoint conventions/specifications required
  - Quantization: analog ranges→discrete codes
  - Sample rate ≠ resolution
- Worked example ADC step-size example
  - 8 bits; 5.12 V span
  - 256 codes
  - Step=5.12/256=0.02 V
  - Same span, more bits: smaller steps

Execution time and bottlenecks | Exam focus

- CPU performance equations
  - CPU_time=instruction_count×CPI×clock_period
  - CPU_time=instruction_count×CPI/clock_rate
  - CPI: instruction-class weighted average
  - IPC=1/CPI: consistent whole-program definitions
  - Higher frequency ≠ faster program
  - Increased instruction count/stalls
  - Throughput: completed work/time
  - Latency: individual job delay
- Amdahl’s law
  - f: accelerated execution fraction

- s: local speedup factor
- Speedup=1/((1−f)+f/s)
- Unaffected fraction limits gains
- Same workload comparison
- Include memory and branch penalties
- Clock speed ≠ performance

• Worked example Performance calculation examples
- 10^6 instructions; CPI=2; 1 GHz
- CPU time=2 ms
- f=0.5; s=2
- Speedup=1/(0.5+0.25)=4/3
- Overall speedup ≠ 2

Pipeline overlap and timing | Exam focus

• Ideal pipeline timing
- Different instructions overlap stages
- Primary gain: throughput
- Balanced k-stage single-issue pipeline
- n instructions: k+n−1 cycles
- No stalls assumed
- Nonoverlapped work: nk stage cycles
- Steady-state: one instruction/cycle
- Individual latency: multiple stages

• Pipeline clock and overhead
- Clock: slowest stage plus register overhead
- Short streams: fill/drain cost
- Fetch/decode overlaps another execution
- Multiple issue: several instructions/cycle
- Multiple-issue resources required

• Worked example Five-stage timing example
- k=5; n=10
- Ideal cycles=14
- Nonoverlapped cycles=50

- Speedup=50/14≈3.57
- Short stream: below limiting fivefold gain

Hazards and instruction-level parallelism | Exam focus

- Pipeline hazard classes
  - Structural: shared-resource competition
  - Data: unavailable required result
  - Forwarding: early result supply
  - Some dependencies still stall
  - Control: unresolved branch next PC
  - Prediction: guess direction/target
  - Speculative continuation
  - Misprediction: discard work
- Instruction-level parallelism mechanisms
  - Superscalar: multiple instructions/units per cycle
  - Dependencies constrain issue
  - Out-of-order: ready before stalled older operations
  - Preserve architectural results and exceptions
  - Dynamic allocation: runtime entries/units
  - Wider fetch: improved instruction supply
  - Wider fetch ≠ lower memory latency

Multiprocessing, vectors and Flynn classes | Exam focus

- Flynn stream classification
  - SISD: one instruction, one data
  - SIMD: one instruction, multiple data
  - MIMD: multiple instruction/data streams
  - MISD: multiple instructions, common data
  - MISD uncommon
  - Vector instruction: sequence of elements
  - Physical lanes may batch elements
  - Vector ≠ simultaneous entire sequence
- Multiprocessor organization and scaling

- SMP: shared memory/I/O
- Peer processors under OS
- Distributed memory: separate address spaces
- Interprocessor communication
- More cores: available parallel work
- Serial dependencies and coordination limits
- Core i7: product family
- Core count requires model

Historical generations and processor widths

- Generation conventions and caveats
  - First: vacuum tubes
  - Second: transistors
  - Third: integrated circuits
  - Fourth: microprocessor/VLSI-era PCs
  - Stored programs: first era
  - High-level languages: transition into second
  - Dates cross generation boundaries
  - Magnetic-core memory spans eras
  - Witnessed-use clue: nonunique generation
  - Multiple generations improved performance
- Integration and x86 width history
  - Smaller individual components
  - Increased capacity, density, performance
  - Trends ≠ every-model guarantees
  - 80386: 32-bit x86, 1985
  - Before 80486/Pentium
  - 80286: 16-bit predecessor
  - n-bit CPU ≠ uniform register widths
  - PC/IR/address widths may differ

Underspecified and malformed questions

- Question-scope ambiguities

- Temporary storage: RAM/registers/cache
- Explicit role determines interpretation
- Memory includes ROM/flash
- Power-off disk/memory choice: RAM assumption
- Architecture lifetime requires assumptions
- Power efficiency requires assumptions
- Generational gains: not exclusive definitions

• Malformed bank-item handling
- TEST options displaced into stem
- Reconstructed intent ≠ unique answer
- Gate question: actual symbol required
- Absent drawing: definitions insufficient
- Opcode role explicit: figure unnecessary
- Identify minimal missing phrase
- Repair options or diagram
- Weak key ≠ authority

Data Communication and Computer Networking

Revision Outlines

Communication components and quality | Exam focus

- Communication components
  - Data: raw representations
  - Information: interpreted data
  - Message; sender; receiver; medium; protocol
  - Source creates data
  - Transmitter: data → signal
  - Receiver reconstructs data
  - Distance: engineering factor, not sixth component
- Communication quality
  - Delivery: intended recipient
  - Accuracy: preserved content
  - Timeliness: deadline
  - Jitter: arrival-delay variation
  - Average delay can conceal disruptive jitter
  - Throughput: achieved transfer rate
  - Reliability: dependable operation
  - Security: unauthorized actions
  - Scalability: growth without unacceptable degradation

Network size, access and architecture

- Network geographic scope
  - PAN: personal Bluetooth devices
  - LAN: building or campus
  - MAN: metropolitan region
  - WAN: distant regions
  - Device count alone insufficient

- Internet: global network of networks
- Internet as largest WAN example

• Access scope and service roles
- Intranet: organizational internal services
- Extranet: approved outsiders
- Access scope ≠ geographic size
- Client requests; server supplies
- Peers perform both roles
- Dedicated servers can also be clients
- DNS; backup; update dependencies
- Ten-user cutoff: guideline only
- Centralization: simpler administration, concentrated dependencies
- Peer networks still require security

Protocols, standards and layers | Exam focus

• Protocols and standards
- Syntax: message structure
- Semantics: field meanings
- Timing: exchange schedule and rate
- Standards enable interoperability
- De facto: established use
- De jure: formal approval
- ISO: international standards
- IEEE: 802.3 Ethernet; 802.11 Wi-Fi
- ITU-T: telecommunications
- ANSI: US standards coordination
- EIA: interface standards
- FCC: regulator

• Layer models
- Reference model: responsibilities and interfaces
- Model ≠ executable software
- OSI: seven reference layers
- TCP/IP: implemented suite, usually four layers

- Five-layer model separates physical and link
- State model before counting

Signals, periods and bandwidth | Exam focus

- Signal characteristics
  - Analog: continuous; digital: discrete symbols
  - Data type ≠ signal type
  - Digital bits modulate analog carriers
  - Sine wave: amplitude, frequency, phase
  - f: cycles/second
  - T=1/f
  - λ=v/f; propagation speed v
  - Phase: within-cycle displacement
  - Composite signals combine frequencies
- Bandwidth and carrier placement
  - Frequency bandwidth=highest−lowest; hertz
  - Low-pass includes zero frequency
  - Band-pass: nonzero frequency range
  - Baseband: no carrier shift
  - Passband: modulated carrier
  - Hz ≠ bit/s
  - 2 kHz → T=0.5 ms

Bit rate, baud and channel capacity | Exam focus

- Bit rate versus baud
  - R: bits/second; bit interval=1/R
  - Baud: symbols/second, not voltage transitions
  - M states → log2(M) bits/symbol
  - R=baud×log2(M), before coding overhead
  - Manchester: extra transitions
  - Binary ≠ universally baud=bit rate
- Channel capacity bounds
  - Nyquist: ideal noiseless channel

- R≤2B log2(M)
- Shannon: Gaussian noise
- C=B log2(1+S/N)
- S/N: linear power ratio
- SNR(dB)=10 log10(S/N)
- Bandwidth; signal levels; noise
- Distance affects attenuation, delay, achievable rate
- Distance absent from simplified formulas

• Worked example Nyquist and Shannon example
- B=3000 Hz; M=4
- 2×3000×log2(4)=12,000 bit/s
- S/N=15 → log2(16)=4
- Shannon limit=12,000 bit/s
- 15 dB ≠ linear ratio 15

Delay, impairments and transmission modes | Exam focus

• Delay components
- Transmission delay=packet bits/link bit rate
- Propagation delay=distance/signal speed
- Processing delay; queueing delay
- 1000 bytes at 10 Mbit/s
- Serialization=0.8 ms
- Higher rate leaves propagation unchanged

• Signal impairments
- Attenuation: reduced strength
- Distortion: changed shape, unequal frequency delays
- Noise: unwanted energy
- Thermal noise: random thermal motion
- Crosstalk: nearby-channel leakage
- Impulse noise: short disturbance
- Amplifiers amplify noise too
- Digital repeaters regenerate decisions
- Corrupted bits not always recoverable

- Transmission modes
  - Simplex: one direction
  - Half duplex: alternating directions, walkie-talkie
  - Full duplex: simultaneous directions, telephone
  - Serial: sequential bits
  - Parallel: multiple conductors, skew risk
  - Asynchronous serial: character start/stop bits
  - Synchronous framing: ongoing clock recovery
  - Serial timing ≠ browser asynchronous requests

Cables, wireless and connectors | Exam focus

- Guided media
  - Twisted pairs reduce coupled interference
  - UTP: unshielded; STP: shielded
  - Coax: axial conductor and outer shield
  - Fiber: guided light; EMI resistance
  - Fiber: high capacity and long reach
  - Suitable optics and installation required
  - Specification determines speed, cost, reach
  - Fiber remains tappable
- Unguided media and Ethernet cables
  - Radio; microwave; infrared
  - Ground waves follow Earth
  - Sky waves: ionized-layer reflection
  - Line-of-sight requires visibility
  - Microwaves belong to radio spectrum
  - Infrared generally blocked by walls
  - Antenna determines directionality
  - Omnidirectional: broad azimuth, not uniform 3-D
  - Straight-through: unlike MDI/MDI-X roles
  - Crossover: like roles, PC-PC, switch-switch
  - Auto-MDI-X removes legacy cable constraint

Topology and failure reasoning | Exam focus

- Topology and link scope
  - Physical: link layout
  - Logical: communication/access relationships
  - Point-to-point: two endpoints
  - Multipoint: shared medium
  - Full mesh uses point-to-point links
  - Contradictory module mesh statement
- Topology failure reasoning
  - Bus: shared backbone; unterminated-break disruption
  - Star: central device
  - Star edge failure: one endpoint
  - Star central failure: whole star
  - Ring: loop; single-break vulnerability
  - Protected dual rings differ
  - Token Ring uses token passing
  - Token access not inherent to rings
  - Tree: hierarchy; hybrid: combined arrangements
- Full undirected mesh calculations
  - Links=n(n−1)/2
  - Interfaces/node=n−1
  - 50 nodes → 1225 links
  - Redundancy improves fault tolerance
  - Higher cable and interface cost

NICs, repeaters and hubs | Exam focus

- NIC and physical functions
  - NIC: computer network interface
  - NIC commonly implements physical and link
  - Physical interface ≠ exclusively layer 1
  - Connectors; signaling; voltage; timing
  - Physical coding and modulation
  - PHY-MAC interface: hardware signaling

- Repeaters, hubs and device roles
  - Repeater: layer-1 signal regeneration
  - No destination-address inspection
  - Active hub: multiport repeater
  - Signals repeated onto other ports
  - Half-duplex hub: shared collision domain
  - Passive hub: connector
  - No MAC learning or collision elimination
  - Network printer: endpoint consuming services
  - Routers; hubs; access points: intermediaries
- Legacy IEEE 802.3 PHY architecture
  - AUI: attachment unit interface
  - MAU: medium attachment unit
  - MAC interfaces to PLS
  - PLS: Physical Signaling sublayer
  - Physical-layer signaling: PLS
  - Architecture-specific, not universal OSI terminology
  - Other PHY designs: RS; PCS; PMA
  - Reconciliation; coding; medium attachment

Bridges, switches and access points | Exam focus

- Layer-2 bridges and switches
  - Learn source-MAC-to-port mappings
  - Look up destination MAC
  - Known unicast: destination port
  - Same incoming segment: filter
  - Unknown unicast or broadcast: VLAN flood
  - Switch need not select one port
  - Wireless AP: wireless-wired link bridge
- Switch collision and broadcast boundaries
  - Separate collision domain per port
  - Full-duplex point-to-point Ethernet: no collisions
  - Full duplex: no CSMA/CD

- Independent simultaneous conversations
- Broadcasts remain within VLAN
- Additional controls may filter broadcasts
- Layer-3 switch also routes
- Function overrides marketing name

Routers, gateways and traffic controls | Exam focus

- Routers and gateways
  - IP forwarding: destination prefixes, next hops
  - Routers separate broadcast domains
  - Local link broadcasts normally stopped
  - Default gateway: usually router
  - Protocol gateway: application-format translation
  - Gateway ≠ inherently every OSI layer
- Firewall, VPN and modem
  - Firewall: policy-based allow/reject/inspect
  - Firewall need not encrypt
  - VPN: Virtual Private Network
  - Logical private connection over another network
  - Secure VPN: tunnel authentication and encryption
  - Modem: outgoing modulation, incoming demodulation
  - Home router may combine distinct functions

OSI layers and encapsulation | Exam focus

- Encapsulation and routed hops
  - Application data → segment/datagram → packet
  - Packet → frame → bits/symbols
  - Receiving reverses encapsulation
  - Each layer serves layer above
  - Router removes incoming link frame
  - Router inspects IP; creates next-hop frame
  - MAC addresses usually change per hop
  - End-to-end IP ordinarily persists until translation

- 7 Application
  - Network services for applications
  - Message/data
- 6 Presentation
  - Representation; compression; encryption
  - Data
- 5 Session
  - Dialog setup; checkpoints; recovery
  - Data
- 4 Transport
  - Process delivery; segmentation
  - Protocol-dependent reliability
  - TCP segment; UDP datagram; port
- 3 Network
  - Logical addressing; routing
  - Packet; IP
- 2 Data link
  - Hop delivery; framing; MAC access
  - Error detection
  - Frame; MAC
- 1 Physical
  - Signaling; bit/symbol transport
  - Bits/symbols

Link control, framing and error detection | Exam focus

- Link control and framing
  - LLC: upper-protocol identification/multiplexing
  - LLC may provide link control
  - MAC: medium-dependent addressing/access, frame delivery
  - Network addressing: layer 3
  - Ethernet; PPP; HDLC: link protocols
  - Headers: addresses, control/type fields
  - Synchronization and framing depend on protocol

- Error detection versus recovery
  - Redundancy detects, not necessarily corrects
  - Even parity: even total ones
  - 1011 → parity bit 1
  - Odd flips detected; even counts missed
  - CRC: polynomial bits, XOR-division remainder
  - Receiver repeats CRC division
  - Internet checksum: end-around carry, complemented sum
  - Burst: first-last corruption, unchanged interiors included
  - Recovery: retransmission or sufficient FEC redundancy

Access control and reliable-link procedures | Exam focus

- Random access
  - ALOHA: opportunistic sending
  - CSMA: carrier sensing
  - Legacy half-duplex Ethernet: CSMA/CD
  - Collision→jam→stop→random backoff
  - Jam indicates collision, not clearance
  - Wi-Fi 802.11: CSMA/CA
  - Waiting/backoff and acknowledgments
  - Hidden terminals remain despite sensing
- Controlled access and reliable links
  - Polling; reservation; token turns
  - Correct coordination avoids contention collisions
  - Channelization: frequency, time, code
  - Stop-and-wait: one frame, await ACK
  - Timeout retransmission; sequence-bit duplicate detection
  - Sliding window: multiple unacknowledged frames
  - Link reliability/flow control: hop-local
  - Transport reliability/flow control: end-to-end
  - Ethernet detects corruption without delivery guarantee

Line coding, modulation and sampling

- Line coding
  - Digital bits → digital signal patterns
  - NRZ: no forced mid-bit return
  - Manchester: clocking mid-bit transition
  - Manchester bit convention must be specified
  - Differential Manchester: relative-state transitions
  - AMI: alternating successive-one polarities
  - Pseudoternary: alternating zero pulses
  - Block coding adds redundancy
  - Scrambling replaces problematic patterns
  - Scrambling need not increase bit count
  - Multilevel: 2B1Q; 8B6T
  - MLT-3: voltage-transition sequence
- Modulation and digitization
  - ASK: amplitude; FSK: frequency; PSK: phase
  - QAM: amplitude plus phase
  - AM/FM/PM carry analog data
  - PCM: sampling→quantization→encoding
  - Practical sampling: fs>2fmax
  - b bits → 2^b levels
  - Uncompressed rate=fs×b
  - PAM: pulse amplitude; PWM: pulse width
  - PAM/PWM alone ≠ binary PCM code

Network delivery and IPv4 header | Exam focus

- IPv4 service and header
  - IP: connectionless best-effort delivery
  - No end-to-end ordering or retransmission
  - IPv4 address: 32 bits
  - Header: 20-60 bytes, not 32 bits
  - IHL: 32-bit words
  - Total length: header plus payload

- TTL limits router lifetime
- Protocol field: TCP, UDP, upper protocol

• IPv4 checksum and fragmentation
- Header checksum exists
- Module no-error-checking claim incorrect
- Identification; flags; fragment offset
- Sources and routers fragment when allowed
- Destination reassembles
- DF plus insufficient MTU: forwarding fails
- Possible ICMP diagnostic

Transport choice, ports and connections | Exam focus

• TCP versus UDP
- TCP: reliable ordered byte stream
- Sequence numbers; ACKs; retransmission; windows
- Flow control protects receiver
- Congestion control protects network
- UDP: independent datagrams, ports, checksum
- No built-in setup, reliability, retransmission, ordering
- Interactive traffic: late versus lost data
- Application reliability possible over UDP

• Transport endpoints
- Ports identify transport endpoints
- Application chooses transport
- Socket buffer requests subject to OS limits
- TCP connection: source/destination IP and ports
- TCP protocol understood in four-tuple
- IPv4 protocol number ≠ port
- 80: usual HTTP port
- Clients often use ephemeral ports
- SCTP: reliable messages, multiple streams

Trace the TCP handshake | Exam focus

- TCP three-way handshake
  - SYN → SYN+ACK → ACK
  - SYN consumes one sequence position
  - ACK: next expected sequence
  - Client ISN=395; server ISN=260
  - Client SYN: seq395
  - Server SYN+ACK: seq260 ack396
  - Client ACK: seq396 ack261
  - Application byte counts advance sequence numbers
  - FIN consumes one sequence position
- TCP connection interpretation
  - Logical transport state, no reserved circuit
  - Ports multiplex separate applications
  - Separate applications ≠ one TCP stream
  - Layer-4 reliability depends on protocol

Upper layers and the TCP/IP correspondence | Exam focus

- OSI upper-layer roles
  - Session: dialog establishment and management
  - Checkpoints enable interrupted-transfer recovery
  - Presentation: encoding translation, compression, encryption
  - Compression reduces representation size
  - Encryption hides content, need not shrink
  - Application protocols serve applications directly
  - Practical functions distributed differently
  - TLS ≠ separate physical presentation device
- TCP/IP correspondence
  - OSI 5-7 → application
  - Transport retained
  - OSI network → Internet
  - Link plus physical → network access
  - Model does not guarantee reliability

- Application service builds on process transport
- Vague service labels may overlap

DNS, DHCP and address resolution | Exam focus

- DNS and DHCP
  - DNS: names → records
  - A: IPv4; AAAA: IPv6
  - DNS: UDP or TCP port 53
  - DHCP: dynamic IPv4 leases
  - Address; mask; gateway; DNS server
  - Discover → offer → request → acknowledge
  - IP itself does not assign addresses
- Address resolution and companion protocols
  - ARP: on-link IPv4 next-hop → MAC
  - Broadcast request; normally unicast reply
  - Off-subnet ARP resolves gateway MAC
  - Remote target MAC not resolved
  - Network/link boundary; often link-layer classification
  - Network-protocol label: loose textbook convention
  - RARP: hardware → IP
  - BOOTP/DHCP: boot configuration
  - ICMP: errors and diagnostics
  - IGMP: IPv4 multicast membership
  - Companions ≠ constituent pieces of IP

Web, file, mail and remote-login protocols | Exam focus

- Application service protocols
  - HTTP: stateless request/response semantics
  - HTTP message body optional
  - HTTPS: HTTP with TLS
  - FTP: separate control/data connections
  - FTP ≠ SFTP
  - SMTP: mail submission and relay

- POP3: retrieval
- IMAP: server-mailbox access/synchronization
- SMTP ≠ end-user mailbox retrieval

• Usual ports and remote access
- FTP control 21; SSH 22
- Telnet 23; SMTP relay 25
- DNS 53; HTTP 80
- POP3 110; IMAP 143; HTTPS 443
- Telnet: unencrypted terminal
- SSH: protected remote access
- Configured ports may differ
- HTTP exchange ≠ physical duplex mode

IPv4 notation and classful defaults

• IPv4 representation
- Four decimal octets, each 0-255
- 32 binary bits
- Octet weights: 128,64,32,16,8,4,2,1
- 11000001 10000011 00011011 11111111 → 193.131.27.255
- 257 invalid
- Syntax validity ≠ host assignability
- .0/.255 host validity depends on prefix

• Historical classful defaults
- A: leading 0, /8
- B: leading 10, /16
- C: leading 110, /24
- D: 1110 multicast; E: 1111 reserved
- Ordinary A: 1-126; 0/127 special
- B: 128-191; C: 192-223
- Usable: 2^24−2; 2^16−2; 254
- Modern CIDR actual prefix overrides classes

Special ranges, private addresses and IPv6

- Private and special IPv4 ranges
  - RFC1918: 10.0.0.0/8
  - 172.16.0.0/12 through 172.31.255.255
  - 192.168.0.0/16
  - Reusable internally; no global public routing
  - LAN public addressing permitted with allocation
  - 127.0.0.0/8 loopback; commonly 127.0.0.1
  - Loopback works without external link
  - 255.255.255.255: limited broadcast
  - 0.0.0.0: possible unconfigured source
  - Link-local/APIPA: 169.254.0.0/16
  - Automatic selection excludes edge /24 blocks
- IPv6 contrasts
  - 128-bit hexadecimal addresses; flexible prefixes
  - No IPv4-style broadcast
  - Fragmentation: sources only, never routers
  - No automatic traffic encryption
  - Authentication remains necessary
  - Address space, bit length, encryption-support options
  - Specific comparison needed for unique answer

Masks, prefixes and binary AND | Exam focus

- Prefix masks
  - /p: p ones, then 32−p zeros
  - Partial octets: 128,192,224,240,248,252,254
  - /19=255.255.224.0
  - /20=255.255.240.0
  - /21=255.255.248.0
  - 255.248.0.0 ≠ /21
- Network by binary AND
  - Network=address AND mask
  - Host zeros: block start

- Host ones: broadcast
- 172.16.28.252/20: third-octet increment 16
- Network=172.16.16.0; broadcast=172.16.31.255
- Written host/prefix need not be aligned
- 172.105.180.0/21 → 172.105.176.0/21

Find subnet bounds and usable hosts | Exam focus

- Subnet bounds and capacity
  - Ordinary broadcast LAN: h=32−p
  - Total=2^h; usable=2^h−2
  - Reserve network and directed broadcast
  - Changing-octet increment=256−mask octet
  - Round down to increment multiple
  - Last address=next boundary−1
- Subnet boundary examples
  - 192.168.2.64/26: .64-.127
  - Usable .65-.126: 62 hosts
  - 192.168.10.25/28 → .16-.31
  - 172.17.128.0/21 → 172.17.135.255
  - .128.255 and .135.0: valid interior hosts
  - .135.255: broadcast
  - /27: 32 total, 30 usable

Borrow bits and choose a mask | Exam focus

- Equal subnet borrowing
  - Parent /a → child /p
  - Borrowed bits b=p−a
  - Subnets=2^b, no modern subtract-two
  - Class B /16 → /19
  - b=3 → eight subnets
  - 8192 addresses; 8190 conventional hosts
  - /19 alone does not identify parent
- Choose mask from demands

- Smallest h satisfying 2^h−2≥H
- p=32−h
- 24 hosts → /27
- 52 or 57 → /26
- 100 → /25; 500 → /23
- At least six subnets → b=3
- /24 → /27: eight subnets
- Check subnet count and host capacity

Design and implement a VLSM address plan | Exam focus

• VLSM allocation method
- Different masks for LAN sizes
- Sort largest demands first
- Round reservations into power-of-two blocks
- Aligned, nonoverlapping allocations within parent
- Explicit remaining free space

• VLSM worked plan
- /24 demands: 100,60,29
- .0/25: hosts .1-.126
- .128/26: hosts .129-.190
- .192/27: hosts .193-.222
- Remaining .224/27
- Two 100-host LANs: separate /24
- .0/25 plus .128/25
- Count router interfaces unless already included

• Address-plan implementation
- Unique usable IPs and masks
- Default gateway inside local subnet
- Router interfaces and routes
- DHCP exclusions
- Test local→gateway→remote→DNS separately
- Check duplicate addresses and overlap
- Verify return routes before application blame

Point-to-point links and route aggregation

- Point-to-point prefix exceptions
  - Conventional /30: four addresses, two endpoints
  - Often smallest offered WAN choice
  - RFC3021 /31: supported point-to-point links
  - Both /31 addresses usable
  - Subtract-two rule inapplicable to /31
  - /32: single address or host route
  - /30 not universally smallest link prefix
- CIDR aggregation
  - Classless Inter-Domain Routing
  - Replaces rigid class allocation
  - Aggregate contiguous aligned common-prefix blocks
  - Adjacent-looking labels may not aggregate
  - Four consecutive /26 blocks → /24
  - Preserve more-specific routes for correct next-hops

NAT and port translation

- NAT variants
  - Static NAT: fixed address mapping
  - Dynamic NAT: mapping pool
  - PAT/NAPT: address-plus-port mapping
  - Multiple private clients share public address
  - Translation ≠ ARP or encryption
  - Return traffic requires translation state
  - Configured inbound mapping alternative
- Public Internet prerequisites
  - 10.0.0.1 commonly uses gateway NAT
  - Valid routing and public external address
  - NAT supplies neither physical connectivity nor DNS
  - Public address ≠ permission to use
  - LAN: correctly allocated public/private unicast
  - Subnet correctness remains necessary

Routing algorithms and destination sets

- Routing versus forwarding
  - Routing computes paths
  - Forwarding selects packet next hop
  - Longest matching prefix first
  - Then routing-system tie breakers
  - Default /0: no more-specific match
  - Static: configured; dynamic: reachability exchange
  - Distance vector: neighbor estimates
  - Link state: distributed links, shortest paths
  - Interior protocols: within autonomous system
  - BGP: between autonomous systems
- Destination sets and distribution
  - Flooding: eligible-link copies, duplicate/lifetime controls
  - Random walk: probabilistic neighbor
  - Unicast: one; broadcast: all within scope
  - Multicast: subscribed group
  - Anycast: one shared-address member
  - Anycast nearest by routing metric
  - Multi-destination target list ≠ necessarily broadcast
  - Multicast trees: reverse-path checks
  - Ethernet spanning tree: separate layer-2 mechanism

Circuit, message and packet switching

- Circuit versus packet switching
  - Circuit: setup → reserved transfer → release
  - Predictable allocated bandwidth
  - Setup/blocking and idle reservation costs
  - Traditional telephone call
  - Packets statistically share links
  - Queueing: variable delay and loss
  - Voice-over-packet remains possible
  - Buffering; scheduling; capacity planning

- Forwarding and switching variants
  - Datagrams forwarded independently
  - Virtual circuit establishes forwarding state
  - Virtual-circuit reservation optional
  - Message switching stores complete message
  - No dedicated end-to-end message path
  - Store-and-forward waits for full packet/frame
  - Cut-through starts after sufficient header
  - Fragment-free waits collision-sensitive initial portion
  - Routing ≠ guaranteed geometrically shortest path

Multiplexing and channelization

- Multiplexing and demultiplexing
  - Multiple streams share one link
  - Receiver separates streams
  - Multiplexing ≠ broadcasting
  - FDM: frequency bands, guard spacing
  - WDM: optical wavelengths
  - TDM: time slots
  - Synchronous TDM: recurrent slots, including idle
  - Statistical TDM: demand allocation, identification
- Channelization versus port multiplexing
  - FDMA: user frequency allocation
  - TDMA: user time allocation
  - CDMA: coded-signal separation
  - Physical multiplexing ≠ transport port multiplexing
  - Combining-signals question: multiplexing
  - Missing correct option remains missing

Identify networking layers and addressing assumptions

- Ambiguous networking stems
  - Identify exact layer, protocol, scope
  - Link and transport: flow/error control

- Locally administered MAC addresses
- Bridges and switches: both layer 2
- LAN: public or private unicast
- UDP and IP: both connectionless
- Session dialog ≠ TCP connection
- Overlaps need narrower stem/multiple selections

• Damaged addressing and option questions
- Host without prefix: capacity indeterminate
- Explicit classful assumption permits default
- Subnet count requires parent block
- Obsolete zero/all-ones subnet exclusions: explicit only
- Missing concept ≠ duplicated distractor answer
- Restore OCR-damaged original options

Data Structure and Algorithms

Revision Outlines

Data types and representations | Exam focus

- ADT versus representation
  - ADT values and operations
  - Array-backed LIFO stack
  - Linked-node LIFO stack
  - Abstraction: relevant details
  - Patient urgency and arrival order
  - Hair colour irrelevant
- Structure classifications
  - Linear: arrays, lists, stacks, queues
  - Trees: hierarchies
  - Graphs: general connections
  - Heterogeneous records
  - Integer, character, floating fields
  - Ordinary array: one element type

Arrays: access, capacity, shifting | Exam focus

- Array address arithmetic
  - Equal-size elements
  - Zero-based indexing
  - address(A[i])=base+i×element_size
  - RAM positional access: O(1)
  - Row-major rectangular matrix
  - base+(row×columns+column)×element_size
- Array capacity and growth
  - Fixed capacity chosen beforehand
  - Capacity choice: beyond compile time
  - Resizing: larger block and copy
  - Middle insertion: linear shifts

- Spare-capacity append: O(1)
- Resizing append: O(n)
- Geometric growth: amortized O(1) append

• Worked example Insertion by rightward shifting
- Spare slot required
- Insert 9 at index 1
- Shift 6 before 4
- [2,4,6]→[2,9,4,6]

Operations, records, memory ownership

• Structure operations
- Traversal: every item
- Searching: target location
- Insertion/deletion: membership changes
- Updating: stored value changes
- Sorting: item reordering
- Merging: sequence combination
- Sorted merge: smaller front first
- [1,4]+[2,3]→[1,2,3,4]
- Destruction: owned storage release

• Records and memory ownership
- C struct: named fields
- Array and pointer fields
- Singly linked node: payload, same-type link
- C declaration: no field initializers
- Modern C++: default member initializers
- new↔delete
- new[]↔delete[]
- Repair links before freeing
- No dereferencing freed nodes

Algorithm properties

• Separate algorithm obligations

- Finiteness: every valid input terminates
- Definiteness: unambiguous instructions
- Effectiveness: executable steps
- Correctness: specified valid-input results
- Finite algorithms can be incorrect

• Problem and algorithm specification
- Inputs, outputs, assumptions
- Procedure language independence
- Implementation language dependence
- Efficiency resource model
- Omitted characteristic ≠ false characteristic

Resource cost models

• Resource cost models
- Time: chosen operations
- Space: memory and live recursive frames
- Distributed communication volume
- Assignment unit ≠ one second
- Sequential costs add
- Independent nested iteration counts multiply
- Conditional worst case: test+costlier branch

• Input-case comparisons
- Equal input sizes
- Best, average, worst cases
- Average requires distribution
- Expected total=m f(n)
- Independence unnecessary
- Exact coefficients depend on operation counting

Asymptotic bounds

• Asymptotic bound meanings
- Eventually nonnegative functions
- O: $f(n) \leq c\, g(n)$, $n \geq k$

- c>0
- Ω: reversed inequality
- Θ: upper and lower bounds
- o: f/g→0
- ω: f/g→∞
- Best/average/worst-case functions
- Space functions also bounded

• Polynomial bounds and witnesses
- $3n^2+4n+1 \leq 8n^2$, $n \geq 1$
- $O(n^2)$: c=8,k=1
- Ignore constants and lower-order terms
- Transitivity: f≤cg≤cdh
- Both thresholds required

• Worked example Increasing growth rates
- 1<log n<n<n log n
- n log n<n^2<n^3
- n^3<2^n<3^n
- 3^n<n!<n^n
- Log bases: constant-factor difference

• Exam trap Tightness versus valid upper bounds
- Linear functions also $O(n^2)$
- Tightest listed order usually intended
- Mixed O/Θ options may overlap

Loops and hidden access costs

• Loop and stack counting
- i*=2: about $\log_2 n$ iterations
- Independent nested n-loops: $\Theta(n^2)$
- Constant work per iteration
- n recursive levels: Θ(n) stack
- Constant live storage per level

• Hidden indexed-access costs
- List implementation matters

- Java LinkedList.get(i): nearer-end traversal
- get(0),get(2),...: $\Theta(N^2)$ total
- Many middle indices: linear traversal each
- Iterator scan: $\Theta(N)$
- Damaged return: runtime uncertainty
- Missing cost convention: no exact runtime

Singly linked chains | Exam focus

- Singly linked chain representation
  - Node: data and next
  - Head: first node
  - Noncircular terminator: null
  - Empty: head==null
  - Dynamic node allocation
  - Pointer storage overhead
  - No direct index arithmetic
- Linked-chain traversal
  - Follow next from head
  - Stop at null
  - Display/count: process every node
  - Counting: $\Theta(n)$
  - Maintained size avoids recounting
  - Array-linked lists still follow links
  - Separate positional index exception

Safe linked-list updates | Exam focus

- Safe insertion and head deletion
  - Prepend: new.next=head; head=new
  - Insert-after: new.next=p.next before p.next=new
  - Head-only append: tail traversal
  - Maintained tail avoids append traversal
  - Delete-head: save old=head
  - Advance head=head.next

- Release old in C++
- Deletion boundary cases
  - Singly linked tail needs predecessor
  - Known tail insufficient
  - Empty and single-node cases
  - Only-node removal: head=tail=null
  - Successor deletion: bypass before deallocation
  - Clear owned chain: visit/release every node
- Three-pointer chain reversal
  - Initially prev=null, cur=head
  - Save next=cur.next
  - Redirect cur.next=prev
  - Advance prev=cur; cur=next
  - Finish head=prev
  - Linear time
  - Constant extra pointers
- Worked example Insertion link-order trace
  - 3→7→9; insert 5 after 3
  - First 5→7, then 3→5
  - Reversed assignments lose 7→9

List variants and practical choices | Exam focus

- Doubly linked and circular lists
  - Doubly linked nodes: prev
  - Head/tail: constant-time end operations
  - Singly linked tail deletion: predecessor needed
  - Circular tail.next=head
  - Traversal stops on return to start
  - Handle empty circular list separately
- Practical linked-list selection
  - Changing sequences with known insertion positions
  - Position search: linear
  - Polynomial terms: coefficient, exponent

    - Directory and block links
    - Nonbinary-tree child lists
    - Hash chains
    - Browser history: doubly linked cursor
    - Playlist: circular option
    - Airline seats: specify record model
- Index access comparison
    - Array: O(1)
    - Singly linked list: O(n) worst
- Front insertion comparison
    - Array: O(n) shifts
    - Singly linked list: O(1)
- Append comparison
    - Array spare capacity: O(1)
    - Linked list with tail: O(1)
    - Linked list head only: O(n)
- Tail deletion comparison
    - Array: O(1)
    - Singly linked list: O(n)
    - Predecessor required

Stack implementation conventions | Exam focus

- Stack convention and operations
    - LIFO=FILO
    - Push: insert
    - Pop: remove/return top
    - Peek: observe top
    - Zero-based occupied-slot top
    - Empty: top=-1
    - Full: top=C-1
    - Check before A[++top]=x
    - Check before returning A[top--]
- Alternative stack implementations

- Count-based empty: size=0
- Count-based push: A[size++]=x
- Keep empty/full conventions consistent
- Fixed-array push/pop: O(1)
- Linked stack: head insertion/deletion O(1)
- Singly linked tail pop: predecessor traversal

• Exam trap Stack failure conditions

- Overflow: insertion into full storage
- Underflow: removal from empty storage
- isEmpty check ≠ underflow

Stack traces and applications | Exam focus

• Stack operation tracing

- Record bottom-to-top after each instruction
- Record popped values separately
- Push 1,2; pop→2; remaining [1]
- Complete output: 2,2,1,1,2

• Stack applications

- Nested calls: latest returns first
- Bracket matching: openings pushed
- Closing bracket matches top
- Reversal: push input then pop
- Undo: latest action→redo
- Redo: action→undo
- New edit after undo: clear redo
- DFS/backtracking: latest pending choice

Expression notation and evaluation

• Expression notation and conversion

- Infix: operator between operands
- Precedence and parentheses
- Prefix: operator first
- Postfix: operator last

- Fixed arity determines grouping
- Infix→postfix: output operands immediately
- Stack pending operators
- Pop higher precedence before pushing
- Left-associative: pop equal precedence
- Parentheses override operator order

• Postfix evaluation and expression trees
- Push operands
- Binary operator: pop right then left
- Push left op right
- Prefix tree traversal: preorder
- Postfix tree traversal: postorder
- Stack useful, other representations possible

• Worked example Expression conversion traces
- A+B*C-D/E→ABC*+DE/-
- 8 3 -: pop 3 then 8
- 8−3=5
- (A+B)*C→AB+C*
- Prefix: *+ABC

FIFO queue operations | Exam focus

• Ordinary queue policy
- FIFO=first-come, first-served
- Enqueue at rear
- Dequeue/peek at front
- Linked head/tail operations: O(1)
- Append tail, remove head
- First insertion: set both pointers
- Last removal: clear both pointers
- Head-only insertion: tail traversal

• Queue applications and modeling
- Equal-priority printer jobs
- Shared-resource waiting

- Asynchronous buffers
- Level-order exploration
- Play sequence
- Airline seats versus waiting requests
- Application name insufficient for representation

• Worked example Conditional queue trace
- Enqueue 4,8; peek=4
- peek>5 false: no removal
- Enqueue 1,6; dequeue
- Result: [8,1,6]

Circular buffer conventions | Exam focus

• Circular buffer count convention
- Noncircular rear exhaustion: false overflow
- Released front slots reusable
- Modulo wraparound
- Front: next removal
- Size: item count
- Insertion slot=(front+size)%C
- Empty: size==0
- Full: size==C

• Circular dequeue and alternate convention
- Read A[front]
- front=(front+1)%C
- Decrease size
- Alternative: reserve one unused slot
- Full: next(rear)==front
- Different empty test and capacity
- State convention before tracing

• Worked example Physical versus logical queue order
- Capacity six, initially full
- Remove 11; enqueue 9 at slot 0
- Physical: [9,13,15,17,19,21]

- Logical: [13,15,17,19,21,9]
- Noncircular version may overflow

Deques and queue policies | Exam focus

- Deque versus dequeue
  - Deque: insertion/removal at both ends
  - No guaranteed middle access
  - Linked endpoints: O(1)
  - Linked middle indexing: O(n)
  - Circular-array deque: possible indexed access
  - Dequeue: removal operation
- Queue combination and capacity policies
  - Merge: concatenate or alternate fronts
  - Predicate split: matching output queue
  - Append preserves subgroup order
  - Full buffer: reject, block, overwrite
  - Capacity alone specifies no policy

Priority queues and tie rules | Exam focus

- Priority queue service
  - Minimum or maximum priority convention
  - Urgency before arrival
  - Interleaved insertions and extractions
  - Binary heap peek: O(1)
  - Binary heap insert/extract: O(log n)
- Priority queue implementations and ties
  - Unordered sequence: cheap insertion
  - Unordered sequence: linear extremum removal
  - Ordered sequence: reversed tradeoff
  - Equal priority ≠ inherent FIFO
  - Stable service: (priority,arrival_counter)
  - Specify priority direction and tie rule
- Exam trap FIFO tie-policy trap

- Guide assumes undefined stability
- FIFO ties: additional policy
- No universal priority-queue stability

Tree vocabulary and binary shapes

• Rooted-tree vocabulary
- Root: no parent
- Other nodes: one parent
- Connected and acyclic
- n nodes: n−1 edges
- Unique path between each pair
- Depth: root-to-node edges
- Height: maximum depth
- Single-node height: zero
- Leaves: no children

• Binary-tree shape distinctions
- Binary: at most two children
- Full: zero or two children
- Complete: earlier levels filled
- Complete: last level fills leftward
- Perfect: leaves at same level
- Perfect: two children per internal node
- Balanced: logarithmic height under stated rule
- Balanced ≠ necessarily perfect

BST ordering and search | Exam focus

• Global BST ordering invariant
- Distinct keys assumption
- Entire left subtree smaller
- Entire right subtree larger
- Immediate-child checks insufficient
- 10→left 5→right 12: invalid
- Duplicates: count or placement policy

- BST operations and search costs
  - Compare; follow one child
  - Search ends at equality or null
  - Insertion: same path, new leaf
  - Operation cost: O(height)
  - Balanced: logarithmic; chain: linear
  - Root hit: $\Theta(1)$ best case
  - Optimal BST: access probabilities
  - Minimize expected search cost
  - Construction distinct from searching
- Optimal-BST dynamic programming
  - Every key interval
  - Try each possible root
  - $O(n^2)$ intervals × O(n) roots
  - Construction: $O(n^3)$
  - Probability-weighted search depth
  - Objective ≠ minimum worst-case height
- Worked example BST insertion shape trace
  - Insert 40,20,50,10,30,60
  - 40 children: 20,50
  - 20 children: 10,30
  - 50 right child: 60
  - Height: two edges
  - Sorted 1,2,3,4,5: right chain

Tree traversal rules

- Tree traversal rules
  - Preorder: node,left,right
  - Inorder: left,node,right
  - Postorder: left,right,node
  - Recursive application; null does nothing
  - Sorted inorder requires BST
  - Preorder: copying

- Postorder: children-before-parent deletion
- Level order: FIFO queue

• Traversal worked tree

- Root 4; children 2,6
- Leaves 1,3,5,7
- Preorder: 4,2,1,3,6,5,7
- Inorder: 1,2,3,4,5,6,7
- Postorder: 1,3,2,5,7,6,4
- Level order: 4,2,6,1,3,5,7
- Randomized visitation possible
- Randomized ≠ standard named DFS order

BST deletion methods

• BST deletion cases

- Leaf: clear parent link
- One child: parent bypass
- Two children: copy inorder replacement
- Predecessor: left-subtree maximum
- Successor: right-subtree minimum
- Delete replacement at original position
- Replacement deletion: at most one child

• BST deletion by merging

- Promote right subtree
- Attach old left at right minimum
- Symmetric left promotion
- Root deletion: update root
- Copying versus reconnecting: different valid shapes
- Name deletion method before tracing

Balanced search trees

• AVL balancing and rotations

- Balance factor=height(left)−height(right)
- Allowed factors: −1,0,1

- Insertion/deletion: update heights upward
- LL: right rotation
- RR: left rotation
- LR: left-child left, node right
- RL: symmetric double rotation
- Rotations preserve inorder keys

• Balanced search-tree guarantees
- 30,20,10: LL imbalance
- Repair root 20; children 10,30
- Red-black colouring discipline
- Logarithmic height
- Avoid linear worst-case search
- Ordered ranges versus ordinary hashing

Heap order and indexing

• Heap shape and order
- Binary heap: complete tree
- Max-heap: parent≥children
- Min-heap: reversed order
- No BST-style subtree ordering
- Max-heap root: maximum
- Distinct keys: second largest root child
- Duplicate maxima: distinct runner-up may deepen
- [9,9,9,8,7,6,5]: runner-up 8, index 3
- Minimum may occupy any leaf
- Minimum search: linear worst case

• Heap array indexing
- Zero-based root: 0
- Zero-based children: 2i+1,2i+2
- Zero-based parent: floor((i−1)/2), i>0
- One-based children: 2i,2i+1
- One-based parent: floor(i/2)
- Child index < current heap size

- Exam trap Heap indexing convention trap
  - Right child 2i+1: one-based
  - Zero-based heap indexing
  - Explicit convention change required

Heap restoration and construction | Exam focus

- Heap restoration and construction
  - Insert at next complete-tree position
  - Swim while parent order violated
  - Root removal: final element→root
  - Reduce heap size
  - Sink with more dominant child
  - Stop when heap order restored
  - Swim/sink: logarithmic paths
  - Bottom-up internal-node sinking: O(n)
  - Repeated insertion construction: O(n log n)
- Worked example Max-heap removal trace
  - Start [10,8,6,2,1,4,5]
  - Remove 10; move 5; sink
  - Result [8,5,6,2,1,4]
  - Remove 8; move 4; exchange 6
  - Result [6,5,4,2,1]

Addressing and hash functions

- Direct addressing versus hashing
  - Direct addressing: every possible key
  - Finite universe U: $\Theta(|U|)$ slots
  - Hash address h(key)∈0..m−1
  - Actual entries plus bucket capacity
  - h(k)=k mod 6
  - 36 home address: 0
  - Collision may change final storage slot
- Hash-function properties

- Determinism: equal keys, same chosen-function address
- Uniformity: evenly spread expected keys
- Universal hashing: random suitable-family function
- One function chosen per table
- Choice independent of fixed key set
- No unrelated fresh function per lookup
- Collision probability controlled
- Zero collisions not guaranteed

Collisions, probing, load factor | Exam focus

• Collision handling and load
  - Separate chaining: bucket lists
  - Load α=n/m
  - α: average chain length
  - Expected O(1+α): suitable distribution
  - Worst case: O(n)
  - Open addressing: entries inside array
  - Probe until key or available slot
• Probing schemes and deletion
  - Linear probing: (h(k)+j)%m
  - m=7; keys 10,17,24
  - Home 3; final positions 3,4,5
  - Quadratic probing: changing step
  - Double hashing: (h1(k)+j h2(k)) mod m
  - Step relatively prime to m
  - Relatively prime step: all-slot coverage
  - Collisions remain possible
  - Deletion: tombstones or reinsertion
  - Avoid premature search termination
• Linear-probing wraparound
  - Seven slots; home address 3
  - Probe order: 3,4,5,6,0,1,2
  - Home slot ≠ guaranteed final slot

Maps, sets, dictionaries, caches | Exam focus

- Maps, sets, lists: contracts
  - Map: unique keys→values
  - Different keys may share values
  - Map entries traversable
  - Set: no duplicate elements
  - Null policy: implementation-dependent
  - Iteration order: implementation-dependent
  - Lists preserve positions
  - Lists generally allow duplicates
  - Java LinkedList permits null
- Dictionaries, symbol tables, caches
  - Hashing: expected O(1) lookup
  - Balanced tree: ordered O(log n) lookup
  - Bounded cache: eviction policy required
  - FIFO eviction: queue
  - LRU: hash+doubly linked recency list
  - Hit: move to recent end
  - Full: evict least-recent end
  - Best structure depends on operations/policies

Linear search | Exam focus

- Linear search cost cases
  - Successive target comparisons
  - Stop at equality or exhaustion
  - Sorted or unsorted data
  - First match: Θ(1)
  - Absent/last match: n comparisons
  - Worst time: Θ(n)
  - Uniform successful positions: (n+1)/2 average
  - Average remains linear
- When scanning suffices
  - Single unsorted query: sorting rarely justified

- Small lists: setup costs matter
- Mean: examine all values
- Variance: examine all values
- Arbitrary parity counts: examine all values
- Sorting avoids none of these scans

Binary search | Exam focus

- Binary search interval invariant
  - Ascending random-access sequence
  - lo=0; hi=n−1
  - mid=lo+(hi−lo)//2
  - Equality: finish
  - target<A[mid]: hi=mid−1
  - Otherwise lo=mid+1
  - Failure: lo>hi
  - Descending order: reverse comparisons
  - Comparable keys; finite valid bounds
  - Fixed preallocation unnecessary
- Binary search costs and variants
  - Worst probes=$\lfloor \log_2 n \rfloor + 1$, $n > 0$
  - 100 keys: seven probes
  - Million keys: about twenty probes
  - Iteration: constant space
  - Recursion: logarithmic stack
  - Linked midpoint traversal loses logarithmic total
  - Boundary variants: retain candidate at equality
- Worked example Binary search worked interval
  - Target 19 in [11,13,15,17,19,21]
  - mid=2: 15
  - Keep indices 3..5
  - mid=4: 19
  - Binary debugging: ordered good/bad boundary
  - Arbitrary fault locations insufficient

Search variants and assumptions | Exam focus

- Interpolation search assumptions
  - Numeric index estimation
  - lo+(hi−lo)×(key−A[lo])/(A[hi]−A[lo])
  - Round to valid index
  - Guard equal endpoints
  - Guard out-of-range targets
  - Sorted, suitably uniform data
  - Expected probes: O(log log n)
  - Pathological distribution: linear
  - Binary search: logarithmic worst case
  - Binary search: no uniformity requirement
- Other sorted-search variants
  - Ternary: compare two split points
  - Retain relevant third
  - Ternary search: logarithmic
  - Jump search: block-boundary probes
  - Then scan selected block
  - Regular intervals: unit steps or jumps
  - Interpolation also requires sortedness
  - Prerequisite MCQ options may overlap

Sorting properties and cost comparison | Exam focus

- Sorting properties and comparison assumptions
  - Stability: preserve equal-key input order
  - In-place: fixed extra element storage
  - Total auxiliary space includes recursive stack
  - Stable/in-place alone identify no unique algorithm
  - Specify implementation, input case, resource
- Bubble sort with early stopping
  - Best: n
  - Average/worst: n^2
  - Constant extra storage

- Stable

- Selection sort profile
  - Best/average/worst: n^2
  - Constant extra storage
  - Unstable
- Insertion sort profile
  - Best: n
  - Average/worst: n^2
  - Constant extra storage
  - Stable
- Quicksort profile
  - Best/average: n log n
  - Worst: n^2
  - Average stack: log n
  - Worst stack: n
  - Unstable
- Ordinary array mergesort profile
  - All cases: n log n
  - Linear extra storage
  - Stable with left-first ties
- Heapsort profile
  - All cases: O(n log n) bound
  - Iterative sink: constant storage
  - Unstable
- Exam trap Overlapping bounds and logarithm placement
  - $n \log(n^2)=2n \log n$
  - Same growth as n log n
  - Different from $n(\log n)^2$
  - Big-O options can overlap

Bubble sort | Exam focus

- Bubble-sort adjacent passes
  - Ascending: swap only left>right

- Left-to-right pass: remaining maximum→end
- Shrink unsorted boundary each pass
- Up to n(n−1)/2 pair comparisons
- Average/worst: quadratic

• Bubble optimizations and stability
- Early stop: no-swap pass finishes
- Sorted input with early stop: linear
- Fixed pass schedule: quadratic
- Strict swaps preserve ties
- Cocktail sort: bidirectional bubble passes
- Cocktail stability: strict swaps

• Worked example Bubble first-pass trace
- [4,1,3,2]→[1,4,3,2]
- Then [1,3,4,2]
- Then [1,3,2,4]
- Maximum 4 reaches final position

Selection sort | Exam focus

• Selection-sort comparison and exchange counts
- Scan suffix minimum
- Exchange minimum with A[i]
- n−1 meaningful placements
- n(n−1)/2 comparisons regardless of order
- At most n−1 distinct-position exchanges
- Final self-swap: n swap calls possible

• Selection-sort suitability and instability
- Expensive record moves; cheap comparisons
- Large dataset alone insufficient
- [2a,2b,1]→[1,2b,2a]
- Equal-key order reversed
- Constant element storage: nonunique property

Insertion sort | Exam focus

- Insertion-sort sorted-prefix invariant
  - First element initially sorted
  - Save next value
  - Shift larger predecessors right
  - Insert into gap
  - n−1 insertion passes
  - Sorted input: one check, no shifts
  - Sorted time: linear
  - Reverse shifts: 1+2+…+(n−1)
  - Reverse time: quadratic
- Insertion-sort adaptiveness and stability
  - Few inversions: few shifts
  - Nearly sorted or small arrays
  - Strictly larger shifts preserve ties
  - Linked implementation avoids array shifts
  - Linked insertion position still requires search
- Worked example Insertion into sorted prefix
  - Insert 2 into [1,4,7]
  - Save 2
  - Shift 7 before 4
  - Write after 1
  - Result: [1,2,4,7]

Shell sort | Exam focus

- Shell-sort gap passes
  - Positive gaps ending at 1
  - Gap h: insertion-sort h-spaced subsequences
  - Earlier long moves reduce disorder
  - Final h=1 finishes sorting
  - h-sorted: $A[j] \le A[j+h]$
  - Both indices must exist
- Shell-sort complexity and stability

- Bounds depend on gap sequence
- Knuth increments: 1,4,13,...
- Knuth worst order: O(n^(3/2))
- Distant moves can reorder ties
- Generally unstable
- Constant extra element storage

• Worked example Shell-sort gap trace
- Start [5,4,3,2,1]
- Gap 3: positions (0,3),(1,4)
- After gap 3: [2,1,3,5,4]
- After gap 1: [1,2,3,4,5]

Quicksort partition rules | Exam focus

• Quicksort partition balance
- Choose pivot, partition, recurse
- Balanced partitions: n log n
- Repeated extremes: n+(n−1)+...
- Extreme-pivot time: quadratic
- Expected n log n: input/random-pivot model
- Random versus median-of-three tradeoffs
- No universally best pivot policy

• Partition invariants versus exact arrays
- Ten distinct values; final pivot 5
- Five smaller before; four larger after
- Internal order depends on partition algorithm
- Multiple arrays may satisfy invariant
- Name algorithm for exact trace
- Same-array rearrangement
- Worst recursive storage: linear

Mergesort and external sorting | Exam focus

• Merge-sort recurrence and stability
- Split halves, sort, merge

- Advance smaller current front
- Left-first ties preserve equal-key order
- Ordinary array buffer: linear
- Logarithmic levels × linear merge work
- Sorted input: n log n
- Merge-skipping optimization exception

• Merge variants and external sorting

- Top-down: recursive
- Bottom-up: iterative widths 1,2,4,…
- In-place merging variants: different tradeoffs
- Linear merge: operation, ambiguous sort label
- Internal sorting: dataset fits memory
- External sorting: sorted disk runs
- Buffered stream merges reduce I/O
- Reverse input ≠ maximum merge comparisons

Heapsort | Exam focus

• Ascending in-place heapsort trace

- Build max-heap
- Exchange root with final active element
- Shrink active heap
- Sink new root
- Removed maximum joins sorted suffix
- Repeat until one active item

• Heapsort costs and distinguishing mechanism

- Bottom-up build: linear
- Repeated extraction: O(n log n) worst
- Iterative sink: constant auxiliary storage
- Long root-to-end exchanges reorder ties
- Generally unstable
- Any comparison sort can compare priorities
- Heap extraction: distinctive mechanism

Radix sorting | Exam focus

- LSD radix sort
  - Fixed-width digit keys
  - Least-significant digit first
  - Move left with stable digit passes
  - Stability preserves earlier digit ordering
  - d digits; radix r
  - Counting passes: O(d(n+r))
  - Auxiliary storage: O(n+r)
- MSD radix and key assumptions
  - Leading digits first
  - Subdivide equal-prefix groups
  - Strings: lexicographic comparison
  - Accessible digits required
  - Specify length and signed-value conventions
  - Arbitrary unstructured keys unsuitable
  - Large radix/long keys/small n: comparison alternative

Graph models and representations | Exam focus

- Graph relationship models
  - Vertices V; edges E
  - Directed: u→v distinct from v→u
  - Following accounts: directed
  - Mutual friendship: undirected
  - Weighted edges: costs
  - Unweighted shortest paths: edge counts
  - DAG: directed acyclic graph
  - Trees: connected acyclic graphs
  - Graph structures may contain cycles
- Graph representation costs
  - Matrix space: $\Theta(V^2)$
  - Matrix edge lookup: O(1)
  - Adjacency-list space: $\Theta(V+E)$

- Lists enumerate neighbours
- Dense simple graphs: near-quadratic edges
- Every-edge work increases with density
- Vertex-only work needn't inspect every edge

Breadth-first search | Exam focus

- BFS discovery discipline
  - Mark source on enqueue
  - Dequeue; examine neighbours
  - Unvisited neighbour: mark and enqueue
  - Store predecessor[v] on discovery
  - Predecessors reconstruct paths
  - FIFO: distance d before later layers
- BFS guarantees and costs
  - Unweighted minimum-edge paths
  - No arbitrary minimum-weight guarantee
  - Adjacency lists: O(V+E)
  - Matrix scan: $O(V^2)$
  - Disconnected graph: restart at unvisited vertices
  - Neighbour order changes tied traversal
  - Minimum distances unchanged
- Worked example BFS alphabetical-neighbour trace
  - Edges A-B,A-C,B-D,C-D
  - Queue [A]→[B,C]→[C,D]→[D]→[]
  - Discovery: A,B,C,D
  - D distance: two

Depth-first search | Exam focus

- DFS latest-pending-work discipline
  - Mark vertex
  - Recursively finish each unmarked neighbour
  - Then continue later neighbours
  - Explicit stack alternative

- Queue changes discipline
- Marking prevents cyclic infinite repetition

• DFS guarantees, applications, costs
- Alphabetical recursive discovery: A,B,D,C
- Reachability witness, not necessarily shortest
- Adjacency lists: O(V+E)
- Visited state: O(V)
- Possible stack depth: O(V)
- Cycle and connectivity analysis
- Directed cycle test: active versus completed

Shortest paths and spanning trees | Exam focus

• Dijkstra shortest-path conditions
- Nonnegative weights
- Source distance 0; others ∞
- Finalize smallest unsettled distance
- Relax outgoing edges
- dist[v]=min(dist[v],dist[u]+w(u,v))
- Min-priority queue: next vertex
- Equal edge costs: BFS sufficient
- Negative edges: justified alternative
- Bellman-Ford alternative

• Minimum spanning tree objective
- Connected undirected weighted graph
- Minimum total connection cost
- Prim: cheapest current-tree crossing edge
- Kruskal: increasing edge weights
- Reject already-connected endpoints
- MST ≠ shortest source routes
- Equal weights: multiple MSTs possible

• Worked example Triangle MST and shortest paths
- AB=1; BC=2; AC=10
- Prim/Kruskal: AB,BC

- MST total: 3
- Dijkstra from A: B=1
- C: 10→3 through B

Kadane and KMP

- Kadane maximum contiguous subarray
  - Required nonempty answer
  - Initialize ending=best=A[0]
  - ending=max(x,ending+x)
  - best=max(best,ending)
  - Linear time
  - Constant working storage
  - best=0 silently allows empty answer
  - All-negative nonempty case fails
- Knuth-Morris-Pratt pattern matching
  - Pattern inside text
  - Preprocess longest proper prefix/suffix lengths
  - Mismatch: prefix-length fallback
  - Avoid restarting comparisons from beginning
  - Text pointer needn't move backward
  - Preprocessing+scan: O(pattern length+text length)
- Worked example Kadane and KMP worked values
  - Input [-2,3,-1,4,-5]
  - Ending values: −2,3,2,6,1
  - Best=6 from [3,−1,4]
  - ABAB prefix lengths: [0,0,1,2]

Missing data and misleading labels

- Trace completeness and answer-key limits
  - Complete operation sequence
  - Initial state
  - Capacity
  - Output orientation

- State assumptions before calculation
- Highlighted key cannot remove valid overlaps

• Implementation versus best-choice questions
- Can implement versus best under costs
- ADT permits multiple representations
- Serial-access memory ≠ inherently LIFO
- Heap file: unordered database records
- Heap file unrelated to binary heap
- Insertion: available space
- Lookup: scanning possible
- Ordered output: sorting or index
- Beginning insertion unnecessary

Web Programming

Revision Outlines

Browser, server and technology roles | Exam focus

- Internet, Web and service roles
  - Internet: connected networks
  - Web: resource system, commonly HTTP
  - Browser/user agent: requests and HTML/CSS rendering
  - Permitted JavaScript execution
  - HTTP server: requests→representations
  - Server may invoke application code
  - Client/server roles can share one machine
- Technology responsibilities
  - HTML: structure/meaning; CSS: presentation
  - JavaScript: programmable interactivity
  - PHP; Python; JSP/Java; ASP.NET
  - Server responses and database access
  - C#: commonly ASP.NET
  - Node.js: JavaScript runtime, not language
  - JavaScript on client or server
  - Browser sandbox; permission-based file/device APIs
  - Nontrivial programs possible
  - No unrestricted arbitrary client-hardware access

Hosting, URLs and static versus dynamic content | Exam focus

- Hosting, addressing and URLs
  - Hosting: serving environment
  - Domain registration: reserved name
  - DNS: names→network records
  - ISP: connectivity
  - URL: scheme, host, optional port
  - Path; query; fragment

- Host: machine/service name
- Fragment: local document location

• Static versus dynamic content

- Static server: stored files
- Dynamic: logic/request/data-built representations
- Static pages may run interactive scripts
- Dynamic content need not visibly change
- Website: resource group; page: document/resource
- Open standards ≠ identical browser support
- FrontPage; Macromedia HomeSite5: historical editors
- Editors ≠ languages or modern requirements

Read an HTTP/1.1 exchange | Exam focus

• HTTP/1.1 message anatomy

- Request: method, request-target, version
- Headers→blank line→optional body
- GET /courses?id=7 HTTP/1.1
- Host: example.edu; terminating blank line
- Body may be absent
- Response: version, status, optional reason
- Headers→blank line→permitted content
- Request line ≠ response status line

• HTTP representation, framing and state

- Content-Type: representation format
- Content-Length/transfer framing: message boundaries
- Stateless request semantics
- Sessions/databases maintain application state
- HTTP2/3 encode differently
- Request-line universality requires HTTP/1.x scope
- Asynchronous client use still receives responses

Methods, status and GET/POST reasoning | Exam focus

• HTTP methods and intended effects

- GET: safe representation retrieval
- Intentional state changes: avoid GET links
- POST: process supplied content
- PUT: create/replace identified representation
- DELETE: remove resource association
- Safe: intended state effects
- Idempotent: repeated same intended effect
- Same response not required
- GET/PUT/DELETE idempotent; arbitrary POST not

• Status codes and form transport
- 200 success; 201 creation; 204 no-content
- 301 permanent redirect; 400 malformed request
- 401 missing/invalid authentication; 403 refusal
- 404 unavailable/not found; 500 unexpected failure
- GET forms: URL query name/value pairs
- POST forms: body name/value pairs
- Neither method encrypts
- POST body visible in developer tools
- Server/configuration imposes size limits

HTTPS and resource-oriented APIs | Exam focus

• HTTPS and REST constraints
- HTTP over TLS
- Channel confidentiality/integrity; certificate-based server authentication
- No input-safety or merchant-trust guarantee
- REST: client/server separation
- Stateless requests; cacheability; uniform resource interface
- Resources: identifiers and representations
- JSON common, not required
- JSON output alone ≠ REST

• Resource API reasoning
- /students/7: GET read, PUT replace
- DELETE requests removal

- Authentication; authorization; validation
- Consistent error representations
- Retry safety before repeating requests
- Durable facts stored server-side
- Stateless requests can use persistent databases

HTML document, text and metadata | Exam focus

- HTML document structure and metadata
  - HyperText Markup Language
  - <!DOCTYPE html>: modern standards mode
  - html root; head metadata; body content
  - title: page/tab title
  - Meta charset: encoding
  - Description: summary; viewport: mobile viewport behavior
  - h1-h6: heading hierarchy
  - h6 conventionally smallest default
  - CSS size not mandated
- Text semantics and element anatomy
  - p: paragraph; br: line break
  - hr: thematic break
  - pre: preserved whitespace; code: code
  - strong: importance; em: stress
  - b/i semantics ≠ importance/emphasis
  - sub/sup: subscript/superscript
  - <!-- comment -->: hidden display, visible source
  - Entities: © < > &
  - Tags: boundaries; element: content/attributes
  - Void element example: img

Links, images and accessible meaning | Exam focus

- Links and browsing targets
  - Read
  - href: destination, not url/link

- href="#details"→id="details"
- _self: current; _blank: new context
- _parent: parent; _top: top-level
- Browser preference: tab versus window

• Images and accessibility
- img src: source; alt: alternative
- Informative images: meaningful alternative text
- Decorative images: empty alt
- Width/height reserve dimensions
- title ≠ replacement for alt
- Anchor may wrap image
- Programmatic labels; clear links; keyboard access
- Visual styling alone ≠ accessibility
- Attributes in opening tags
- Unique document-identifying IDs

Lists, tables and navigation | Exam focus

• Lists and navigation
- ul: unordered; ol: ordered
- li: list items
- Ordered list: start and numbering type
- CSS list-style-type: presentation
- dl: definition list
- dt: term; dd: description
- Semantic nav: list of links

• Tables and navigation reset
- table→tr rows→th/td cells
- th: header; td: data, not table
- colspan: columns; rowspan: rows
- Account for previously occupied cells
- Accessible header associations
- CSS borders, background, padding, border-collapse
- Legacy bgcolor/bordercolor not preferred

- nav ul: list-style-type:none;margin:0;padding:0
- Reset removes bullets/default spacing only
- Reset ≠ horizontal links/dropdown

Form controls and successful submission | Exam focus

• Form submission and controls
- action: processing URL
- method: submission; enctype: encoding
- Default method: GET
- Named enabled successful controls contribute values
- Unchecked checkbox/radio usually absent
- label for→control id
- name: submitted field identifier
- Same-name radios: single choice
- Checkboxes: independent selections
- select→option; textarea: multiline

• Input behavior and validation limits
- size: approximate visible text width
- maxlength: entered-text length limit
- Neither establishes server security bound
- Password input masks, never encrypts
- submit: submission; button: no intrinsic submission
- email/required: browser validation feedback
- Arbitrary clients bypass validation
- GET/POST ≠ universal per-element parsing

Frames, media and graphics

• Legacy frames and iframe
- frameset/frame: separate window documents
- Old frameset replaces body
- Named targets select changed frame
- Independent scrolling leaves other frames stationary
- Frameset/frame obsolete in modern HTML

- iframe embeds document within ordinary page
- Untrusted iframe: appropriate permissions/sandboxing

• Media and graphics
- audio/video: supported media, controls, fallback
- embed/object may embed supported resources
- Generic video-tag question may be nonexclusive
- bgsound/Flash history ≠ current support
- Canvas: scripted bitmap drawing context
- SVG: scalable DOM shapes
- Canvas getContext("2d")→fillRect(x,y,w,h)
- Paths: beginPath/moveTo/lineTo/stroke
- Informative graphics require text alternatives

CSS attachment and selector meaning

• CSS attachment and syntax
- External: link rel="stylesheet" href="styles.css"
- External link in head
- Internal: style; inline: style attribute
- External reuse, consistency, caching
- No universal smaller-page/accessibility guarantee
- selector { property: value; }
- CSS comments: /* ... */

• Selector meanings
- p: type; .notice: class; #logo: ID
- h1,h2: grouped selectors
- nav ul: descendants; nav>ul: direct children
- [type="email"]: attribute match
- :hover/:focus: pseudo-class states
- ::before: pseudo-element part
- Selector chooses declaration targets
- Selector ≠ color-setting property

Resolve cascade, specificity and inheritance | Exam focus

- Cascade winner procedure
  - Origin, importance, layer context first
  - Then specificity and source order
  - Same-origin/layer author context assumed
  - ID>class/attribute/pseudo-class>type/pseudo-element
  - Counts compared lexicographically
  - Not unlimited base-ten total
  - Equal specificity: later applicable rule
  - Important beats normal within relevant context
- Specificity, importance and inheritance
  - p#intro.note example
  - #intro blue beats .note red
  - p green!important beats both normal rules
  - Inherited color/fonts: no stronger specified value
  - Margin/padding/border normally not inherited
  - :not(): argument specificity
  - Module universal-zero simplification incorrect

Style properties and box calculations | Exam focus

- CSS properties and box parts
  - color: text; background-color: background
  - font-size: size; font-family: face
  - font-style: italic; font-weight: weight
  - text-align: inline-content alignment
  - text-decoration: underline effects
  - Margin outside border
  - Padding inside border
  - Border surrounds content/padding
- Box-model calculations and relative units
  - Content-box: width200,padding10,border2→224px
  - Border-box width200→content176px
  - Margins excluded from border-box width

- Shorthand: top/right/bottom/left
- One: all; two: vertical/horizontal
- Three: top/horizontal/bottom; four: each side
- Adjacent vertical block margins may collapse
- Padding never collapses
- Non-replaced inline width/height differs from blocks
- rem: root font size
- font-size em: parent font size
- Many other em lengths: element computed-font-size

• Historical body color attributes
- text: default text color
- bgcolor: background
- Modern CSS: color and background-color

Layout, alignment and responsive design | Exam focus

• Flow, flex, grid and alignment
- Normal flow: blocks/inlines
- Flexbox: direct children, main axis
- justify-content: main; align-items: cross
- Horizontal row: vertical cross-axis alignment
- Grid: explicit rows and columns
- grid-template-columns:1fr 2fr
- vertical-align: inline/table-cell contexts
- vertical-align ≠ universal block centering
- align-content: extra-space track/line distribution
- align-content ≠ arbitrary single-child centering

• Positioning and responsive design
- Relative: offset, retained flow space
- Absolute: removed from flow, containing block
- Positioned ancestor often establishes containing block
- Fixed: commonly viewport
- Flexible layout/images; viewport metadata; media queries
- @media(max-width:600px){.cards{grid-template-columns:1fr}}

- min-width768px: wider viewports, not mobile-only
- Compatibility: feature tests and fallbacks
- Compatibility ≠ responsive size handling

JavaScript syntax, inclusion and declarations

- JavaScript inclusion and language identity
  - External .js: <script src="app.js"></script>
  - JavaScript source: no HTML closing-tag syntax
  - defer: after parsing, before DOMContentLoaded
  - Originated at Netscape; distinct from Java
  - Prototype-based objects; first-class functions
  - Sophisticated applications supported
- JavaScript lexical and declaration rules
  - Case-sensitive identifiers
  - Initial letter/underscore/$; no digits/spaces
  - var: function/global scope
  - let/const: block scope
  - const: no rebinding, object mutation allowed
  - // line comments; /* */ blocks
  - Semicolons and automatic insertion
  - Newlines can affect parsing after return
  - "use strict": directive string, not keyword
  - Parse syntax before output computation

Types, conversions and missing values

- JavaScript value categories
  - Primitives: undefined,null,Boolean,Number,BigInt,String,Symbol
  - Objects: arrays, functions, dates
  - Float not separate Number primitive
  - Arrays technically objects
  - typeof new Date()→"object"
  - typeof null→"object": historical behavior
- Missing values and conversion

- Declared unassigned/missing ordinary property→undefined
- Undeclared direct read→usually ReferenceError
- Either string quote style; appropriate escapes
- String-involved + may concatenate
- Arithmetic / converts numeric strings
- "abc"/"def"→NaN
- Nonzero Number/0→signed Infinity; 0/0→NaN
- parseInt("12.8",10)→12, stops at dot
- Parsing ≠ rounding
- Global isNaN converts first
- Number.isNaN does not coerce

Trace assignments, comparisons and postfix updates | Exam focus

• Operator tracing
  - Record value and type
  - Specified evaluation order
  - ==/!=: coercion
  - ===/!==: no coercion
  - Strict operators coexist with loose operators
  - condition?x:y selects one branch
• Assignment, concatenation and postfix microtraces
  - var a="20";var b=a=30
  - Both Number30; a+b=60
  - Supplied key50 incorrect
  - "1 plus 1 is "+1+1
  - Left-to-right result: "1 plus 1 is 11"
  - Postfix a++ yields old value
  - Increment follows yielded value
  - a=10;a==a++→10==10;body sees11
  - a=1;print a-- then a→10
  - JS sequencing not portable to C/C++
  - C/C++ unsequenced modifications trap

Math, string and array API microtraces

- Math and substring traces
  - Math.abs(-9)=9
  - Math.floor(5.9)=5
  - Math.sqrt(36)=6
  - Zero-based indexes
  - substring(start,end): start inclusive, end exclusive
  - "exitexam".substring(2,6)→"itex"
- Array operations and reduction
  - push/pop: end; unshift/shift: beginning
  - slice: selected shallow copy
  - splice(start,deleteCount,...items): mutating removal/insertion
  - reverse: in-place reversal
  - reduce: sequential left-to-right accumulator
  - reduceRight: sequential right-to-left, not parallel
  - [1,2,3].reduceRight((a,v)=>a-v,0)
  - Accumulators: −3,−5,−6
  - No initial value: last present item
  - Empty without initial value: throws
  - Exact method spelling and case

Control flow, functions and errors | Exam focus

- Loop execution order
  - for: initialize once→test→body→update
  - while: pretest
  - do...while: at least one execution
  - break exits; continue advances iteration
  - n=1;n<7;n++ with n%5==0
  - Message only at n=5
- Functions and error handling
  - function f(x){return x*2;}
  - Parameter; return value; stored/passed functions
  - Arrow: compact, different this/constructor behavior

- Runtime argument limits reject absolute unlimited claim
- try/catch: thrown runtime errors
- throw: signal; finally: cleanup
- Syntax versus runtime versus logical errors
- bool ans=true invalid; let ans=true valid
- Literal "type of" ≠ typeof operator

DOM selection, creation and safe updates | Exam focus

• DOM selection and timing
- DOM: hierarchical document nodes
- getElementById("x"): element or null
- querySelector(".item"): first match
- querySelectorAll: static NodeList
- Other plural lookups may be live
- Guard missing elements
- Run after relevant markup exists

• DOM creation, attachment and safe mutation
- createElement("div")→textContent="Hello"→appendChild(el)
- Creation alone does not insert
- createTextNode: text; appendChild: attachment
- innerHTML parses/replaces descendants
- innerHTML+= reparses/replaces previous descendants
- Listener/state destruction possible
- Untrusted plain text: textContent
- style.backgroundColor: style property
- classList.add: class mutation
- getElementByld: lowercase-l misspelling, nonexistent method

Events and validated submission | Exam focus

• Events and listener options
- Events: click,input,keydown,mouseover,submit,load
- addEventListener("click",handler): register, not call
- Multiple listeners coexist

- once: remove after use
- capture: earlier propagation phase
- passive: promise no default cancellation
- Event bubbling ≠ browser default action

• Submission validation and cancellation
- submit listener checks valid()
- Failure only: e.preventDefault()
- addEventListener return false does not cancel
- Inline/onsubmit return false: caller-dependent cancellation
- Cancelable keyboard default: preventDefault
- Different key event need not disappear
- Browser checks: feedback
- Repeat validation/authorization server-side

Window, dialogs and browser properties

• Window, dialogs and browser properties
- window: browsing context; document: document property
- alert: information; confirm: Boolean
- prompt: text or cancellation null
- console.log: diagnostics
- location.reload(): current-page reload
- screen.availHeight; screen.availWidth; screen.colorDepth
- Exact camelCase; ColorsDepth invalid

• Globals, load and document hazards
- window.onload: load-handler property
- Load typically waits dependent resources
- Global let/const not window properties
- Classic top-level var bindings differ
- document.write: method/function property
- Post-load write may replace document
- document.bgColor: legacy property
- Document.bgcolor: wrong object/case
- Modern: document.body.style.backgroundColor

- Workers: concurrent computation
- DOM normally page main thread

PHP syntax, names and types

- PHP syntax and legacy delimiters
  - PHP: Hypertext Preprocessor
  - .php file; <?php ... ?>
  - Final closing tag optional
  - Emits ordinary text and HTML
  - PHP8 rejects <script language="PHP"> delimiters
  - Short <? configuration-dependent
  - Prefer full opening form
  - Stray >> not closing tag
- PHP variables, types and constants
  - $ then letter/underscore
  - Subsequent letters/digits/underscores
  - $AGE18 valid; $18AGE invalid
  - Case-sensitive variable names
  - Dynamic typing; optional type constraints
  - Scalars: int,float,string,bool
  - Structured: arrays/objects; null: no value
  - var_dump: type and value
  - Constants normally not redefined during execution

PHP output, strings and comparisons

- PHP output and concatenation
  - echo: language construct, no return
  - echo: comma-separated expressions
  - print: one expression, returns1
  - printf/print_r: other output mechanisms
  - Neither inserts spaces/newlines automatically
  - print5 twice→55
  - "I"."am"→Iam

- Dot concatenates; + arithmetic

- PHP string and comparison traces
  - Double quotes: variables and escapes
  - Single quotes: dollar names normally literal
  - Literal $lfc: single quotes/escaped dollar
  - Zero-based offset: "maroon"[2]→r
  - Surrounding PHP must be valid
  - strlen: bytes
  - strpos: integer0 or false
  - Failure test: ===false
  - strrev: reverse bytes; str_replace: replace matches
  - 8==8.0 true; 8===8.0 false
  - echo false→empty; echo true→1
- Raw versus rendered PHP output
  - PHP sends markup characters
  - Browser interprets response
  - echo "A
B": raw tag, displayed lines
  - Malformed </br>: HTML error recovery
  - Specify raw versus rendered trace

PHP scope, parameters and persistent locals | Exam focus

- PHP scope and static storage
  - Outside variable: global scope
  - Function locals/by-value parameters: separate bindings
  - Explicit global $x or $GLOBALS["x"]
  - Superglobals available across scopes
  - Static local persists across lifecycle calls
  - Static storage ≠ merely visibility scope
  - Uninitialized static local: null
  - Echo uninitialized static: no characters
- Parameter and static-counter traces
  - Global $i=10; sample($i) by value
  - Function increments/prints local parameter

- Caller remains10
- Before,during,after→10,11,10
- Commas only when source writes them
- External global declaration ≠ reference parameter
- Teaching taxonomy: local,global,static
- Static counter0; successive increments→1,2

PHP arrays, loops and callable behavior | Exam focus

- PHP array keys and syntax
  - Integer/string keys map to values
  - Unspecified numeric indexes start0
  - [20,12,23,19,18]: index3→19
  - ["top"=>"black"]: top→black
  - Contiguous => required
  - Split = >: parse error
- PHP loops and dynamic behavior
  - foreach($items as $key=>$value): entries
  - for/while: tests and updates
  - $sub-=$sub means $sub=$sub-$sub
  - Self-subtraction ≠ decrement
  - $sum=0;foreach([2,4]) accumulation→6
  - Functions: arguments, return, output
  - Classes/objects/variables also support server processing
  - No single mechanism defines dynamic PHP
  - Explicit braces clarify nesting

Request data, server validation and injection | Exam focus

- PHP request sources and existence
  - $_GET: query; $_POST: supported form POST
  - $_FILES: upload metadata
  - $_SERVER: request/environment
  - $_COOKIE: received cookies
  - $_SESSION: session data after startup

- $GLOBALS: global-symbol array
- $_GLOBAL: not built-in
- $_REQUEST: configured GET/POST/cookie combination
- Ambiguous precedence; prefer intended source
- $_REQUEST ≠ database interface
- isset: exists and non-null
- isset ≠ truth or valid syntax

• Secure request-processing sequence
- Read→required/type/length/range checks
- Authenticate→authorize→process
- SQL binding separates values from commands
- Contextual HTML/attribute/JS output encoding
- textContent for browser plain text
- Injection/XSS testing detects, never itself fixes
- Client validation/encryption/connection cannot replace controls

Cookies and browser storage | Exam focus

• Cookies and PHP headers
- Browser-held name/value data
- Automatic attachment only to matching requests
- Domain; path; expiry; Secure; SameSite
- HttpOnly: restricted script access
- Secure: secure transport required
- Cookies not indiscriminately sent everywhere
- setcookie sends response header
- Before output unless buffering defers output
- $_COOKIE: received values

• Browser storage and session interpretation
- localStorage: origin-specific strings, persistent until cleared
- sessionStorage: page session/tab scope
- Neither automatically transmitted like cookies
- Scripts can explicitly send stored values
- IndexedDB: structured local storage

- Cookies/localStorage/sessionStorage/IndexedDB: client storage
- localStorage tokens: script-access risk
- Storage limits vary
- Reopen login: persistent/remember token or restoration
- Stored data alone ≠ server acceptance

Server sessions and the authorization trust path | Exam focus

- PHP server-session lifecycle
  - session_start(): start/resume, enable $_SESSION
  - Browser: opaque session identifier
  - Session data: server-controlled storage
  - Missing views→1; later requests increment
  - unset: one entry
  - Destroyed session data ≠ cleared browser cookie
  - Current variables separately cleared
  - Logout handles complete lifecycle
- Session authorization trust path
  - Client logged_in=1 cookie: attacker-controlled
  - Flag-only trust permits authorization bypass
  - Valid session ID plus server user/role
  - Check every protected action
  - Unrelated flag edit grants no rights
  - HTTPS ≠ trustworthy client claims
  - Omitted server check: scenario indeterminate
  - Protect IDs; regenerate after login
  - Appropriate expiration; no sensitive URL tokens

Uploads, files and storage safety | Exam focus

- Secure upload workflow
  - POST; multipart/form-data; named file input
  - accept: browser hint, not server enforcement
  - Validate server-side size/content
  - Inspect upload errors

- Use temporary upload path
- Safe generated destination; move_uploaded_file
- Avoid executable storage
- Distrust client filenames/extensions

• File APIs and mode traces
- file_exists: file or directory
- is_file: regular file
- fopen: stream or false
- r: existing read; r+: read/write, untruncated
- w: truncate/create write; w+: also read
- a: append/create; x: exclusive creation
- fclose: release stream
- readfile: emit content, return byte count
- echo readfile(...): also prints count
- Check failures before access
- Constrain paths to permitted locations

Database connections, TLS and actual APIs | Exam focus

• PHP database connections
- MySQL: relational DBMS
- Persistent structured data; concurrent clients
- PHP8: MySQLi or PDO
- Removed mysql_* extension requires migration
- mysqli_connect(host,user,password,database,port,socket)
- Database selection may be deferred
- Defaults make multiple arguments technically optional
- Exactly-one-optional-field claim incorrect

• Database TLS and API signatures
- MySQLi: mysqli_init()→mysqli_ssl_set(...)→mysqli_real_connect(...)
- Required verification configuration
- Nonstandard: mysqli_ssl_connect(); pdo_connect_ssl()
- Nonstandard: mysql_ssl_connect(); secure_connect()
- PDO: driver DSN/options

- Procedural MySQLi, object methods, PDO differ
- affected_rows() alone not generic PHP function

SQL operations, result rows and safe rendering | Exam focus

- SQL operations and identifier generation
  - SELECT retrieves; INSERT adds
  - UPDATE ... SET modifies; DELETE removes
  - WHERE limits matching/affected rows
  - DELETE without WHERE: ordinarily all rows
  - Constraints, permissions, errors still apply
  - CREATE DATABASE/TABLE: storage definitions
  - Primary key identifies rows
  - AUTO_INCREMENT: numeric values, gaps/explicit values possible
  - Not gapless business numbering
- MySQLi results and safe rendering
  - mysqli_query: result object for result queries
  - Other successful statements: true
  - Failure/exception depends on settings
  - mysqli_fetch_assoc: associative rows
  - mysqli_fetch_row: positional; mysqli_fetch_object: objects
  - mysqli_num_rows: result-row count
  - mysqli_affected_rows($conn): changed/inserted/deleted, options-dependent
  - mysqli_insert_id($conn): generated AUTO_INCREMENT value
  - Escape rendered cells; prepare submitted values
  - mysqli_close: connection close

PHP includes, objects and dates | Exam focus

- PHP includes and version-dependent failure
  - include: current-scope evaluation; failure warning
  - require: critical failure
  - PHP8 failed require raises catchable Error
  - Uncaught Error stops normal execution
  - E_COMPILE_ERROR description: older PHP

- include_once/require_once: no repeated inclusion
- No arbitrary user-selected include path

• Objects and explicit time assumptions
- Class: properties/methods; new: instance
- $obj->field: property; $obj->method(): call
- public: external; private: class-restricted
- Constructor initializes instance
- $this: instance-method context
- DateTime/DateTimeZone: time and zone
- diff→DateInterval; DatePeriod iterates intervals
- Explicit zones and calendar assumptions
- Do not assume server-local timezone

AJAX and response handling | Exam focus

• AJAX and XHR workflow
- Asynchronous JavaScript and XML
- Responses: JSON,text,HTML,XML
- Technique, not language/standalone framework
- XML not mandatory
- open("GET",url,true)→register handlers→send()
- Response and error handlers
- onreadystatechange may fire repeatedly
- readyState4: completed
- Then check status and data

• Callbacks, fetch and response ordering
- Callback: runtime/API-invoked later function
- Browser invokes available-response handler
- Remote server never directly executes browser code
- await fetch(url)→response
- Check response.ok; await response.json()
- Handle network and parse errors
- Asynchronous result unavailable to immediate next statement
- Older late response must not overwrite newer

Polling, WebSockets and data formats | Exam focus

- Polling versus WebSocket
  - Short polling: repeated update requests
  - Long polling: hold request until event/timeout
  - Respond, finish, then client reconnects
  - Long polling ≠ same-response continuous unsolicited stream
  - WebSocket: handshake then persistent bidirectional messages
  - AJAX/WebSocket both allow nonblocking interaction
  - Both carry JSON; broad stems nonexclusive
- Formats and transport decisions
  - JSON: objects,arrays,strings,numbers,Booleans,null
  - Double-quoted keys/strings
  - Functions/undefined not JSON values
  - XML: valid interchange alternative
  - REST can return JSON or XML
  - Select latency, direction, connection cost, recovery
  - Asynchronous alone insufficient transport criterion
  - Validate messages; authorize each operation

Node server, event loop and scaling | Exam focus

- Minimal CommonJS Node HTTP server
  - const http=require("node:http")
  - http.createServer((req,res)=>{...}).listen(8000)
  - Incoming requests invoke listener
  - res.writeHead(200,{"Content-Type":"text/plain"})
  - res.end("Hello"): complete response
  - Define startup-log variables
  - Module hostname otherwise undefined
- Event-loop concurrency and scaling
  - Event loop schedules JavaScript callbacks
  - OS/worker facilities: asynchronous work
  - Concurrency: overlapping progress
  - Parallelism: same-instant execution

- One loop thread handles waiting requests
- Long CPU loops/synchronous I/O block callbacks
- Workers/processes/load balancer distribute work
- Shared data/session coordination not automatic
- Multiple clients or overlapping single-client requests

Races, locks and transaction invariants | Exam focus

- Race conditions and synchronization
  - Two requests read stock1
  - Both accept sale and write0
  - Two sales consume one item
  - Interleaving-dependent correctness despite short synchronous callbacks
  - Mutex: critical-section protection
  - Semaphore: bounded access
  - Spinlock: busy-wait
  - Spinlocks unsuitable for browser/event-loop blocking
  - Several mechanisms synchronize; mutex not unique
- Database invariants and retry safety
  - Atomic conditional stock decrement
  - UPDATE stock SET qty=qty-1
  - WHERE id=? AND qty>0
  - Check affected rows
  - Related writes in appropriate transaction
  - Row lock/serializable transaction for multistep logic
  - Conflict rollback/retry
  - Process-local lock misses other balanced servers
  - Idempotency tokens plus unique storage constraints
  - Retried payment must not duplicate transaction

Frameworks and version-aware setup | Exam focus

- Frameworks, libraries and components
  - Framework: conventions/reusable structure
  - Library: callable capabilities

- jQuery: DOM simplification
- React: component interfaces
- Angular: component platform, distinct from AngularJS
- Vue: interactive interfaces
- Responsive-interface description not uniquely identifying
- React props: inputs; state: changing data
- JSX requires transformation
- JSX ≠ ordinary browser JavaScript syntax

• Version-aware framework setup
- Bootstrap: layout/components
- jQuery→Popper→Bootstrap: version4-era order
- Bootstrap5 drops required jQuery
- Angular CLI: ng new,ng serve,ng build
- ng generate; ng test
- e2e support configuration-dependent
- ng serve starts the development server
- Create React App: deprecated historical tooling
- Historical module tooling ≠ universal current recommendation

Manage content, performance and revisions | Exam focus

• Content, persistence, history and deployment
- CMS: editing, publishing, roles, media
- No per-file source edits required
- Database: persistent facts
- Database ≠ entire server application
- VCS: history, branches, merges
- Review and collaboration
- Deployment: publish chosen build
- Deployment separate from version control

• SEO and performance mechanisms
- SEO: meaningful content, headings, metadata
- Accessible links; crawlable structure
- Keyword metadata alone ≠ ranking guarantee

- Caching: reused representation/result, validity/invalidation rules
- Server cache: rendered pages/query results
- Unsafe shared key can expose private content
- Minification: remove source characters
- Bundling: combine; compression: transfer encoding
- Preloading: anticipated early fetch
- Excess bundling/preloading may compete with critical-work
- Measure performance
- Responsive and cross-browser testing separately

Assemble and evaluate an end-to-end feature | Exam focus

• Course-search browser feature
- Semantic headings/navigation; labeled search form
- Responsive grid; single-column fallback
- Submit registration after parsing
- Prevent default for asynchronous implementation only
- Encode query values
- GET /courses?q=...
- Loading state and useful errors
- No untrusted HTML insertion

• Search/enrollment end-to-end verification
- Validate query length/type; authenticate when required
- Parameterized SELECT; bounded results
- JSON with appropriate status/content type
- Client: status→parse→textContent nodes
- Enrollment: POST, CSRF, server authorization
- Capacity checks; transaction; success after commit
- Test empty/bad input and access denial
- Network failure; delayed responses; repeated clicks
- Simultaneous enrollment; narrow screens; keyboard use
- All four blueprint outcomes integrated

Validate syntax, context and output

- Literal source and trace preconditions
  - Valid syntax; known initial values
  - Execution context; raw/rendered representation
  - x=4+"4"5: missing-operator syntax error
  - bool ans=true: invalid declaration
  - type of new Date(): syntax error
  - PHP >> instead of ?>: parsing defect
  - Trace output only for valid syntax
  - Split = > invalid without authorized repair
- Answerability and output interpretation
  - Response text ≠ browser rendering
  - Emitted markup changes displayed output
  - Visual breaks may vanish in plain-text choices
  - Missing correct60 leaves no correct option
  - Storage/interface libraries/script languages/channels overlap
  - Broad descriptions need narrower distinction
  - Record the question number
  - Check assumptions before choosing
- Check exact code and punctuation
  - JavaScript coercion: x=4+"4"
  - Original output44, answer A
  - Extracted extra5: different invalid program
  - PHP closing delimiter: ?>
  - Original output r, answer D
  - Extracted >> falsely creates syntax error
  - Read the code exactly as shown
  - Trace strings and numeric conversions
  - Current extracted text: identify conflict

Operating System

Revision Outlines

Objectives and resource management | Exam focus

- OS objectives and services
  - CPU, memory, storage allocation
  - Device and communication resources
  - Convenience: interfaces, files, device abstractions
  - Efficiency: effective resource use
  - Efficiency ≠ every process's fastest response
  - Development tools use OS services
  - Application development ≠ core resource management
- Evolution and related properties
  - Maintainable interfaces/modules
  - Hardware support, features, repairs
  - Backward compatibility: supported older behavior
  - Compatibility ≠ every old binary
  - Scalability: greater workload/resources
  - Portability: other platforms
  - Fault tolerance: correct service despite failures
  - Recovery: service restoration afterward
  - Updates/monitoring: administrative help
  - Automatic updates: not universally defining

Batch, sharing and real time | Exam focus

- Execution organization
  - Batch: little post-submission interaction
  - Multiprogramming: CPU work during another job's wait
  - Several available jobs
  - Multiprogramming ≠ fixed time slices
  - Time-sharing: rapid scheduling/preemption, interaction
  - Single CPU: interleaved execution

- Multiple CPUs: true parallelism
- Shared memory ≠ mandatory private memories

• System categories and timing
- Single/multiuser: concurrent users, not applications
- Network systems: remote services
- Distributed systems: coordinated machines
- Location hiding possible
- Real-time correctness: timing included
- Hard deadlines: mandatory
- Soft misses: degraded service
- Real time ≠ simply fast
- Spooling/multitasking: mechanisms, not generation proofs

Kernel, shell, privilege and boot | Exam focus

• OS components and privilege
- Kernel: resources, protection, system calls
- Shell: command interpretation
- GUI: another interface
- Browsers, office tools, editors: applications
- Drivers: commonly kernel, sometimes user-space
- Kernel mode: privileged operations
- Supervisor/kernel mode: often synonyms
- Exclusive mode options: ambiguity

• Boot sequence and resident components
- Firmware/bootloader before kernel
- Kernel loaded into RAM
- First core OS component
- Kernel ≠ first power-on program
- Loadable modules: running-kernel driver additions
- Modular systems allow runtime loading
- Resident kernel: session-long activity
- Antivirus/disk tools: utilities
- User processes plus privileged components

APIs, system calls and virtualization | Exam focus

- System call versus library API
  - System call: protected kernel-service request
  - Process creation, file read, socket opening
  - API/library wrappers
  - Library function ≠ mandatory kernel entry
  - Application API versus user/kernel boundary
  - OS-service interface: scope required
- Services and virtual machines
  - Process control
  - File/device management
  - Information retrieval and communication
  - Video editing: application operation
  - write_database(): application-specific, not generic syscall
  - Virtualization: virtual machine environment
  - Hypervisor: guests on one host
  - Virtual memory: process-address translation
  - Kernel/shell/API/VM: distinct labels

Processes, programs and context

- Program versus process
  - Program: stored instructions
  - Process: executing instance
  - Address space and resources
  - Process ≠ universal application manager/UI
  - PCB: identity and state
  - Saved PC/registers: resume execution
  - Scheduling information
  - Memory/accounting details
  - I/O and open-resource state
- Context switch and task terminology
  - Save one context, restore another
  - Application-work interruption: overhead

- Interrupt/syscall: privilege-only switch possible
- Privilege switch ≠ process switch
- Task: OS-specific meaning
- Process/thread/schedulable entity
- Process-versus-task: no universal distinction

Process states, queues and schedulers

• Process-state transitions
- New: created/admitted
- Ready: eligible, not running
- Running: CPU in use
- Waiting/blocked: event required
- Terminated: execution ended
- Running→ready: quantum expiry/preemption
- Running→waiting: blocking I/O
- I/O completion: waiting→ready
- Completion ≠ immediate running guarantee

• Queues and scheduling roles
- Ready queue: eligible work
- Device queues: blocked requests
- Job queue: system processes
- Admission queue: awaiting memory/admission
- Long-term: job admission
- Medium-term: suspension/swapping
- Short-term: ready-work selection
- Dispatcher: handoff/context restore
- Priority preemption: policy-dependent running→ready

Creation, fork/exec and termination

• Process creation mechanisms
- Initialization, user requests, service requests
- Running process’s creation call
- POSIX fork(): child creation

- Child return: 0
- Parent return: child PID
- Success assumption
- Distinct address spaces
- Explicit resource sharing possible
- exec: current image replacement
- exec ≠ new process creation

• Termination categories
- Voluntary: normal completion/program-detected error
- Involuntary: unrecoverable fault/killing
- exit(): normal runtime termination
- abort(): abnormal termination request
- Nonexistent/protected memory: fatal fault
- Bad-input exit: application decision
- Terminate exit: imprecise category
- Parent exit ≠ universal descendant killing

Threads and mapping models

• Thread ownership and sharing
- Own PC, registers, stack
- Shared code and address space
- Usually shared open files
- Process: potentially many threads
- Thread: one process at a time
- Threads ≠ group of processes
- Thread execution need not be sequential
- Independent versus cooperating processes
- Cooperation: communication/shared data

• Thread management and mappings
- User-level: library management
- Kernel threads: OS-known
- Many-to-one: blocking call blocks all
- One context: no simultaneous multi-CPU use

- One-to-one: another thread runs when blocked
- Many-to-many: users multiplexed over kernel threads
- Speed/reliability: implementation-dependent
- Management layer ≠ guarantees
- Browser/Excel names ≠ threading-model proof

IPC and blocking semantics

- IPC mechanisms
  - Shared memory: common region
  - Conflicting updates require synchronization
  - Message passing: send/receive
  - Pipes: streams
  - Message queues: message boundaries
  - Sockets: endpoints
  - RPC: remote operation
  - HTTP: application communication protocol
  - HTTP can carry IPC
- Blocking and nonblocking contracts
  - Blocking send: API-defined wait condition
  - Handoff, buffering or receipt
  - Blocking receive: wait for message
  - Nonblocking: prompt data/progress/would-block
  - Rendezvous: sender waits until receipt
  - File locks: access coordination
  - Locks ≠ arbitrary payload transfer
  - Locking: IPC synchronization
  - Asynchronous messaging ≠ synchronization-free

Races and critical-section correctness

- Race and critical-section criteria
  - Race: conflicting uncoordinated interleavings
  - Critical section: protected shared-state code
  - Critical section ≠ resource itself

- Mutual exclusion: safety property
- Progress
- Bounded waiting
- Stated correctness assumptions
- Failures: races, starvation, deadlock

• Synchronization structure
- Small critical regions
- Same protection for every conflicting access
- Barrier: defined participants reach point
- Monitor: encapsulated protected operations
- Condition synchronization
- Inconsistent lock order: deadlock risk

• Worked example Lost-update example
- Two reads: counter=0
- Each computes/stores 1
- Two increments → 1, not 2
- Entire read-modify-write under one lock

Semaphores and resource counts

• Semaphore operations and models
- Atomic wait/P/down
- Atomic signal/V/up
- Nonnegative wait: consume token or block
- Signal: token supply, possible wakeup
- Negative counts: alternative waiter encoding
- State the count model
- Binary semaphore: one permit
- Counting semaphore: several permits
- Mutex: ownership rules
- General semaphore: no equivalent ownership requirement

• Worked example Semaphore and bounded-buffer example
- Mutex initial value: 1
- wait(m); critical work; signal(m)

- N slots: empty=N, full=0, mutex=1
- Producer: wait empty, then mutex
- Insert; unlock; signal full
- Holding mutex while waiting empty
- Consumer unable to free slot

Turn variables and atomic primitives

• Strict alternation and lock-variable races
- turn=0/1
- Process i: wait while turn≠i
- Enter, then hand turn over
- Alternation enforces turns
- Outside peer: progress violation
- Separate test/set: race
- Both may observe free

• Atomic synchronization primitives
- Test-and-set; compare-and-swap
- Atomic check/update
- Ordinary check-then-write: non-atomic
- Peterson: flags plus turn
- Two processes; memory-model assumptions
- Semaphore: more than two possible
- Local interrupt disabling excludes no other CPUs
- Not general user-space locking
- Interrupts: event notification/synchronization
- Interrupt synchronization exclusions: overbroad

Scheduling metrics and calculations | Exam focus

• Scheduling equations and goals
- A: arrival; S: start; C: completion
- Response=S−A
- Turnaround=C−A
- Single burst B, no I/O

- Waiting=C−A−B
- I/O: subtract CPU and blocked time
- Ready-queue waiting only
- Throughput=completed_jobs/elapsed_time
- CPU_utilization=busy_time/elapsed_time
- Goal-dependent response/turnaround/wait minimization
- Useful throughput/utilization maximization

• Timeline-first calculation method
- Arrivals and idle gaps
- Preemptions and overhead
- Per-process values before averages
- First response ≠ final completion

• Worked example Scheduling-metric example
- A=2; S=5; B=4; C=9
- Uninterrupted CPU burst
- Response=3; turnaround=7; waiting=3
- Six completions/two hours=3 jobs/hour

Preemptive and non-preemptive decisions | Exam focus

• CPU scheduling variants
- Non-preemptive: completion/block/yield ends burst
- Preemptive: running→ready allowed
- Timer quantum or urgent arrival
- FCFS/non-preemptive SJF: non-preemptive
- RR/SRTF: preemptive
- Priority: either variant
- Cooperative: voluntary release

• CPU versus resource preemption
- Save context, reallocate CPU time
- Occupied printer/lock: unsafe withdrawal possible
- Time-slicing ≠ removing no-resource-preemption
- Frequent switches: overhead
- Non-preemptive long bursts delay interaction

FCFS and convoy effects | Exam focus

- FCFS behavior and limits
  - Ready bursts: arrival order
  - Ordinarily non-preemptive
  - Idle CPU: jump to next arrival
  - Convoy: long burst delays short jobs
  - Finite work ahead: eventual start
  - Eventual service ≠ small waiting
  - FCFS ≠ universally worst turnaround
- Worked example FCFS simultaneous-arrival example
  - All arrive 0; P1=24,P2=3,P3=3
  - Tie order: P1,P2,P3
  - P1:0-24; P2:24-27; P3:27-30
  - Waiting 0,24,27 → mean 17
  - Turnaround 24,27,30 → mean 27
  - Different arrivals: recompute actual times

SJF/SJN and remaining time | Exam focus

- Shortest-job scheduling rules
  - SJF/SJN: shortest ready next burst
  - Non-preemptive selection
  - SRTF: reconsider arrivals
  - Shortest remaining work; newcomer may preempt
  - Future bursts: estimates needed
  - SJF minimum-average-wait theorem
  - Fixed simultaneous ready set
  - Known bursts; no preemption/switch cost
  - Theorem ≠ universal preemptive-policy comparison
- Shortest-job traps
  - Repeated short arrivals: long-job starvation
  - Original burst ≠ remaining burst
- Worked example SRTF example
  - P1: arrival 0, burst 8

- P2: arrival 1, burst 2
- P1:0-1; P2:1-3; P1:3-10
- Waiting: P1=2, P2=0
- Non-preemptive SJF: no P1 interruption

Priority, starvation and aging | Exam focus

- Priority scheduling assumptions
  - Highest-priority ready job
  - Numeric convention: smaller or larger wins
  - Arrival preemption: specify variant
  - Continued high-priority load: low-priority starvation
  - SJF: long-job starvation too
  - Priority/SJF options: multiple valid answers
- Aging and priority inversion
  - Aging: waiting raises effective priority
  - Explicit aging rule
  - Workload/priority scheme conditions
  - Reduced indefinite postponement
  - Aging ≠ deadlock cure
  - Inversion: urgent waits on low-priority holder
  - Inheritance: temporarily raise holder priority

Round Robin and quantum bounds | Exam focus

- Round Robin queue behavior
  - Cyclic ready queue
  - At most q per turn
  - Unfinished→tail; completed/blocked→leave
  - Quantum-boundary arrival ordering
  - Large q approaches FCFS
  - Tiny q increases switch overhead
  - Recurring opportunities ≠ equal total consumption
- RR next-turn waiting bound
  - n continuously ready jobs

- No priority bypass; zero overhead
- Next-turn wait≤(n−1)q
- One interval, not lifetime waiting
- Unrestricted arrivals/blocking: bound limitations
- Include specified context-switch costs

• Worked example RR trace example
- P1=5,P2=3; arrivals=0; q=2
- P1:0-2; P2:2-4; P1:4-6
- P2:6-7; P1:7-8
- Completions: P1=8,P2=7
- Waiting: 8−5=3; 7−3=4
- Mean waiting=3.5

Multilevel queues and feedback | Exam focus

• Multilevel scheduling
- Separate foreground interactive/background batch classes
- Between-queue and within-queue selection
- Fixed interqueue priority: lower-queue starvation
- Class time shares: starvation avoidance
- Feedback queues: behavior-based movement
- CPU-intensive demotion
- Periodic priority boosting

• Multilevel trace assumptions and traps
- Plain multilevel ≠ automatic movement
- Initial class
- Quantum per level
- Promotion/demotion rules
- Interqueue policy
- Binary search tree: implementation data structure
- Tree ≠ CPU selection policy
- Infinite: not standard policy category

Fair-share and proportional guarantees | Exam focus

- Fair-share allocation
  - Stated weights
  - Per-user/group versus per-process shares
  - More processes under RR: aggregate advantage
  - Proportional shares: policy choice
  - Fair ≠ necessarily equal
- Lottery, stride and guarantee limits
  - Lottery: tickets and winning selection
  - Weights 3:1 → expected 75%:25%
  - Short-run variation
  - Lottery ≠ deterministic starvation deadline
  - Stride: deterministic share-progress tracking
  - Guaranteed scheduling: specify guarantee/assumptions
  - Finite FCFS/RR: no indefinite postponement possible
- Worked example User-share example
  - A: four ready tasks; B: one
  - Per-process RR: A≈80% overall
  - Per-user halves: A=50%,B=50%
  - Each A task≈12.5%
  - B’s task=50%

Evaluating scheduling claims | Exam focus

- Workload-dependent scheduling comparisons
  - Response, waiting, fairness, throughput tradeoffs
  - Workload and objective first
  - Apply algorithm; compare measured results
  - Burst times alone: insufficient maximum wait
  - Unconstrained arrivals, priorities, overhead
  - Finite workload versus open stream
  - Open stream: potential starvation
- Scheduling-question ambiguity
  - FCFS and SJN: both non-preemptive

- Equal share: process/user/group scope
- Highest turnaround: concrete workload required
- FCFS worst: not absolute
- RR: conventional interactive choice
- Turns improve responsiveness
- Quantum and overhead conditions

Deadlock and necessary conditions | Exam focus

- Deadlock versus blocking/starvation
  - Every member waits on blocked peer
  - Ordinary holder wait: normal blocking
  - Starvation: unfair indefinite delay
  - Starvation need not contain dependency cycle
  - Mutual exclusion: one nonshareable-resource holder
  - Hold-and-wait: retain while requesting
  - No resource preemption: voluntary release only
  - Circular wait: closed dependency chain
- Four-condition interpretation
  - Standard reusable-resource model
  - All four necessary
  - General policies ≠ current deadlock proof
  - Inspect actual allocations and requests
  - Prevent one condition: prevent class
  - CPU preemption can coexist with deadlocks
- Worked example Two-lock deadlock example
  - P1 holds A, requests B
  - P2 holds B, requests A
  - No release before acquisition
  - Nonshareable locks
  - Retained holdings
  - No forced release
  - Circular dependency

Resource graphs and cycle tests | Exam focus

- Resource-allocation graph rules
  - P→R: request
  - R-instance→P: assignment
  - Follow directed waits
  - No cycle → no deadlock
  - Single instance/type: cycle → deadlock
  - Multiple instances: cycle insufficient
  - Available instance may permit completion
  - Wait-for graph: resource nodes replaced
  - P→Q: P waits for Q
- Single-instance cycle example
  - P0 holds R0, requests R1
  - P1 holds R1, requests R0
  - Wait-for edges: P0→P1, P1→P0
  - Cycle: neither proceeds
  - Multiple instances: inspect every assignment
  - Check available instances before conclusion

Prevention and removed conditions | Exam focus

- Deadlock prevention mechanisms
  - Constrain resource requests
  - Safe sharing removes mutual exclusion
  - All-before-holding requests remove hold-and-wait
  - Safe withdrawal/restart attacks no-preemption
  - Global acquisition order removes circular wait
  - Physical resources: sharing/preemption limitations
- Release-and-retry interpretation
  - Release all holdings when waiting
  - Removes simultaneous holding and waiting
  - Voluntary release versus forced preemption
  - Reclaiming description: ambiguous condition target
  - Prevention: utilization waste possible

- Prevention: starvation possible
- Deadlock freedom ≠ fairness

Banker avoidance and safety testing | Exam focus

- Banker safety algorithm
  - Avoidance: resulting state remains safe
  - Known maximum claims required
  - Need=Max−Allocation
  - Work=Available
  - Find unfinished process: Need≤Work componentwise
  - Simulate completion: Work+=Allocation
  - Finish all → safe sequence
  - Request≤Need and request≤Available
  - Tentative grant requires successful safety test
  - Unsafe ≠ already deadlocked
  - Banker: avoidance, not scheduling/detection/prevention
- Worked example One-resource safety example
  - Available=1
  - P1: allocation=1, remaining need=1
  - P2: allocation=2, remaining need=2
  - Finish P1 → Work=2
  - Then finish P2
  - Safe sequence: P1,P2
  - Maximum claims known before testing

Detection, recovery and ignoring | Exam focus

- Detection and recovery
  - Detection: current-state deadlock
  - Terminate selected processes
  - Reclaim safely preemptable resources
  - Roll back to checkpoint
  - Termination/rollback: both recovery methods
  - Recovery-choice questions require criterion

- Repeated victim selection: starvation

- Detection timing and intentional ignoring
  - Every-request versus periodic checking
  - Overhead versus detection-delay tradeoff
  - No universal best interval
  - Ostrich: ignore sufficiently rare problem
  - Real systems: selected prevention/detection
  - Deadlock acceptance: intentional ignoring possible
  - Detection graph versus avoidance safety graph
  - Scheduling queue: different problem

Address binding, loading and linking

- Address-binding stages
  - Compile-time: known placement, absolute addresses
  - Load-time: relocation on placement
  - Execution-time: runtime translation
  - Movement requires hardware support
  - Logical/virtual: process-generated addresses
  - Physical: RAM addresses
  - MMU: logical→physical
  - Reverse-direction bank stem: malformed
- Loading, linking and swapping
  - Dynamic loading: routine on demand
  - Dynamic linking: external references load/runtime
  - Suitable library pages shared
  - Static linking: executable code copies
  - Traditional swapping: whole process→backing store
  - Paging: individual pages
  - Context switch: execution-state change

Contiguous allocation and fit policies

- Contiguous base/limit translation
  - One continuous process region

- Base: process physical start
- Limit: permitted length
- Check 0≤logical_offset<limit
- Physical=base+offset
- Relocation register ≠ normally OS base
- Last-valid-offset limit: changed inequality
- Specify limit convention

• Hole-selection policies
- First-fit: first adequate hole
- Next-fit: resume previous search
- Best-fit: smallest adequate hole
- Worst-fit: largest hole
- Search time versus future utilization
- No universally optimal policy
- Binary-fit: not standard listed policy

• Worked example Fit and boundary examples
- Holes 100,500,200,300; request 180
- First-fit=500; best-fit=200; worst-fit=500
- Base=4000; length=100
- Offset 99→4099
- Offset 100: trap

Internal and external fragmentation

• Fragmentation and consolidation
- Internal: unused allocated-unit space
- External: free total split inadequately
- Contiguous request cannot fit holes
- Coalescing: adjacent free holes joined
- Compaction: move allocated regions
- Consolidate free space
- Compaction requires relocation support

• Allocation-model fragmentation limits
- Fixed-frame paging: no external frame fragmentation

- Final page: possible internal waste
- Variable segmentation/contiguous partitions: external fragmentation
- Paging OS ≠ every allocator fragmentation-free
- Track units; round allocations upward
- Requests need not fill blocks

• Worked example Fragmentation examples
- 10 KiB object; 4 KiB pages
- ceil(10/4)=3 pages
- Allocate 12 KiB; waste 2 KiB
- Two 3 KiB holes: total 6 KiB
- Contiguous 5 KiB request fails

Paging, page tables and TLB

• Paging translation
- Equal fixed page/frame size P
- Page=floor(v/P)
- Offset=v mod P
- Page table: frame plus protection/status
- Physical=frame×P+offset
- TLB: translation cache
- TLB miss: possible table walk
- TLB miss ≠ nonresident page

• Page-table sizing and validation
- Virtual-page count and entry size
- Table size ≠ physical RAM alone
- Check protection and present bits
- Table lookup ≠ CPU scheduling

• Worked example Paging calculation example
- P=1024; v=2053
- Page=2; offset=5
- Frame=7 → physical=7173
- 16-bit virtual space; 1 KiB pages
- 64 virtual pages

- Page number: 6 bits

Segmentation versus paging

- Segment translation
  - Variable logical units: code/stack/arrays
  - Segment table: base/limit/permissions
  - Address pair: segment,offset
  - Validate segment and bound
  - Physical=base+offset
  - Out-of-range offset: trap
  - No spill into next segment
- Basic paging/segmentation contrasts
  - Paging: fixed units, page tables
  - Paging: internal waste
  - Segmentation: variable units, segment tables
  - Segmentation: external fragmentation
  - Segments reflect program structure
  - Ordinary pages: largely application-transparent
  - A/B/C contrasts all true
  - Multiple true contrasts: no unique choice
- Worked example Segment boundary example
  - Segment 1: base=8000, length=500
  - Offset 120→8120
  - Offset 500: invalid
  - Length-based limit convention

Virtual memory and page faults

- Virtual-memory purpose and fault sequence
  - Translated, protected address spaces
  - Nonresident portions: storage backing
  - No added physical RAM
  - Capacity constraints remain
  - Demand paging: load on access

- Nonresident access: trap
- Validate mapping/permissions
- Find frame or select victim
- Save dirty victim if needed
- Obtain contents; update mapping; restart

• Fault exceptions and thrashing
- Page fault ≠ mandatory disk read
- Zero-filled pages; shared mappings
- Illegal access: reject, not load
- Excess active-page demand: thrashing
- Repeated faults dominate computation
- Reduce active load
- Add suitable memory

• Worked example Page-fault cost example
- Access: 100 ns
- Additional page-in: 5 ms
- Fault probability: 10^−4
- Added delay≈500 ns/access
- Rare long delays matter

Page-replacement traces

• Replacement policy rules
- Victim needed: no suitable free frame
- FIFO: earliest loaded page
- FIFO hit: insertion time unchanged
- LRU: least recently accessed
- LRU recency: every access
- Optimal: farthest future use
- Optimal: offline lower bound
- LFU: reference counts
- LFU tie/aging rules required

• Replacement trace method and properties
- One row per reference

- Identify hits before eviction
- Initial loads count as faults
- Belady’s anomaly: more frames, more faults
- FIFO may exhibit anomaly
- LRU/optimal: stack property
- Page buffering: useful free/modified pages retained
- Reduced effective fault cost

• Worked example FIFO versus LRU example
- Three empty frames
- References: 1,2,3,1,4,1
- FIFO: faults 1,2,3,4,last 1
- FIFO total: five faults
- LRU: refresh 1 on hit
- Evict 2 for 4
- Last 1: hit
- LRU total: four faults

Controllers, drivers and I/O services

• I/O hardware and software roles
- Controller: hardware status/control/data registers
- Driver: OS requests→device operations
- Device independence and uniform naming
- Hide brand-specific details
- Error handling: supported reports/retries
- Polling: repeated readiness checks
- Interrupts: CPU attention requests
- DMA: setup then block transfer
- No per-byte CPU work

• I/O responsibility boundaries
- Kernel/driver answer: abstraction scope required
- Plug-and-play: compatible hardware discovery/configuration
- Driver may still be needed
- Input/output/storage/NIC: managed hardware

- Accounts: identities
- Paper/network errors: report/handle
- Total power loss: hardware support required
- OS software alone insufficient

Buffering, caching, spooling and clocks

• Buffering versus caching versus spooling
- Buffer: temporary in-transit data
- Absorb speed/transfer-size differences
- Cache: copies for faster reuse
- Spool: queued device jobs
- Printer jobs: usually disk-backed
- Producers avoid continuous device ownership
- Combined implementation, different purposes

• Timers, monitoring and disk responsibilities
- Timer: timekeeping and timeouts
- Accounting and preemptive scheduling
- Monitor: CPU/memory/storage/network usage
- Measurement ≠ resource creation
- Disk initialization/filesystem creation: OS/tools
- Boot begins: firmware
- Bad-sector remapping: drive firmware possible
- OS-all-disk-functions claim: scope boundaries

Files, descriptors and operations

• File content, metadata and handles
- Metadata: name, size, owner
- Permissions and timestamps
- open: obtain reference
- read/write: byte transfer
- seek: offset change
- close: release reference
- truncate: size change

- unlink: name removal
- Descriptor: process-local integer handle
- Open-file state: offset/access mode
- Descriptor ≠ offset

• Read semantics and filesystem formats
- Seekable read: offset advances
- Short reads possible
- Zero read: normally EOF
- Sockets: ordinary read/write supported
- socket/connect/send: socket operations
- read ≠ uniquely non-network call
- FAT32/NTFS/ext4: filesystem formats
- HTTP: application communication protocol

Directories, mounts and links

• Directory organizations
- Single-level: unique names, one namespace
- Two-level: users separated
- Tree: nested directories
- Absolute and relative paths
- Links: shared graphs
- Modern systems: commonly hierarchical
- Feature label ≠ excluding simpler arrangements

• Links and filesystem attachment
- Hard link: same file, another name
- Usually within one filesystem
- Symlink: stored pathname
- Cross-filesystem targets; dangling links possible
- Unlink one name ≠ content deletion
- Remaining names/open references retain content
- Mount: attach filesystem at mount point
- Mount ≠ format
- NFS: remote filesystem access

- FTP: transfer, not itself mounting

File-block allocation methods

- Contiguous, extent and linked allocation
  - Contiguous: consecutive blocks
  - Logical block i→start+i
  - Direct access
  - Growth and external fragmentation problems
  - Extents: several contiguous runs
  - Linked: block chains
  - Growth without whole-file contiguous reservation
  - Block i: traverse preceding links
  - FAT: chains in table
  - Cached FAT ≠ direct arithmetic mapping
- Indexed allocation and inode lookup
  - Per-file pointer array/tree
  - Direct lookup; metadata overhead
  - Unix inode: direct and indirect pointers
  - Pointer size and block size
  - Index capacity
  - Metadata-read caching assumptions
- Worked example Index-capacity example
  - 1024-byte index; 4-byte pointers
  - 1024/4=256 block addresses
  - Data blocks: 1024 bytes
  - One level: 256 KiB data
  - Exclude metadata/reserved entries

Protection, authentication and threats

- Identity and access control
  - Authentication: verify identity
  - Authorization: permitted operations
  - ACL: principals and rights

- Owner/group/other: simpler bit model
- Owner full access: not universal
- Everyone read: not universal
- Read/write/execute/append/delete: distinct rights
- Filesystem-specific permission rules

• Security controls, threats and interfaces
- Least privilege: limited damage
- Password hashing: no stored plaintext
- One-time passwords: expire after use
- Encryption: confidentiality with correct keys
- Encryption ≠ automatic integrity
- Trojan: useful-software disguise
- Worm: independent propagation
- Virus: attached host
- DoS: availability harm
- Cryptography/authentication/firewalls: product-dependent support/configuration
- Interfaces: CLI, GUI, speech/natural language, VR
- Interface list: not closed

Linux, Android and history limits

• Kernel, distributions and product licensing
- Linux: Unix-compatible, written from scratch
- Linus Torvalds
- Distribution: kernel/userland/libraries/tools
- Android base kernel: Linux
- Framework/applications: higher layers
- AOSP: open source
- Devices: proprietary apps/services possible
- Windows/macOS: proprietary products
- macOS: open-source components too
- Both proprietary options: ambiguous answer
- Microsoft Office: application suite

• Product and historical-claim limits

- Windows: Microsoft, after MS-DOS desktop era
- Solaris: Sun origin
- First widely used: definition/evidence required
- First multitasking: definition/evidence required
- UNIX ≠ first ever multitasking
- Generation/function pairing: historical evidence
- Mobile majority: population/date required
- Architecture facts ≠ market-share proof

Repairing ambiguous bank questions

- Missing or multiple answer concepts
  - Requested concept omitted
  - Several valid examples: reject uniqueness
  - System call: kernel-service request
  - Process: executing program
  - CPU/memory/I/O: multiple managers
  - Device/process/file/security choices: missing umbrella
  - Virtual memory: isolation, translated space
  - Virtual memory ≠ unlimited memory
- Minimal wording repairs
  - Specify OS versus API
  - Scheduling variant
  - Arrivals and quantum
  - Exact process/resource graph
  - Base-limit convention
  - Comparison scope
  - Marked answer ≠ evidence
  - Missing edges: unrecoverable from theory

Software Engineering

Revision Outlines

Engineering processes and work products | Exam focus

- Software engineering and process
  - Systematic development, operation, maintenance
  - Product: executable code plus artifacts
  - Requirements, designs, test assets
  - Configuration and user documentation
  - Process: activities, ordering, roles, products
  - Entry/exit conditions
  - Specification → design/implementation → validation → evolution
- Decomposition and programming approaches
  - Procedural: functions
  - Object oriented: state/behavior objects
  - Either iterative or sequential
  - OOP doesn't guarantee shorter projects
  - Reuse, encapsulation, consistent models
  - Reduced duplication, easier change
  - Poor object design still complex
  - Functional: function evaluation, controlled side effects
  - Event-driven: reacts to events

Lifecycle and process selection | Exam focus

- Lifecycle phases and design levels
  - Requirements → design → implementation → testing
  - Deployment → maintenance
  - Analysis: clarify problem
  - System design: architecture
  - Object/detail design: interfaces, algorithms
  - Implementation: code
  - Testing: behavior evaluation

- Deployment: into service
- Activities overlap and repeat

• Contextual process selection
- Requirement stability and technical risk
- Delivery urgency, stakeholder availability
- Team skills and assurance needs
- Inexperience alone insufficient
- Risk-driven cycle, prototype, planned stages
- Choice depends on actual uncertainty
- Testing/documentation required in every process

Waterfall and V-model

• Classical and modified waterfall
- Planned phases, reviewed deliverables
- Stable understood requirements, familiar technology
- Late feedback raises change cost
- Controlled revisiting remains possible
- Modified waterfall: explicit feedback
- Milestone visibility
- Risk of frozen incorrect assumptions

• V-model stage/test pairing
- Requirements → acceptance tests
- System design → system tests
- Component/detail design → integration/unit tests
- Coding at bottom
- Early test planning
- Stable requirements, strong assurance fit
- Shape/process guarantees neither safety nor correctness

Iteration and incremental delivery

• Iteration versus increment
- Iteration: revisit and improve
- Increment: add usable functionality

- Processes may combine both
- Registration → payments → reporting: incremental
- Registration-prototype feedback revisions: iterative
- Each increment: design, implement, test

• Incremental-delivery benefits and limits
- Prioritized partial capability delivered early
- Smaller changes: easier debugging, feedback
- Architecture/integration still planned
- Iteration needn't rebuild whole system
- Increments needn't be independent modules
- Early version: prototype/iterative/Agile possibilities
- Purpose determines classification

Agile values and Scrum

• Agile values and work descriptions
- People/collaboration over rigid procedures
- Working results over exhaustive documents
- Customer participation over contracts
- Adaptation over fixed plans
- Right-hand values still matter
- Planning/documentation still needed
- User story: small need and value
- Acceptance criteria support story
- Epic: broader work grouping

• Scrum timeboxes and assurance
- Sprints: at most one month
- Sprint Goal, selected work, usable Increment
- Increment meets Definition of Done
- Scope discussion before/during planning allowed
- Fixed timebox, adapted content
- Sprint Goal protected
- Agile with assurance/documentation possible
- Safety-critical alone doesn't prove unsuitability

Spiral development and risk

- Risk-driven spiral cycle
  - Objectives and alternatives
  - Assess/reduce important risks
  - Develop/verify appropriate next result
  - Plan further commitment
  - Prototype risky technology before large investment
  - Risk determines work amount/type
- Spiral selection and limitations
  - Risk emphasis distinguishes spiral
  - Changing requirements alone insufficient
  - Incremental/prototyping/sequential activities incorporable
  - Uniquely most-general: textbook convention
  - Not universal theorem
  - Risk analysis needs skill
  - Low-risk small projects: excessive overhead possible

Prototypes, RAD and reuse

- Prototyping purpose and lifecycle
  - Requirements, usability, technical feasibility
  - Throwaway: discard after learning
  - Evolutionary: deliberately refine toward production
  - Quick versions may omit robustness/security/architecture
  - Evaluate before unchanged deployment
  - Prototyping may include coding
  - Construction label doesn't exclude coding
- RAD and reuse-oriented development
  - RAD: rapid construction, timeboxing
  - User participation, reusable components
  - Available users, skilled designers/developers
  - Reuse: search components → evaluate fit
  - Possible requirements adjustment
  - Design around interfaces, build gaps

- Product-line requirements/architecture/components/tests reusable

Feasibility and requirement levels | Exam focus

- Feasibility versus system analysis
  - Worth doing and deliverable?
  - Technical, economic, operational, schedule constraints
  - Existing limitations, organizational impacts
  - Benefits, cost, available skills
  - Integration risk
  - System analysis: detailed problem/behavior
  - Activities inform each other
- Requirement levels and SRS
  - Business: purpose and goals
  - User: high-level stakeholder needs
  - System: detailed services and constraints
  - Software Requirements Specification: agreed requirements
  - Module natural language: client communication
  - Analysis models: technical formalization
  - Real specifications may include diagrams

Functional rules and prohibitions | Exam focus

- Functional obligations and prohibitions
  - Observable service or behavior
  - Input/error reactions
  - Forbidden actions included
  - Intrusion sensor → alert
  - Manager-only refund approval: functional
  - Negative statement ≠ automatically nonfunctional
- Precise requirement formulation
  - Actor/trigger, action, observable result
  - Generate PDF report: functional
  - No other-customer balance disclosure: functional security
  - Broad security quality versus implementing functions

- Must/should/not-work wording: classification may remain ambiguous

Nonfunctional qualities and constraints | Exam focus

- Nonfunctional attributes and constraints
  - Performance: response time/throughput/resource use
  - Reliability: failure-free service
  - Availability: ready for use
  - Usability: task success/ease of learning
  - Security: protection
  - Compatibility: cooperating environments
  - Portability: moving environments
  - Maintainability/supportability: change, diagnosis, evolution
- Measurable qualities and classification traps
  - Search ≤2 seconds, 100 concurrent users
  - Performance requirement
  - New-user checkout ≤1 minute, unaided
  - Usability requirement
  - Use Java: technology/pseudo-requirement
  - Often separately classified system constraint
  - Dependability: reliability, safety, robustness
  - Portability not usual central dependability category
  - Peak-load degradation: primarily performance
  - Not automatically reliability

Requirement quality and traceability | Exam focus

- Requirement quality criteria
  - Correct: actual stakeholder needs
  - Complete: agreed scope, normal/exceptional behavior
  - Consistent: no contradictions
  - Clear/unambiguous: one intended interpretation
  - Feasible/realistic: achievable within constraints
  - Verifiable: objective fulfillment evidence
  - Replace vague fast/most users

- Defined metric, workload, threshold

• Traceability and conflict negotiation
  - Backward: source/rationale
  - Forward: design, code, tests
  - Requirement dependencies recorded
  - Traceability matrix links artifacts
  - Impact analysis support
  - Not merely cost/schedule ledger
  - Negotiate incompatible stakeholder demands
  - Independent incompatible approvals insufficient
  - Elicitation difficulties: scope, understanding, volatility

Elicitation, FAST and QFD | Exam focus

• Elicitation techniques
  - Interviews: stakeholder needs
  - Document analysis: manuals, contracts, records
  - Observation/ethnography: actual daily work
  - Prototypes: tacit expectations
  - Workshops/JAD: users, managers, developers
  - Facilitator and recorder
  - Consensus-building
  - Multiple sources, observed-work assumption checks

• FAST, QFD and brainstorming
  - FAST: environment elements, outputs, inputs lists
  - Consolidate lists → smaller specifications
  - QFD: customer-value prioritization
  - Normal needs: explicit
  - Expected needs: assumed
  - Exciting needs: unanticipated benefit
  - Brainstorm ideas before evaluation/prioritization
  - Not every idea becomes committed requirement

Scenarios and analysis objects | Exam focus

- Scenarios, models and analysis objects
  - Scenario: one concrete execution
  - Use case: related success/alternative scenarios
  - Functional model: use cases/scenarios
  - Structural/object model: classes
  - Dynamic model: states/interactions
  - Entity: domain information, Incident
  - Boundary: interaction mediation, ReportForm
  - Control: use-case coordination
  - UI form normally boundary, not entity
- Object identification and CRC
  - Nouns → candidate objects
  - Possessives → candidate attributes
  - Verbs manages/reports-to → candidate associations
  - Heuristics, not automatic parsing rules
  - CRC: Class, Responsibilities, Collaborators
  - Control per use-case/actor: starting heuristic
  - Not mandatory UML rule/fixed lifetime

Actors, goals and system scope | Exam focus

- Actors and external roles
  - Person, device or another system
  - One person: multiple roles
  - Multiple people: one role
  - Primary actor: seeks goal
  - Supporting actor: provides service
  - Identify main-function user groups
  - Administration/maintenance roles
  - External information-exchanging systems
- Use-case diagram scope
  - Actors outside boundary rectangle
  - Use-case ovals inside

- Goal name: verb plus noun
- Roles, goals, relationships
- No full algorithm/UI layout
- Internal component versus external actor
- Chosen system boundary determines role

Use-case flows and guarantees | Exam focus

- Complete use-case specification
  - Name/goal, actors, trigger
  - Preconditions
  - Main success flow
  - Alternatives/exceptions
  - Postconditions
  - Active subject-verb steps
  - Consistent abstraction level
  - Errors separated from normal flow
- Submit Incident scenario
  - Precondition: authenticated officer
  - Officer supplies location/description
  - System validates → records → returns ID
  - Missing location: problem and correction request
  - Failure: save nothing
  - Success: incident and ID persist
  - Failure guarantee: no partial record
  - Alternate flow: conditional variation
  - Exception flow: failure
  - Preconditions before entry, trigger starts execution

Include, extend and generalization | Exam focus

- Include versus extend
  - Include: required reused behavior
  - Submit Order includes Validate Order
  - Dashed include arrow: base → included

- Extend: conditional extension-point behavior
- Apply Discount extends Checkout
- Extend arrow: extension → base
- Base meaningful independently
- Withdrawal extends Print Receipt: reversed example

• Generalization and abstract use cases

- Specialized actor/use case → general
- Hollow triangle toward general
- Abstract: behavior for specialization
- Not directly instantiated complete use case
- Reuse alone insufficient definition
- No direct actor line insufficient definition
- Included behavior executes when inclusion reached
- Base conditional branches not necessarily all executed

UML classes and multiplicity | Exam focus

• Class diagrams and UML

- Class name, attributes, operations
- + public, - private, # protected
- ~ package visibility
- Class diagram: static types/relationships
- Object diagram: particular instances/links
- OMG maintains/standardizes UML
- Modeling language, not lifecycle process
- Structural things: conceptual/physical structural elements

• Multiplicity interpretation

- One end: objects per opposite-end object
- 1: exactly one
- 0..1: optional
- 0..*: any nonnegative number
- 1..*: at least one
- Read each end separately
- One-to-many doesn't imply composition

Association, aggregation and composition

- Association, dependency and aggregation
  - Association: related instances
  - Dashed dependency: client uses supplier
  - Generalization: solid line, hollow triangle
  - Triangle toward superclass/general element
  - Shared aggregation: hollow diamond
  - Diamond at whole end
  - Weak whole-part meaning
  - No detailed lifecycle policy implied
- Composition ownership and diamond placement
  - Filled diamond at whole end
  - At most one composite per part
  - Owned parts destroyed with composite
  - Defined detachment/transfer possible
  - Never-independent claim oversimplified
  - Car composing Engine: diamond beside Car
  - Points-to-Engine wording imprecise
  - Bank intended relationship: composition

Sequence diagrams and emergency reporting | Exam focus

- Sequence diagram semantics
  - Interactions across lifelines
  - Order/time generally downward
  - Messages between participants
  - Activation rectangles: execution
  - Creation message starts lifeline
  - X: modeled destruction
  - No garbage-collector timing guarantee
  - Synchronous call: wait
  - Asynchronous message: sender proceeds
- Emergency-report interaction example
  - Module heuristic: actor → boundary → control

- Entities follow
- Not mandatory UML column order
- FieldOfficer → ReportEmergencyButton
- EmergencyReportForm: enter details
- AcknowledgmentNotice: later response
- DispatcherStation: dispatcher side
- Initiating element: ReportEmergencyButton

Activity, state and data-flow models | Exam focus

- Activity and state-machine notation
  - Activity: workflow
  - Filled circle: start
  - Rounded actions
  - Guarded diamond: decision/alternative merge
  - Thick bars: fork/join
  - Partitions/swimlanes: responsibility
  - Join synchronizes concurrent flows
  - State machine: object lifecycle
  - Event-triggered transitions
  - Optional guards/actions
- Diagram selection
  - Sequence: participant message order
  - State machine: event-dependent state changes
  - Activity: workflow
  - Sequence/state both temporal behavior
  - Significant-state objects: entity or control
  - DFD: data stores, flows, transformations
  - Inputs → outputs
  - DFD isn't UML

System design and subsystem decomposition | Exam focus

- System versus detail design
  - Analysis model → architecture

- Design goals, subsystem responsibilities/dependencies
- Hardware mapping, data management
- Access control, control flow
- Startup/shutdown/error policies
- Detail design: APIs, parameter/return types
- Exceptions, components, algorithms
- Subsystem: manageable logical collection
- Services and interface
- Not necessarily physical replaceable artifact

• Subsystem decomposition and architecture documentation
- Group related objects/functionality
- Minimize cross-boundary associations
- Isolate communication infrastructure where useful
- Workflows inform, don't dictate internals
- Analysis: domain needs
- Subsystem decomposition: principally design result
- Architecture documents: communication, major decisions
- Not every source execution step

Cohesion, coupling and refactoring

• Cohesion, coupling and hiding
- Cohesion: related purpose within module
- Coupling: dependencies between modules
- Seek high cohesion, low coupling
- Shared mutable data increases coupling
- Internal-representation knowledge increases coupling
- Stable operations hide changeable decisions

• Cohesion types and modularity trade-offs
- Logical: similar-category operations, often flag-selected
- Temporal: same-time operations
- Functional: one task
- Split unrelated three-purpose component
- Modularity improvement

- Behavior-preserving restructuring: refactoring
- Answers needn't be mutually exclusive
- Smaller modules: understanding, independent work
- Reuse and maintenance benefits
- Too many interfaces increase complexity

Layers, tiers and architecture views

• Layers versus tiers
- Layer: logical abstraction level
- Tier: deployment separation
- Closed: only immediately lower calls
- Open: approved layer skipping
- A,B,C,D top-to-bottom
- A→B closed-compliant, A→C skips
- Lower→upper violates simple dependency rule
- Explicitly defined mediation exception

• Structures, views and architectural concerns
- Structure: elements/relations themselves
- View: selected-aspect representation
- Module: code organization
- Component-and-connector: runtime interaction
- Allocation: software→hardware/teams/environment
- Concurrency views: processes/threads
- Quality attributes/scenarios evaluate architecture
- Processor allocation, decomposition, style selection

Architecture styles and integration

• Architecture styles and trade-offs
- Repository: shared central data
- Common access versus bottleneck/schema dependence
- MVC: model, view, controller
- Domain, display, interaction separation
- Client-server: requester/provider roles

- Peer-to-peer: both roles
- Pipe-and-filter: successive data-stream transformations
- Three-tier: presentation/application logic/storage
- Four-tier variant: split presentation client/server

• Integration boundaries and architecture diagrams
- API gateway: single entry
- Routes/combines service calls
- Hides service boundaries from clients
- Central policies versus bottleneck risk
- Component diagram: components/interfaces/dependencies
- Deployment diagram: artifacts on nodes
- Required interactions/qualities guide style
- No universally best architecture

Design principles and operation contracts | Exam focus

• Design principles
- Abstraction/decomposition manage complexity
- Consistency, traceability, clear responsibilities
- Top-down: high-level refinement
- Bottom-up: existing-component assembly
- Hybrid: combine both
- SOLID: Single Responsibility
- Open-Closed: compatible extensions
- Liskov substitution
- Interface Segregation: avoid unused operations
- Dependency Inversion: depend on abstractions
- DRY: limit duplicated knowledge
- KISS: avoid unnecessary complexity
- YAGNI: avoid speculative work

• Operation contracts and full features
- Preconditions, postconditions, invariants
- withdraw requires amount>0
- Also amount<=balance

- Ensures balance=oldBalance-amount
- Invariant: balance>=0
- Validation: input checks
- Implementation: enforce transition
- UI/API plus persisted behavior required
- Diagram alone insufficient

Analysis-to-implementation example | Exam focus

- Submit Incident implementation responsibilities
  - Boundary: input
  - Service: validation/coordination
  - Repository: save Incident entity
  - require authenticated
  - missing(location) → validationError
  - new Incident(input) → repository.save
  - Return incident.id
  - Save and ID assignment atomic
  - Failure requires rollback
- Model-derived tests and traceability
  - Missing location: saves nothing
  - Valid input: saved ID returned
  - Storage failure: error, no partial record
  - Unauthorized input rejected
  - Requirement → use-case step → service operation
  - Repository effect → test
  - Analysis/design made actionable
  - Implementation-practice gap closed

Patterns, variability and testability

- Design patterns
  - Factory Method: subclass-selectable creation operation
  - Reduced concrete-constructor coupling
  - Simple factory: centralized creation

- Not identical to Factory Method
- Singleton: controlled one-instance access
- Observer: notify subscribers
- Memento: store/restore state, representation hidden
- Iterator: traversal
- Decorator: behavior through wrapping
- Bank Momento means Memento

• Variability and testability
- Selectable features, omission/inclusion, extension points
- Overloading: alternate code signatures
- Not every variability mechanism
- Controllable inputs, observable outputs
- Replaceable dependencies, limited structural complexity
- Sandbox: execution isolation
- Self-referential A-and-B option B
- No unique answer established

Planning, WBS, estimation and tools

• Continuous planning and WBS
- Scope/deliverables identification
- Work decomposition
- Effort/duration/cost estimation
- Resources assigned, dependencies sequenced
- Revise using new evidence
- WBS: hierarchical project-scope coverage
- Exclude unrelated work
- Tools coordinate ownership/dependencies/progress
- Judgment not replaced
- No automatic bug repair

• Functional size and development tools
- Function points: inputs, outputs, inquiries
- Internal logical files
- External interface files

- Weights depend on chosen method
- Not merely lines of code
- Not exact direct complexity measure
- Git: version tracking
- Jira: work/defect tracking
- Jenkins: automated CI checks
- Docker: application/runtime dependency containers

Critical-path forward and backward passes

- Critical-path network calculations
  - ES=max(predecessor EF)
  - EF=ES+duration
  - Start ES=0 unless constrained
  - Earliest project finish=max(terminal EF)
  - Backward LF=min(successor LS)
  - LS=LF-duration
  - Backward pass starts required finish
  - Float=LS-ES=LF-EF
  - Critical path: longest total duration
  - Normally zero float, unconstrained schedule
- Parallel tasks and multiple critical paths
  - A=5; B=3,C=4 follow A
  - Duration=5+max(3,4)=9 days
  - B finishes day8, C day9
  - Project waits for C
  - Paths B5+C6 and D8+E3
  - Shared zero-duration start/end
  - Both 11 days: two critical paths

Slack assumptions and PERT

- Slack requires network context
  - ES/EF alone insufficient
  - Successor relationships required

- Project completion constraint required
- C3→D6 parallel A4→B5
- Both complete paths finish day9
- Then C float=0
- Four unrelated rows don't establish network

• CPM versus PERT

- CPM: point durations
- PERT: uncertain durations
- a optimistic, m most-likely, b pessimistic
- TE=(a+4m+b)/6
- Variance=((b-a)/6)^2
- a=3,m=6,b=15 days
- TE=42/6=7 days
- Variance=4 days²
- Weighted estimate, not simple average
- Approximation, not guaranteed completion

ROI, risks and resource optimization

• ROI and risk management

- Common-date discounted benefits/costs assumed
- ROI=(benefits-costs)/costs*100%
- Birr120000 benefits, Birr100000 costs
- ROI=20%, not 120%
- Project risks: resources/schedule
- Product risks: quality
- Business risks: organization
- Likelihood/impact analysis
- Plan responses and monitor

• Risk responses and resource optimization

- Avoidance: remove risk through plan change
- Mitigation: reduce likelihood/impact
- Transfer: contractual consequence assignment
- Retention/acceptance: keep risk

- Passive acceptance: possibly no action
- Leveling: dates adjusted for resource availability
- May alter project finish
- Smoothing: float within schedule constraints
- Neither creates extra staff

Teams, organizations and process maturity

- Organization, teams and development stages
  - Functional organization: discipline specialists
  - Project organization: project grouping
  - Specialization: staffing/knowledge continuity
  - Possible handoff barriers
  - Democratic team: shared decisions
  - Chief-programmer team: centralized technical coordination
  - Different classification dimensions
  - Tuckman: forming→storming→norming→performing→adjourning
  - Orientation → conflict → norms
  - Effective work → closure
- Historical Software CMM maturity
  - 1 Initial, 2 Repeatable
  - 3 Defined, 4 Managed
  - 5 Optimizing
  - Level2: basic project discipline
  - Requirements management and project planning
  - Quality assurance and configuration management
  - Historical labels ≠ modern CMMI terminology
  - Process capability focus
  - No defect-free-product guarantee

Configuration, change and version control | Exam focus

- Configuration management and controlled items
  - Items, versions/baselines
  - Change procedures, status, audits

- Source, requirements, manuals
- Test plans, environments, build configurations
- Ordinary team browser: usually tool
- Controlled test-browser version: possible environment CI

• Change workflow and distributed Git
- Request → impact assessment → decision/prioritization
- Implementation → validation
- Configuration management: consistent affected versions
- Each clone: local history
- Independent branches then merging
- Conflicts/approvals remain possible
- Plan test configuration management early
- Apply throughout testing
- No single exclusive phase

Quality, verification and validation | Exam focus

• Assurance, control, verification and validation
- QA: processes/standards established and checked
- QC: work-product evaluation
- SQA plan: standards/responsibilities/evaluations
- Reviews, audits, assurance records
- Conformance: follow design/specification
- Poor specification can still be followed
- Verification: correct specified-obligation implementation
- Validation: intended stakeholder needs met
- Both: reviews, analysis, execution
- Human-only/computer-only distinction false

• Defect, error, failure and evidence
- Fault/defect: underlying problem
- Error: possibly incorrect internal state
- Failure: externally observed service departure
- Reliability: product quality
- Low cost/short development: project outcomes

- Tests reveal defects, increase confidence
- No general all-input correctness guarantee

Testing levels and acceptance | Exam focus

- Unit, integration and system testing
  - Unit: function/class/small component isolation
  - Mocks/stubs replace dependencies
  - Driver invokes unit
  - Stub replaces called dependency
  - Integration: combined-part interfaces/interactions
  - System: complete integrated system
  - Functional/nonfunctional requirements
  - Representative environment
- Acceptance, alpha, beta and techniques
  - Acceptance: customer/user decision support
  - Agreed needs/criteria
  - Alpha: controlled developer setting, selected users
  - Beta: external users, own environments
  - No universally longest level
  - Unit: level; white-box: technique
  - Black-box unit cases possible

Test types, regression and exploration

- Test types and workload context
  - Functional: SRS/acceptance behavior
  - Performance: speed/throughput/resources under workload
  - Load: expected demand
  - Stress: beyond normal limits
  - Usability/security/compatibility/reliability: respective qualities
  - Normal conditions alone: insufficient classification
  - Functional versus performance still unresolved
- Techniques, regression and exploration
  - Black-box: specified behavior

- White-box: internal structure
- Grey-box: partial internal knowledge
- Regression: unintended change effects
- Confirmation: specific defect repair
- Exploratory: learning/design/execution combined
- Experience-based testing
- Manual/automated: execution support
- No compulsory black-/white-box classification

Equivalence, boundaries and decision tables

• Equivalence partitions and boundaries
  - Expected-similar behavior groups
  - Sample valid and invalid groups
  - Allowed age18..60
  - Partitions: below18,18..60,above60
  - Edges and neighbors
  - 17,18,19,59,60,61
  - Exact boundary set: stated coverage scheme
• Decision tables and state-transition cases
  - Conditions/combinations → actions
  - validCard, enoughFunds
  - TT,TF,FT,FF
  - Dispense only TT
  - Proper rejection otherwise
  - Cover feasible rules
  - Impossible combinations needn't be blindly covered
  - Specification-based techniques
  - Already-closed incident: reject dispatch

Statement, decision and branch coverage

• Structural coverage and limitations
  - Statement coverage=distinct exercised statements/total executable statements
  - Declared test scope determines denominator

- Unreachable statements can prevent 100%
- Document justified exclusions/adjusted scope
- Decision coverage: decision outcomes
- Branch coverage: control-flow edges
- Unconditional edges under stated convention
- Nested decisions may need >2 tests
- Exercised structure ≠ behavioral proof

• Nested-decision coverage example
- Outer true: inner true and false
- Outer false: inner true and false
- Four tests required
- (w,l,h)=(3,2,4),(3,2,1)
- Also (2,3,4),(2,3,1)
- One run can't cover both outer branches
- All paths may be infeasible
- Loops may create infinite paths
- State chosen coverage target

Test process, plans and defect reports

• Test process and monitoring
- Plan objectives/scope/resources/schedule/risks
- Entry/exit criteria
- Analyze basis: testable features, defects
- Design cases and expected results
- Implement data/scripts/environment
- Execute, compare, report defects
- Evaluate completion
- Analysis: what/testability
- Design: how
- Monitoring/control throughout

• Test plans and defect reports
- Plan: intended work
- Incident/defect reports: normally execution outputs

- Report build/environment
- Reproduction steps
- Expected versus actual behavior
- Evidence and severity
- Anyone finding fault should record
- Testers often coordinate, no exclusive permission
- Configuration control: reproducible tested version

Maintenance and software evolution

- Maintenance categories and cost
  - Corrective: fix faults
  - Adaptive: environment changes
  - Perfective: useful capability/performance improvement
  - Preventive: reduce future risk
  - Fragile-code restructuring example
  - Motive determines category
  - Refactoring: preventive or perfective support
  - Years of legacy-code understanding
  - Regression effort
  - Maintenance may dominate lifecycle cost
- Evolution workflow and restructuring
  - Request → impact analysis → release planning
  - Implementation/testing → release
  - Request absent: impact analysis earliest listed
  - Major restructuring: reengineering
  - Architecture changed: architectural transformation more specific
  - Markets/technology/stakeholders/funding change scope/priorities

Numerical and option defects

- Scheduling question defects
  - Earliest finish: 9 days
  - Correct answer absent
  - Slack: latest time − earliest time

- Zero slack requires supplied completion-path assumption
- Equal critical paths: 11 days
- Both corresponding options true
- Network rules before supplied key

• Multiple-answer and invalid-choice defects
- Architectural structures and views
- Evaluate each statement independently
- Reusable requirements, tests, and architecture
- Requirements/tests/architecture: reusable product-line assets
- Correct options or multiple-answer instruction needed
- Defective options don't justify false rules

Context-dependent claims and model limitations

• Missing context and classification limits
- No always-most-time-consuming test level
- Team inexperience doesn't determine process
- Waterfall doesn't make later changes impossible
- Requirements include required/prohibited behavior
- Interface functions versus usability qualities
- Exact interface obligation needed
- Functional/nonfunctional classification depends on obligation

• Model and process qualifications
- Sequence/state both dynamic behavior
- Composition: diamond placement plus ownership
- Safety-critical government Agile unsuitability claim
- Assumes lightweight process without required assurance
- Not universally true
- Guide conventions: scenario hints
- Stronger engineering rules explicit

Design trade-offs and economical improvement

• Goal-based design trade-offs
- Fine-grained self-contained components

- Producer/consumer separation
- Limited shared mutable data
- Often improve maintainability
- Interfaces may add latency
- Isolation may complicate transactions
- Global state: local simplicity, wider coupling
- Measurable quality scenarios
- Component counts alone insufficient

• Pareto and economical improvement
- 80/20: prioritization heuristic
- Few causes may dominate observed impact
- No exact 80%-defects/20%-effort guarantee
- Refactoring: behavior preserved, structure improved
- Optimize against measured needs
- Record rationale
- Update traceability/regression evidence after change

Design and Analysis of Algorithms

Revision Outlines

Problem specification and correctness | Exam focus

- Problem specification and algorithm properties
  - Inputs, output, constraints
  - Empty-solution policy
  - Finite, definite, executable methods
  - Source-language independence
  - Feasible: satisfies constraints
  - Optimal: best feasible
  - Conditional branches: legitimate control flow
- Correctness proof obligations
  - Invariant initialization
  - Invariant preservation
  - Invariant at termination
  - Recursion: base cases
  - Recursion: inductive reduction
  - Testing finds failures
  - Finite tests ≠ universal correctness proof

Cost models and input size | Exam focus

- Cost models and input size
  - Array length n
  - Graph vertices V, edges E
  - Integer bit length
  - RAM: bounded-word arithmetic/access constant
  - Arbitrary-precision arithmetic costs
  - Long-output printing costs
  - Elapsed time: hardware, compiler, cache
  - I/O and competing load effects
- Analysis methods and resource metrics

- Experimental/dynamic: actual runs
- Asymptotic: abstract growth
- Probabilistic: expectations
- No universal taxonomy exclusion
- Graph/dynamic analysis labels not inherently invalid
- Auxiliary space excludes input
- I/O complexity: transfers
- Representation affects costs
- Data type complexity: nonstandard metric name

Input cases versus bounds | Exam focus

• Input-case definitions
- Fixed input size n
- Best: minimum cost
- Worst: maximum cost
- Average: specified input distribution
- Randomized expectation: internal choices
- One fixed input possible
- Smallest input ≠ best case

• Bounds independent of cases
- O/Ω/Θ: function bounds
- Independent of case or resource
- Worst upper bound: all size-n inputs
- O ≠ worst case
- Constant best case not universal
- Quadratic worst case not universal

Formal asymptotic bounds | Exam focus

• Formal asymptotic relations
- Eventually nonnegative f,g
- O: $f(n) \leq c \cdot g(n)$, $n \geq n_0$
- Fixed $c>0$
- Ω: reversed inequality

- Θ: both bounds
- o: f/g→0
- ω: f/g→∞
- Ratio existence condition
- Θ: growth up to constant factors
- No exact-seconds/equal-case implication

• Asymptotic proof scope
- Whole tail inequality
- Selected values insufficient
- Transitive upper-bound chains
- Nonnegative sums: largest growth dominates

• Worked example Bound witnesses and incorrect inequality
- $10n+5 \leq 15n$, $n \geq 1$
- $c=15, n_0=1$
- $3n^2+4n+1 \leq 8n^2$, $n \geq 1$
- $7n^2+3n+5 \leq 8n^2$: false at $n=1$
- Correct witness: $c=15, n_0=1$

Growth ordering and loose bounds | Exam focus

• Representative growth ordering
- $1<\log n<n<n \log n$
- $n \log n<n^2<n^3<\ldots$
- Fixed $b>1$: polynomial$<b^n<n!$
- Log bases: constant-factor difference
- $n \log n, n^2$: superlinear
- $\log n$: sublinear
- Polynomial: fixed input-size exponent

• Loose bounds and finite-input speed
- $O(n^2)$ implies $O(n^3)$
- Upper bounds needn't be tight
- Multiple valid offered bounds
- Growth ordering alone: insufficient finite-speed prediction
- Big-O alone: insufficient finite-speed prediction

- Worked example Ratio comparison example
  - $f=4n^2$; $g=8n$
  - $f/g=n/2\to\infty$
  - $f\in\Omega(g)$
  - $f\in\omega(g)$
  - $f\notin O(g)$
  - $f\notin\Theta(g)$

Loop counts and additive costs | Exam focus

- Loop-count arithmetic
  - Sequential costs add
  - Independent nested counts multiply
  - Dependent loops require sums
  - Constant body × n: $\Theta(n)$
  - $\Sigma_{i=1}^{n} i=n(n+1)/2=\Theta(n^2)$
  - Doubling until $i\geq n$: $\lceil \log_2 n \rceil$
  - Doubling formula condition: $n>1$
  - 10n loop: linear
  - Constant-cost output assumption
- Expected total and exact counts
  - Fixed size: m operations
  - Each expected cost f(n)
  - Linearity: expected total=m f(n)
  - Independence unnecessary
  - Changing sizes: $\Sigma f(n_i)$
  - Tests/increments/arithmetic/assignments affect coefficients
- Worked example Exact loop-count convention
  - Two initializations
  - n+1 tests
  - n increments
  - 2n body operations
  - One return
  - Total: 4n+4

- Different primitives change exact count
- Growth remains $\Theta(n)$

Conditional loops and malformed code | Exam focus

- Conditional nested-loop summation
  - $i=0..n-1$; $j=i..i^2-1$
  - $k=0..j-1$ only when $j\%i==0$
  - $i=0,1$: empty j ranges
  - No remainder by zero evaluated
  - $i\geq 2$: $j=i,2i,\ldots,(i-1)i$
  - Inner work=$i\Sigma_{m=1}^{i-1}m$
  - $i^2(i-1)/2=\Theta(i^3)$
  - Sum over i: $\Theta(n^4)$
  - Condition checks: $\Theta(n^3)$
- Loop assumptions and malformed-code traps
  - Offered n^5,n^7: valid loose upper bounds
  - Neither bound tight
  - Mathematical integers
  - Constant-cost output
  - Signed C++ i*i overflow: undefined
  - Mixed-language syntax: repair before runtime claim
  - Parameter n redeclaration: syntax defect
  - return @;3: syntax defect
  - Syntax defects ≠ logarithmic algorithms

Peak storage and recursion | Exam focus

- Peak storage accounting
  - Simultaneously live storage
  - Cumulative allocations ≠ peak space
  - Requested input/auxiliary/stack convention
  - n active constant frames: $\Theta(n)$
  - log n constant frames: $\Theta(\log n)$
  - Total calls ≠ peak recursion depth

- Hidden storage and time-space tradeoffs
  - Array slicing/copying overhead
  - Index-based pseudocode may omit copies
  - Tail-call elimination: language/runtime-dependent
  - Ordinary C++/Java: no assumed elimination
  - Cached subproblem results save time
  - Caching consumes memory

Recurrences and substitution | Exam focus

- Recurrence construction and expansion
  - Every recursive call
  - Nonrecursive work
  - Base case
  - $T(n)=T(n-1)+\Theta(1)\rightarrow\Theta(n)$
  - $T(n)=T(n/2)+\Theta(1)\rightarrow\Theta(\log n)$
  - $T(n)=2T(n/2)+\Theta(n)$
  - Logarithmic levels × $\Theta(n)$
  - Result: $\Theta(n \log n)$
- Substitution proof method
  - Guess bound
  - Induction with constants
  - Base-case verification
  - Lower-order correction may close induction
  - Unequal/decrementing recurrences: separate analysis

Master theorem and recursion trees | Exam focus

- Master theorem and recursion tree
  - $T(n)=aT(n/b)+f(n)$
  - $a\geq1$; $b>1$
  - $p=\log_b a$
  - Level i: a^i nodes
  - Node size: n/b^i
  - Polynomially smaller f: $\Theta(n^p)$

- f=Θ(n^p): Θ(n^p log n)
- Polynomially larger f: Θ(f(n))
- Case 3 regularity: a f(n/b)≤c f(n)
- Fixed c<1

• Master theorem limitations
- Log-factor boundary: extension or summation
- State base work
- Include leaf costs
- T(n)=T(n−1)+n: outside this version
- Unequal-split quicksort: outside this version

• Worked example Master theorem worked cases
- $4T(n/2)+n \to \Theta(n^2)$, case 1
- $2T(n/2)+n \to \Theta(n \log n)$, case 2
- $T(n/2)+n^2 \to \Theta(n^2)$, case 3
- $4T(n/2)+n^2 \to \Theta(n^2 \log n)$, case 2

Comparison-sorting lower bounds | Exam focus

• Comparison-sort decision-tree lower bound
- Distinct-key orderings: n! leaves
- Height h: at most 2^h leaves
- $h \geq \log_2(n!) = \Omega(n \log n)$
- Mergesort matches worst-case bound
- Array indexing allowed

• Lower-bound model boundaries
- Counting/radix: structured keys
- Outside comparison-only model
- Recurrences: particular algorithm analysis
- Decision tree: entire comparison class
- Supplied inappropriate-technique justifications: all unsound

Choosing algorithm design strategies | Exam focus

• Design strategies from structural conditions
- Divide-and-conquer: smaller largely independent instances

- Combine subproblem results
- DP: reuse defined subproblem states
- Greedy: feasible safe local commitment
- Backtracking: grow/retract when constraints fail
- Branch-and-bound: best possible objective test
- Prune when incumbent cannot be beaten

• Ambiguous strategy descriptions
- Subproblems/choices/incremental/optimization: several strategies
- Exponential runtime insufficient for method choice
- Specify overlap
- Specify local-choice safety
- Specify revisability and exactness
- Specify usable bounds
- Module selection-sort classification: erroneous
- Ordinary selection: repeatedly choose one minimum

Binary search and paired extrema | Exam focus

• Binary-search invariant and costs
- Sorted random-access data
- Inclusive low/high bounds
- mid=low+(high−low)//2
- high=mid−1 or low=mid+1
- Stop: found or low>high
- $T(n)=T(n/2)+\Theta(1)$
- Worst: $\Theta(\log n)$; best: $\Theta(1)$
- Iterative auxiliary space: $\Theta(1)$
- Recursive auxiliary space: $\Theta(\log n)$

• Simultaneous minimum/maximum comparisons
- Two unconditional comparisons per element
- Scan comparisons: $2n-2$, every order
- Pair elements: compare within pair
- Smaller→minimum; larger→maximum
- Even n: $3n/2-2$ comparisons

- Balanced recursion: $2T(n/2)+\Theta(1)=\Theta(n)$
- Two-if scan: no $n-1$ best case

Merge sort | Exam focus

- Merge-sort invariant and procedure
  - Split by indices
  - Recursively sort halves
  - Temporary result: smaller remaining front
  - Output prefix sorted
  - Prefix contains smallest consumed elements
  - Append unexhausted suffix
  - Stable ties: left half first
- Ordinary array merge-sort analysis
  - $T(n)=2T(n/2)+\Theta(n)=\Theta(n \log n)$
  - All input orders
  - Auxiliary buffer: $\Theta(n)$
  - Recursion depth: $\Theta(\log n)$
  - $+O(n)$ alone: upper bound only
  - Tightness requires linear merge work
  - Base cases: empty/singleton subarrays
- Worked example Merge-sort worked trace
  - $[4,1,3,2] \rightarrow [4,1]$ and $[3,2]$
  - Sorted halves: $[1,4]$ and $[2,3]$
  - Select 1,2,3; append 4
  - Original sortedness unnecessary

Quicksort variants and costs | Exam focus

- Quicksort partition variants
  - Choose pivot, partition, recurse
  - Two pointers: advance i below pivot
  - Retreat j above pivot
  - $i \leq j$: swap, advance both
  - Bound scans

- Equal values: ensure progress
- Recurse on resulting ranges
- Hoare boundaries: pivot not necessarily final
- Lomuto: pivot commonly final
- Keep return conventions distinct

• Quicksort costs and refinements
- Balanced splits: $\Theta(n \log n)$
- Highly unbalanced splits: $\Theta(n^2)$
- Random pivots: expected $\Theta(n \log n)$
- Expectation ≠ deterministic guarantee
- Median-of-three retains adversarial cases
- Three-way partition: repeated keys
- Standard quicksort: unstable
- Expected stack $\Theta(\log n)$; worst $\Theta(n)$
- Recurse smaller side, iterate larger
- Refined stack bound: $O(\log n)$

Sorting choices and costs

• Elementary sorting choices
- Insertion: larger sorted-prefix keys shift
- Insertion: strict comparisons preserve stability
- Optimized bubble: adjacent inversion swaps
- Bubble stop: no-swap pass
- Insertion/bubble: $\Theta(n)$ best, $\Theta(n^2)$ worst
- Selection: repeated suffix minimum
- Selection: $\Theta(n^2)$ comparisons every order
- Selection: normally unstable
- All three: $\Theta(1)$ auxiliary array space

• Heap, merge, quicksort tradeoffs
- Max-heap construction: $\Theta(n)$
- Extract/restore: $O(\log n)$ per item
- Heapsort total: $O(n \log n)$
- Heapsort: $\Theta(1)$ auxiliary space

- Heapsort: ordinarily unstable
- Heapsort ≠ standard divide-and-conquer
- Mergesort: stability/worst-case guarantees
- Quicksort practical advantage: implementation/workload-dependent

Strassen matrix multiplication | Exam focus

- Matrix multiplication divide/combine costs
  - Ordinary n×n: n products/sums per entry
  - Ordinary multiplication: $\Theta(n^3)$
  - Eight products: $8T(n/2)+\Theta(n^2)=\Theta(n^3)$
  - Strassen: seven block products
  - Block sums and differences
  - $7T(n/2)+\Theta(n^2)=\Theta(n^{\log_2 7})$
  - Strassen exponent: approximately 2.807
- Strassen assumptions and tradeoffs
  - Zero-padding preserves required product
  - Smaller exponent versus extra additions
  - Temporary storage overhead
  - Possible numerical issues
  - Ordinary base case: small matrices
  - Specify exact versus floating-point arithmetic

Maximum contiguous subarray | Exam focus

- Maximum subarray by divide-and-conquer
  - Contiguous subarray ≠ arbitrary subsequence
  - Compare best left/right/crossing
  - Crossing: left suffix+right prefix
  - Best crossing found in $\Theta(n)$
  - $2T(n/2)+\Theta(n)=\Theta(n \log n)$
- Kadane dynamic programming
  - $end[i]=\max(A[i],end[i-1]+A[i])$
  - Keep largest ending sum
  - Time: $\Theta(n)$

- Extra space: Θ(1)
- Nonempty initialization: A[0], not zero
- All-negative result: largest element
- Empty allowed: zero possible
- Unqualified strategy question: D&C and DP

• Worked example Maximum-subarray examples
- [−2,3,−1,4,−5] ending: −2,3,2,6,1
- Optimum 6: [3,−1,4]
- [−4,−2]: nonempty answer −2

Greedy proofs and counterexamples | Exam focus

• Greedy safety and exchange proofs
- Locally attractive feasible choice
- Commit without later undoing
- Optimal substructure alone insufficient
- Exchange greedy choice into optimum
- No worsening of cost
- Recurse after exchange
- MST cut proof: exchange argument

• Greedy limits and valid applications
- Simple decisions, low overhead
- Irrevocable errors
- Problem-specific applicability
- Greedy heuristic ≠ approximation guarantee
- Coins {1,3,4}; target 6
- Largest-first: 4+1+1; optimum: 3+3
- Activity selection: earliest finish
- Maximize count: nonoverlapping unweighted intervals
- Exchange first activity for earlier finish
- At least as much remaining room

Fractional versus indivisible knapsack | Exam focus

• Fractional knapsack greedy rule

- Positive weights/profits
- Divisible items
- Decreasing value/weight order
- Whole fitting items, then fitting fraction
- Exchange lower-density for higher-density weight
- Value cannot decrease
- Sorting O(n log n); scan linear
- Zero-weight beneficial items: handle separately
- Avoid division by zero

• Indivisibility breaks ratio proof
- Capacity 10
- Items (6,12),(5,8),(5,8)
- Ratio-greedy value: 12
- 0/1 optimum: 8+8=16
- Specify fractional/0/1/unbounded knapsack
- Feasibility ≠ optimality

• Worked example Fractional knapsack worked trace
- Weights 15,10,18; values 24,15,25
- Capacity 20
- Take weight 15 item
- Take half weight 10 item
- Value=24+7.5=31.5
- Initial capacity 20 ≠ item count

Unit-time job sequencing | Exam focus

• Unit-job scheduling assumptions and rule
- One machine
- Initially available jobs
- Unit durations; integer deadlines
- Profit only for on-time completion
- Nonnegative-profit jobs: decreasing profit
- Latest free slot finishing by deadline
- No feasible slot: reject

- Latest placement preserves tighter-deadline slots
- Profit-first exchange for feasible unit-job sets

• Job sequencing complexity and limitations
- Sorting plus slot scans
- O(n log n+nD)
- Cap D at n
- Equal-profit ordering arbitrary
- Earlier deadline tie rule optional
- Variable durations invalidate rule
- Release times invalidate rule

• Worked example Job sequencing worked certificate
- Slots 1…5: J_2,J_4,J_3,J_5,J_1
- Profits: 180+300+190+120+200=990
- All deadlines hold
- J_6 cannot finish by 2
- Five largest profits: 990 upper bound
- Matching bound certifies optimum

Optimal merging and Huffman coding | Exam focus

• Optimal merge pattern
- Combine two smallest weights
- Add sum to total
- Reinsert sum
- Min-heap: O(n log n) time
- Space: O(n)
- p+q: moved-record cost
- Nonempty lists: p+q−1 worst comparisons
- Actual comparisons may be fewer

• Huffman prefix coding and proof
- Frequencies as weights
- 0/1 branches to leaf symbols
- Prefix-free code
- Cost=Σfrequency×depth

- Two least frequencies: deepest siblings
- Exchange, contract, induct
- Ties change codes, preserve optimum cost
- Header overhead may outweigh tiny-file savings

• Worked example Merge/Huffman worked weights
- Weights 2,3,5,6
- 2+3=5
- 5+5=10
- 10+6=16
- Total weighted merge cost: 31
- Same construction: minimum weighted prefix-code length

MST algorithms and cut proofs | Exam focus

• MST properties and uniqueness
- Connected weighted undirected graph
- Spans vertices without cycles
- Minimum total edge weight
- V−1 edges
- Negative edges allowed
- Equal weights: possible multiple MSTs
- Distinct weights suffice for uniqueness
- A tree is a graph

• Prim, Kruskal, cut-exchange proof
- Prim: cheapest tree-to-outside crossing edge
- Kruskal: globally sorted edges
- Union-find: accept different-component endpoints
- Light cut edge exchange into optimum
- No increased total weight
- Matrix Prim: $O(V^2)$
- Binary-heap Prim: O(E log V)
- Kruskal: O(E log E)
- Dense-graph comparison: implementation-dependent
- Disconnected input: minimum spanning forest

- Worked example MST triangle examples
  - AB=1,BC=2,AC=4
  - Kruskal accepts AB,BC
  - Total cost: 3
  - All edges 1: any two optimal
  - Equal-weight example: uniqueness fails

Dijkstra shortest paths | Exam focus

- Dijkstra initialization and relaxation
  - dist[source]=0; others ∞
  - Predecessors initially unset
  - Extract minimum-distance unsettled u
  - dist[v]=min(dist[v],dist[u]+w(u,v))
  - Improvement: update predecessor
  - Minimum ∞: remaining vertices unreachable
  - Stop after destination finalized
  - Discovery alone insufficient for stopping
- Dijkstra proof, failure, costs
  - Nonnegative edges preserve finalized distances
  - Later unsettled paths cannot improve finalized
  - s→a=2; s→b=5; b→a=−4
  - Finalize a=2: miss later distance 1
  - Binary-heap/list: O((V+E)log V)
  - Matrix scanning: $O(V^2)$
  - MST: total connection cost
  - Dijkstra: each source-path distance

DP states and correctness | Exam focus

- Dynamic-programming specification and proof
  - State meaning and choices
  - Recurrence and bases
  - Dependency order
  - Better subsolution must improve full solution

- Failed replacement: missing state information
- Cover all legal final choices
- Generate feasible candidates
- Induct in dependency order
- Store reconstruction choices

• DP signatures and complexity
- Overlapping states motivate reuse
- Optimal substructure supports optimization
- DP also counts and decides
- Optimization not universally required
- Multiple users per state not required
- Cost=states×work per state
- DP may remain exponential

Memoization, tabulation, Fibonacci | Exam focus

• Memoization versus tabulation
- Memoization: recursive cache check
- Store solved state results
- Tabulation: explicit dependency order
- Chosen domain may exclude irrelevant states
- Cache keys: all relevant parameters
- Recursion/locality/sparsity affect cost

• Fibonacci reuse and arithmetic assumptions
- Naive branches recompute F(n−2)
- Naive call count: Θ(φ^n)
- O(2^n): valid loose bound
- F(0)=0; F(1)=1
- F(i)=F(i−1)+F(i−2)
- Memo/tabulation: Θ(n) unit-cost additions
- Two recent values: Θ(1) machine words
- Arbitrary precision: growing bit storage
- Naive overlap ≠ independent D&C ideal

DAG paths and Bellman-Ford | Exam focus

- DAG/multistage shortest-path DP
  - Reverse topological cost-to-go
  - F(t)=0
  - $F(u)=\min_{(u,v)}[w(u,v)+F(v)]$
  - Missing routes: ∞
  - Record minimizing next vertex
  - Alternative: forward source-distance propagation
  - Adjacency-list complexity: Θ(V+E)
  - Negative edges harmless without cycles
  - Module forward/backward labels inconsistent
  - Follow recurrence dependencies
- Bellman-Ford and strategy ambiguity
  - General graphs
  - Relax all edges: up to V−1 passes
  - Additional pass: reachable negative-cycle test
  - Complexity: O(VE)
  - Single-source shortest path ≠ uniquely greedy
  - Graph assumptions select algorithm
- Worked example DAG cost-to-go example
  - s→a=2,s→b=5,a→t=4,b→t=−1
  - F(a)=4; F(b)=−1
  - F(s)=min(6,4)=4
  - Optimal route via b

Floyd-Warshall and reachability | Exam focus

- Floyd-Warshall initialization and invariant
  - All-pairs distances
  - D[i,i]=0
  - D[i,j]: minimum direct-edge weight
  - Missing edges: ∞
  - Negative self-loop: min(0,weight)
  - k outermost

- D[i,j]=min(D[i,j],D[i,k]+D[k,j])
- Before phase k: earlier intermediates only
- Guard unreachable sums
- Guard numeric overflow

• Floyd-Warshall scope and reachability variant
- Time: $\Theta(V^3)$
- Storage: $\Theta(V^2)$
- Negative diagonal: negative cycle
- Pairs traversing cycle: no finite minimum
- Warshall reachability: OR/AND replace min/+
- Floyd cost ≠ universal APSP optimum
- Sparse nonnegative graph: repeated Dijkstra
- Other models permit other methods

• Worked example Floyd-Warshall improvement retention
- D[a,b]=4; D[b,c]=2
- Initial D[a,c]=11
- Phase b: D[a,c]→6
- Later phases retain improvement
- Module reprints 11: incorrect

0/1 knapsack and dominance | Exam focus

• 0/1 knapsack state and recurrence
- Integer capacity W
- K[i,w]: first-i maximum within capacity w
- Base: K[0,w]=0
- weight_i>w: K[i,w]=K[i−1,w]
- Otherwise max(K[i−1,w],value_i+K[i−1,w−weight_i])
- Previous row: each item at most once
- Backtrack changed values: recover items

• Knapsack complexity and dominance
- Time O(nW): pseudo-polynomial
- Binary W encoding: O(log W) bits
- Storage: O(W)

- 0/1 updates: capacities downward
- Upward updates: item reuse, unbounded variant
- Sparse states: (weight,profit)
- Dominating state: no greater weight
- Dominating state: no smaller profit
- Same remaining items required

• Worked example Knapsack optimum and dominance example
- Weights 2,3,4; values 1,2,5
- Capacity W=6
- Choose weights 2+4
- Optimal value: 6
- (weight 5,profit 3) dominated
- Dominating state: (weight 4,profit 5)

LCS and reconstruction | Exam focus

• LCS prefix-state recurrence
- Subsequence: order preserved, gaps allowed
- L[i,j]: prefix lengths i,j
- Empty-prefix rows/columns: 0
- Compare X[i−1] with Y[j−1]
- Match: L[i,j]=1+L[i−1,j−1]
- Mismatch: max(L[i−1,j],L[i,j−1])
- Short-note correction: prefix lengths versus indices

• LCS evaluation and reconstruction
- Increasing rows/columns
- Matching characters: diagonal reconstruction
- Otherwise follow equal-optimum neighbour
- Ties: different valid LCSs
- Time: Θ(mn)
- Full table: Θ(mn) storage
- Value-only two rows: O(min(m,n))
- Reconstruction: retained decisions or another technique

• Worked example LCS versus longest common substring

- X=ABC; Y=AC
- LCS=AC; length=2
- Substring matching: different recurrence
- Substring mismatch resets to zero

Matrix-chain and optimal-BST interval DP | Exam focus

• Matrix-chain interval DP
- A_i dimensions: p[i−1]×p[i]
- Base: M[i,i]=0
- M[i,j]=min_{i≤k<j}(M[i,k]+M[k+1,j]+p[i−1]p[k]p[j])
- Increasing interval length
- Store optimal split
- 10×30,30×5,5×60
- (AB)C: 4500; A(BC): 27000
- Time $O(n^3)$; space $O(n^2)$
- Fixed multiplication order; variable parenthesization

• Optimal-BST interval DP
- Sorted-key successes p_i; gap failures q_i
- W(i,j)=Σ_{k=i}^j p_k+Σ_{k=i−1}^j q_k
- Comparison-count base: C(i,i−1)=0
- C(i,j)=W(i,j)+min_r(C(i,r−1)+C(r+1,j))
- Store optimal root r
- CLRS dummy-depth empty cost: q_{i-1}
- Final-value difference: fixed total gap weight
- Basic time $O(n^3)$; space $O(n^2)$

Reliability design under budget | Exam focus

• Reliability redundancy model
- Independent device failures
- m_i parallel copies; device reliability r_i
- Stage $\varphi_i(m_i)=1-(1-r_i)^{m_i}$
- Series total: $\prod\varphi_i$
- Integer copies $m_i \geq 1$

- Budget: $\Sigma m_i c_i \leq B$
- Extra copies: reliability versus budget
- Correlated failures invalidate independence formula

• Reliability budget DP
- R[i,b]: first-i best within budget b
- Base: R[0,b]=1
- R[i,b]=max_m R[i−1,b−mc$_i$]φ$_i$(m)
- Feasible m≥1
- Empty candidate set: 0
- Time O(stages×budget×maximum copies)
- Value storage: O(budget)
- Retain copy counts for reconstruction

• Worked example Reliability allocation comparison
- Costs 30,15,20; reliabilities 0.9,0.8,0.5
- Budget 105
- Copies (1,2,2): cost 100
- Reliability: 0.9×0.96×0.75=0.648
- Copies (1,1,3): cost 105
- Reliability only 0.63

Exact TSP subset DP | Exam focus

• Exact TSP subset-state recurrence
- Minimum tour: every vertex once, return
- Shortest path needn't cover/return
- Fix start s
- F(S,j): minimum from s through S
- Exactly S; ending j∈S
- Base: F({s},s)=0
- F(S,j)=min_{i∈S−{j}}[F(S−{j},i)+w(i,j)]
- Finish: min_{j≠s}[F(V,j)+w(j,s)]
- Missing edges: ∞

• Held-Karp costs and alternatives
- Store predecessors for tour reconstruction

- Time: $O(V^2 2^V)$
- Space: O(V 2^V)
- Improves factorial enumeration
- Still exponential
- Nearest-neighbour: heuristic, no general exactness
- Backtracking/branch-and-bound: exact TSP alternatives
- Generic method selection needs details

• Held-Karp boundary conditions
- S must contain start s
- Initialize F({s},s)=0
- Impossible states: ∞
- Larger sets: endpoints j≠s
- Avoid nonexistent predecessor states

BFS, DFS, graph representations

• BFS versus DFS discovery
- BFS: mark on enqueue
- FIFO neighbour exploration
- Record predecessor and level
- Levels: minimum edge counts
- Unweighted/equal-weight shortest paths
- DFS: mark on entry
- Recursive or iterative branch completion
- Unvisited neighbour before parent resumption
- Resume parent after branch

• Graph traversal conditions and costs
- Disconnected: restart each unvisited vertex
- Adjacency-list time: O(V+E)
- Auxiliary visited/queue/stack: O(V)
- Matrix scanning: $O(V^2)$
- Exact order requires neighbour order
- Reverse DFS finish order: DAG-only topological
- Printed under DP ≠ DP optimization

Articulation, bridges, biconnected components

- Articulation discovery and low values
  - Removal increases connected-component count
  - Undirected DFS
  - disc[u]: discovery time
  - low[u]: smallest reachable discovery number
  - Reachability from u's subtree
  - Downward tree edges; at most one back-edge
  - Update from children/nonparent back edges
  - Root articulation: ≥2 DFS children
  - Nonroot articulation: some low[v]≥disc[u]
- Articulation, bridges, biconnected blocks
  - Child cannot bypass u to ancestor
  - Bridge: low[v]>disc[u]
  - Block trigger: low[v]≥disc[u]
  - Edge stack: pop through (u,v)
  - Adjacency-list time: O(V+E)
  - Vertex-biconnectivity ≠ directed strong connectivity
- Biconnected-component DFS procedure
  - Simple undirected graph
  - On entry: disc[u]=low[u]=++time
  - Unvisited v: push (u,v), DFS(v)
  - Then low[u]=min(low[u],low[v])
  - Visited nonparent ancestor: disc[v]<disc[u]
  - Push back edge once
  - Back-edge update: low[u]=min(low[u],disc[v])
  - low[v]≥disc[u]: pop through (u,v)
  - Separate root-child articulation rule
  - Bridge test: low[v]>disc[u]
  - Restart each unvisited vertex
  - Multigraph: skip parent edge by identity
  - Do not skip every parent-directed edge
- Worked example Two-triangle low-link check

- 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 articulation: C
- No bridges
- Blocks: {A-B,B-C,C-A}; {C-D,D-E,E-C}
- low[D]=disc[C]
- Articulation ≥ versus bridge >

Backtracking and safe pruning | Exam focus

- Backtracking candidate template
  - Partial assignment x[1..k]
  - For each allowed choice: assign
  - Test possible extension
  - Promising: recurse
  - Return: undo assignment
  - Report complete feasible candidates
  - Stop after one: existence/single-answer request
  - Explicit constraints: variable domains
  - Implicit constraints: relations among choices
- Backtracking completeness and pruning
  - Consider every feasible extension
  - Prune only impossible branches
  - Disliked partial objective insufficient
  - DFS space proportional to depth
  - Total search may remain exponential
  - Equivalent-state memoization reduces work
  - Stronger constraints reduce work
  - Branch count: actual choices/tests

N-queens | Exam focus

- N-queens representation and conflict tests

- x[r]=column; one queen per row
- Earlier row i: conflict comparison
- Reject column conflict: x[r]=x[i]
- Reject diagonal: $|x[r]-x[i]|=|r-i|$
- Distinct rows built into representation
- Legal placement: recurse, then remove
- Column and r−column/r+column sets
- Set-based conflict tests: constant time

• N-queens bounds and conventions
- Distinct columns: at most n! permutations
- Additional partial-node work
- Pruning ≠ polynomial-time guarantee
- Placement/conflict storage: O(n)
- n=2,3: no solutions
- State empty-board convention

• Worked example N-queens placement examples
- n=4: (2,4,1,3) valid
- Rows 1,2; columns 2,3
- Row difference=column difference=1
- Diagonal conflict: reject partial branch

Subset sum | Exam focus

• Subset-sum include/exclude pruning
- Track current sum s
- Track remaining sum R
- Nonnegative weights assumption
- Prune s>target
- Prune s+R<target
- Equality: report subset
- All subsets: consider zero-weight extensions
- Sorted positives strengthen tests
- Sorting cannot force next-item inclusion

• Subset-sum boundary assumptions and bounds

- Negative values: overshoot may recover
- Use valid attainable-range bounds
- At most 2^n subsets
- Recursion depth: O(n)
- Duplicate values: define index-distinct counting

• Worked example Subset-sum feasibility bounds example
- {3,4,5,6}; target 13
- Solution: {3,4,6}
- s=9,R=3: 12<13, failure
- s=9,R=4: equality still possible

Graph coloring | Exam focus

• Fixed-m graph coloring search
- Assign colours 1…m
- Reject already-coloured neighbour's colour
- All colours fail: backtrack
- At most m^V complete assignments
- Additional conflict-checking cost
- Simple test: O(V)
- Exponential bound
- Module polynomial-looking expression: garbled
- Self-loop: proper coloring impossible

• Coloring decision versus optimization
- Decision: do m colours suffice?
- Chromatic number: minimum colours
- Increasing m with complete decision search
- Greedy: vertex-order-dependent colour choice
- Greedy may use too many
- No general greedy optimum proof
- Odd cycle: 3 colours
- Edge-containing bipartite graph: 2 colours
- Edgeless nonempty graph: 1 colour

Hamiltonian cycles | Exam focus

- Hamiltonian cycle backtracking
  - Fix start: eliminate rotational duplicates
  - Extend to unused adjacent vertex
  - At V vertices: require return edge
  - Missing return edge: backtrack
  - Hamiltonian: visit vertices once
  - Euler: visit edges
  - Fixed-start candidates: up to (V−1)!
  - Polynomial checking overhead
- Hamiltonian necessary conditions and TSP
  - Simple undirected graph; V≥3
  - Degree 1 rules out Hamiltonian cycle
  - Articulation point rules out Hamiltonian cycle
  - Absence of obstructions insufficient
  - TSP: optimize weighted Hamiltonian tour
  - Existence search alone doesn't minimize cost
  - Undirected reversal equivalence for unique enumeration

Branch-and-bound states and bounds | Exam focus

- Branch-and-bound node lifecycle and selection
  - Live: generated, eligible for expansion
  - E-node: currently expanded
  - Dead: finished or pruned
  - FIFO: queue order
  - LIFO: stack order
  - Least-cost/best-bound: priority queue
  - Selection affects work/storage
  - Valid bounds preserve exactness
- Branch-and-bound bound directions
  - Minimization incumbent: feasible upper bound U
  - Partial lower bound L: every completion
  - One optimum/value: prune L≥U

- Maximization incumbent: feasible lower bound
- Optimistic completion: upper bound
- Maximization prune: upper≤incumbent
- All optima: retain equality branches

Knapsack branch and bound | Exam focus

- Knapsack fractional-relaxation bound
  - Positive weights/values
  - Sort decreasing density
  - Branch include/exclude
  - Reject overweight nodes
  - Fill remaining capacity fractionally
  - Fractional fill: remaining items only
  - Relaxed profit: upper bound
  - Whole-item completion updates incumbent
  - Fractional result ≠ legal 0/1 answer
- Knapsack branch-and-bound certificate
  - Capacity 10; (6,12),(5,8),(5,8)
  - Root bound: 12+(4/5)8=18.4
  - First included: whole-item value 12
  - First excluded: other two value 16
  - All live upper bounds≤16: optimality
  - Avoid downward floating-point bound rounding
  - Worst-case search: exponential

TSP matrix-reduction branch and bound | Exam focus

- Directed-TSP matrix-reduction bound
  - Diagonal/forbidden entries: ∞
  - Subtract finite row minima
  - Then subtract finite column minima
  - Accumulate reductions
  - Tour: one entry per row/column
  - Reduction sum: lower bound

- Relaxed assignment may contain disjoint subtours

• TSP include-edge reduction and search
  - Include i→j: remove row i/column j
  - Prevent premature subtours
  - Reduce residual matrix
  - Child bound=parent+selected reduced cost+new reductions
  - Compare with best complete tour
  - Expand promising live node
  - Matrix zeros ≠ valid complete tour
• TSP branching and subtour prevention
  - Branch on undecided i→j
  - Include: delete row i/column j
  - Include bound: parent+reduced edge+new reductions
  - Exclude: set entry ∞
  - Exclude bound: parent+new reductions
  - Forced in/out-degree at most one
  - Track forced path head h, tail t
  - Forbid t→h before all vertices included
  - Residual required row/column without choice: infeasible
  - Prune infeasible child
  - Accept one complete all-vertex cycle
  - Reject disjoint cycles
• Worked example TSP subtour procedure check
  - Forced 1→2→1 and 3→4→3
  - One incoming/outgoing each: insufficient
  - Two disjoint 2-cycles ≠ tour
  - Forced 1→2: forbid 2→1
  - Retain prohibition while path size<4
  - Existing three-city bound=25
  - Tour cost=25

P, NP, reductions, encoding | Exam focus

• P and NP definitions

- Decision problems
- P: deterministic polynomial input-bit time
- NP: polynomial-length certificates
- Polynomial-time verification
- Equivalent nondeterministic polynomial-time machine
- NP ≠ non-polynomial
- P⊆NP
- P=NP remains open

• Reductions, hardness, encoding size
- $A \leq_p B$: polynomial many-one instance transformation
- Preserve yes/no answers
- Prove B hard: known-hard A→B
- Reverse direction insufficient
- NP-complete: NP-hard and in NP
- Optimization TSP: NP-hard
- Bounded-tour decision TSP: NP-complete
- Tour/cost verification versus global-optimality certification
- O(nW): pseudo-polynomial knapsack
- Not polynomial in W's bit length

Damaged items and required clarifications

• Conditional-output damaged item
- p=10,n=5
- if(n>p) false: output skipped
- Execution terminates
- No implied feasibility/finiteness/control-flow violation
- Efficiency needs specification
- Required output needs specification

• Missing notation and syntax clarifications
- 3n?+7n+12: missing exponent
- Precise degree undetermined
- Fixed nonnegative integer exponents: polynomial
- Stray ini/inl: repair syntax

- Redeclared n: repair syntax
- Exact counts need primitive-cost model
- Marked key cannot replace missing clarification

Introduction to Artificial Intelligence

Revision Outlines

AI capabilities and scope | Exam focus

- Artificial intelligence scope
  - Reasoning; learning; planning; perception; language
  - Applied AI: useful solutions
  - Cognitive AI: biological intelligence models
  - Human behavior and experimental validation
- Narrow, general and strong AI
  - Weak AI: bounded tasks
  - Face recognition; spam; voice assistants
  - General AI: transferable domain competence
  - Strong AI: actual understanding
  - Humanoid appearance ≠ general intelligence
  - Autonomous vehicles ≠ general intelligence
  - Feature count ≠ general intelligence
- Exam trap Data analysis-AI overlap
  - Learning or inference criterion
  - Categorical exclusion trap

Four approaches and the Turing test | Exam focus

- Four AI approaches
  - Thinking humanly: mental processes
  - Thinking rationally: valid logic
  - Acting humanly: humanlike behavior
  - Acting rationally: expected objective achievement
  - Thought versus behavior evaluation
- Conversational Turing test
  - Judge: human-machine dialogue distinction
  - Language; knowledge; reasoning; learning
  - Operational behavior criterion

- Consciousness-proof trap
- Cognitive models: response times and errors
- Psychological experiments

• Worked example Approach identification
- Best action sequence: acting rationally
- Human-error modeling: thinking humanly

Agents, percepts and action | Exam focus

• Agent interaction
- Sensors perceive; actuators/effectors act
- Cameras versus motors
- Software messages versus API requests
- Human agents versus artificial agents

• Agent function and architecture
- Percept: single observation
- Percept sequence: entire observation history
- f:P*→A
- Agent = architecture + program
- Program implements function
- Environment beyond search space
- Autonomy: experience over designer instructions

Rationality and performance | Exam focus

• Rationality and success measures
- External performance: safety; deliveries; time; cost
- Prior knowledge and percept history
- Available actions
- Best expected performance given information

• Rationality versus omniscience
- Omniscience: advance outcome knowledge
- Rationality without guaranteed success
- All actions fail boundary case
- Rational information gathering and exploration

- Worked example Expected versus realized route time
  - Rational expectation: 20 minutes
  - Unforeseen closure: 40 minutes

Specify the task with PEAS | Exam focus

- PEAS task specification
  - Performance measure; Environment; Actuators; Sensors
  - Specification before agent design
  - Taxi performance: safety and travel time
  - Taxi environment: roads; traffic; passengers
  - Taxi actuators: steering; brakes; displays
  - Taxi sensors: cameras; positioning; speed
- PEAS category traps
  - Sensor influence ≠ actuation
  - Performance measure: success criterion
  - Percept sequence: observation evidence
  - Agent observations versus designer knowledge

Simple and model-based reflex agents

- Simple reflex agent
  - Current-percept condition-action rules
  - Dirty → clean
  - No persistent world-state memory
  - Sufficient current information assumption
  - Partial observation: identical percept, different actions
- Model-based reflex agent
  - Previous state and previous action
  - Current percept
  - World-change and action-effect models
  - Internal-state condition-action rules
  - Shared rule use
  - Distinguishing memory and world model
- Worked example Hidden cleaned-room memory

- Internal state beyond sensor range
- Current dirt detection: no memory requirement

Goals, utility and model updates

- Goals versus utility
  - Goal: desired-state action sequences
  - Utility: desirability and trade-offs
  - Maximum expected utility
  - EU(a)=Σs P(s|a,evidence)U(s)
  - Destination goal
  - Arrival time; safety; cost preferences
- Internal model versus external action
  - Post-action state prediction
  - State update ≠ environment modification
  - Unused goal variable trap
  - BEST_ACTION dependence on goal
  - Coexisting model, goal, utility, learning capabilities

Learning-agent components | Exam focus

- Learning-agent components
  - Performance element: action selection
  - Learning element: future improvement
  - Critic: standard-based evaluation and feedback
  - Problem generator: useful exploratory experience
- Learning: task T, performance P, experience E
  - Spam T: email classification
  - Spam E: labeled messages
  - Spam P: recall at required precision
  - Exploration: information versus immediate reward
  - Exploitation: currently best-known action
  - Learning across reflex, goal, utility agents

Environment dimensions | Exam focus

- Observation and transition dimensions
  - Fully observable: relevant state revealed
  - Partially observable: hidden or noisy information
  - Deterministic: state/action determine successor
  - Stochastic: random outcomes
  - Strategic opponent uncertainty versus randomness
- Environment dimension contrasts
  - Episodic independence versus sequential effects
  - Static deliberation versus dynamic change
  - Semidynamic: clock changes score
  - Discrete alternatives versus continuous ranges
  - Known/unknown: transition-model knowledge
  - Model knowledge ≠ observability
- Worked example Environment classification examples
  - Chess: sequential; discrete; deterministic given moves
  - Untimed chess: static
  - Timed chess: semidynamic
  - Fixed crossword: static
  - Driving: partial; sequential; dynamic; partly continuous

Single-state, multiple-state and contingency problems

- State uncertainty representations
  - Single-state: known deterministic, fully observed
  - Sensorless/multiple-state: no observation
  - Known deterministic transitions
  - Belief state: possible-state set
- Contingency versus exploration
  - Nondeterminism or partial observation
  - Percept-dependent plan branches
  - Unknown state space or transition model
  - Exploration/online search
  - Observation alone insufficient

- Overlapping classifications
- Partial deterministic description: ambiguous category

Formulate before searching | Exam focus

- Goal then problem formulation
  - Initial state and legal actions
  - Transition model
  - Goal test and action costs
  - Search: sequence selection
  - Execution: applying actions
  - State abstraction: irrelevant details removed
- Search problem examples
  - Routes: cities; roads; neighboring successors
  - Accumulated distance or time
  - Solution: legal goal-ending path
  - Optimal: minimum path cost
  - Automatic programming/program synthesis
  - Program satisfying specification
  - Conditional plans and loops: richer representations

Frontiers and evaluation criteria | Exam focus

- Search nodes and frontier
  - State; parent; action; depth; path cost
  - Expansion generates successors
  - OPEN/frontier: pending candidates
  - CLOSED: expanded states
  - Same state ≠ same node/cost
- Search evaluation criteria
  - Completeness under stated assumptions
  - Optimality: best solution
  - Time: generated work
  - Space: simultaneous storage
  - b: branching; d: shallowest goal

- m: maximum depth; l: limit
- Graph V,E versus implicit-tree bounds

Breadth-first search | Exam focus

- BFS: FIFO oldest-first
  - Root → head removal → goal test
  - Unseen successors appended at tail
  - Depth order: 0,1,2,…
  - Complete: finite b, finite goal depth
  - Minimum number of edges
  - Minimum cost: equal positive step costs
- BFS complexity conventions
  - 1+b+…+b^d
  - Generation-time testing: O(b^d)
  - Depth-d expansion: O(b^(d+1))
  - Explicit visited graph: O(V+E) time
  - Explicit visited graph: O(V) space
- Worked example BFS queue trace
  - Children S:[A,B], A:[C,D]
  - Removal: S,A,B,C,D
  - Child-order assumption

Depth-first search | Exam focus

- DFS: deepest pending branch
  - LIFO stack or recursion
  - FILO = LIFO
  - Left-first trace: S,A,C before B
  - Push rightmost first
- DFS guarantees and bounds
  - Implicit-tree time: O(b^m)
  - Pending-sibling space: O(bm)
  - Path-only recursive space: O(m)
  - Generally nonoptimal

- Infinite-path and cycle failure
- Finite visited graph: complete
- Visited-state storage: O(V)
- Bank space O(b^m): time confusion

Uniform-cost search and Dijkstra | Exam focus

- UCS: minimum actual g(n)
  - Priority queue
  - Goal acceptance on removal
  - First-generation goal trap
  - Edge relaxation: cheaper path
  - Updated frontier cost and parent
- UCS and Dijkstra conditions
  - Finite branching; step cost ≥ε>0
  - UCS completeness and optimality
  - Finite graph: nonnegative edges, duplicate handling
  - Dijkstra: reachable minimum distances
  - UCS: goal stop, on-demand states
  - Queue initialization: implementation choice
  - Negative edges: finalized-distance failure

Depth Limits and Bidirectional Search | Exam focus

- Depth-limited and iterative deepening
  - DLS: DFS to l
  - Cutoff versus failure
  - Goals beyond l missed
  - Limits: 0,1,2,…
  - Finite b: shallowest solution
  - Time O(b^d); frontier space O(bd)
  - Optimal: equal step costs
  - Deepest level dominates repeated work
- Bidirectional search and IDA*
  - Start and goal frontiers

- Reverse actions and meeting tests
- Equal-cost BFS: O(b^(d/2)) time/space
- Weighted search: sound stopping bound
- First intersection ≠ cheapest
- IDA*: f=g+h threshold deepening

Heuristics and greedy best-first search | Exam focus

• Heuristic versus uninformed search
- h(n): remaining cost or promise
- Problem-specific knowledge
- Reduced exploration versus heuristic computation cost
- Uninformed: definition and known costs
- UCS uses g, remains uninformed

• Greedy best-first search
- Best-first family: evaluation-ordered OPEN
- Greedy f=h; minimum estimate
- Past cost ignored
- Expensive route trap
- Loops without duplicate checks
- Infinite misleading branches
- Finite graph, full duplicate handling: complete
- Generally nonoptimal
- Finite-tree worst case: O(b^m)

Admissibility, consistency and A* | Exam focus

• A*, admissibility and consistency
- f(n)=g(n)+h(n)
- Admissible: 0≤h(n)≤h*(n)
- h*: true cheapest remaining cost
- Consistent: h(n)≤c(n,n′)+h(n′)
- h(goal)=0
- Consistency ⇒ admissibility on goal-reaching states
- Consistency: nondecreasing path f

- A* guarantees and traps
  - Tree optimality: admissible h
  - No-reopen graph search: consistent h
  - Admissible inconsistent h: reopen improved g
  - Goal test on minimum-f removal
  - Infinite-space completeness: finite branching
  - Positive lower-bounded step costs
  - Exponential worst-case time and memory
  - No universal heuristic speed guarantee

Work a heuristic-search example | Exam focus

- Greedy versus A* trace
  - Edges: SA=2, SB=1, AG=2, BG=10
  - h(A)=2; h(B)=1; h(G)=0
  - Greedy B→G: cost 11
  - A*: f(A)=4, f(B)=2
  - Expand B; G cost 11
  - Expand A; replace G cost 4
  - Remove goal: optimal cost 4
- Heuristic lower-bound examples
  - Unit sliding-puzzle moves
  - Misplaced tiles excluding blank
  - Summed Manhattan distances excluding blank
  - Straight-line ≤ road distance
  - Travel-time conversion: valid speed bound

Local optimization and evolutionary search | Exam focus

- Local search and hill climbing
  - Complete candidate; objective improvement
  - Path may be irrelevant
  - First-improvement: first better neighbor
  - Steepest-ascent: all neighbors
  - Stochastic choice among improvements

- Local maxima; plateaus; ridges
- Random restarts; limited sideways steps
- No guaranteed success

• Genetic and means-ends search
- Population and fitness evaluation
- Parent selection
- Crossover; mutation; replacement
- No global optimality guarantee
- Means-ends: current-goal difference
- Difference-reducing operator
- Recursive precondition solving
- Backward A* misconception

Constraint satisfaction | Exam focus

• Constraint satisfaction problem
- Variables X; domains D; constraints C
- Solution: every variable, every constraint
- Complete: all variables assigned
- Consistent: no violated constraints
- Complete assignment may be inconsistent

• CSP search and propagation
- Backtracking: assign, try, undo failure
- Fewest remaining values
- Degree tie-breaker
- Least-constraining value
- Forward checking: incompatible neighbor removal
- Arc consistency: unsupported binary values removed
- Problem framework, not fourth uninformed traversal

Games and minimax | Exam focus

• Adversarial game formulation
- Competing objectives
- States and player to move

- Legal actions and successor function
- Terminal test and utility
- Perfect information: visible state/history
- Future choices still unknown
- Perfect information may include chance
- Zero-sum: opposite gains/losses

• Minimax and chance nodes
- Terminal utilities
- MIN: minimum; MAX: maximum
- Maximum-valued root action
- Best guarantee against optimal opposition
- Modeled-tree limitation
- Chance nodes: probability-weighted outcomes

• Worked example Minimax backup example
- MIN[3,5]=3
- MIN[2,9]=2
- MAX chooses first branch: 3

Alpha-beta and game evaluation | Exam focus

• Alpha-beta pruning
- Alpha: MAX lower bound
- Beta: MIN upper bound
- Bounds passed downward
- MAX updates alpha; MIN updates beta
- Prune: alpha≥beta
- Same-tree minimax value preserved
- Good move ordering
- Worst O(b^m); ideal O(b^(m/2))

• Cutoff evaluation functions
- Estimated position desirability
- Positive MAX; negative MIN
- +1/0/−1: win/draw/loss convention
- Material: weighted pieces

- Mobility: legal options
- Territory: controlled regions
- Combined features
- Limited computation, no victory guarantee

Represent knowledge for inference | Exam focus

- Knowledge representation and inference
  - Facts; objects; relations; events; rules
  - Syntax: legal expressions
  - Semantics: meaning
  - Knowledge base: assertions
  - Inference system: derived consequences
  - Declarative: what true
  - Procedural: how to act
- Knowledge representation structures
  - Semantic networks: labeled concept relations
  - is-a links
  - Frames: slots; values; defaults
  - Production systems: IF-THEN rules
  - Working-memory matching and rule selection
  - Graph: frequently-bought-together relationships
  - Frames: attributes
  - Recommendation requirements determine suitability

Propositional logic and models | Exam focus

- Propositional connectives
  - Atoms: whole truth-valued claims
  - ¬P: negation
  - P∧Q: both; P∨Q: at least one
  - P→Q false: P true, Q false
  - P↔Q: matching truth values
  - P→Q ≡ ¬P∨Q
  - De Morgan: ¬(P∧Q) ≡ ¬P∨¬Q

- Models, validity and entailment
  - Model: satisfying interpretation
  - Valid: every interpretation
  - Satisfiable: at least one
  - Unsatisfiable: no interpretations
  - Invalid ≠ unsatisfiable
  - P∨¬P: valid
  - P: satisfiable, invalid
  - P∧¬P: unsatisfiable
  - KB⊨α: all KB models satisfy α

First-order logic and quantifier scope | Exam focus

- First-order logic vocabulary
  - Objects; predicates; functions; quantified variables
  - Human(a): property
  - Likes(a,b): relation
  - father(a): function term
  - Propositional logic: no object quantifiers
  - Assigned atom values determine truth
- Quantifier translation and scope
  - All humans: ∀x(Human(x)→Mortal(x))
  - Tall student: ∃x(Student(x)∧Tall(x))
  - Translation patterns, not connective prohibitions
  - ∀x∃y Likes(x,y): potentially different y
  - ∃y∀x Likes(x,y): shared y
  - ¬∀x P(x) ≡ ∃x¬P(x)
  - Negation swaps quantifiers
  - Bound occurrences within correct scope
- Worked example Existential conjunction trap
  - ∃x(StandingAt(x,door)∧Tall(x))
  - Same individual, both predicates
  - Implication: unrelated vacuous witness

Inference, unification and chaining | Exam focus

- Inference and unification
  - Modus ponens: P,P→Q ⊢ Q
  - Modus tollens: P→Q,¬Q ⊢ ¬P
  - Affirming consequent trap
  - Universal instantiation: object substitution
  - Consistent unifying substitution
  - Knows(x,Ada),Knows(Ben,y): x=Ben,y=Ada
  - Distinct constants cannot unify
- Resolution and chaining
  - P∨Q,¬P∨R ⊢ Q∨R
  - Refutation: KB plus ¬α
  - Clause form and contradiction
  - Forward chaining: facts → rules
  - Backward chaining: query → required premises
  - Soundness: truth preservation
  - Completeness: all entailed conclusions
  - Specified-setting qualification

Deduction, induction and abduction | Exam focus

- Reasoning forms
  - Deduction: truth-preserving rules
  - Mortal people + Ada → mortal Ada
  - Induction: observations → generalization
  - New examples may overturn induction
  - Abduction: plausible explanation
  - Wet ground: rain or sprinklers
  - Analogy: similar-situation pattern transfer
- Monotonic and default reasoning
  - Classical entailment: consequences persist
  - Added premises preserve deductions
  - Nonmonotonic defaults: retractable conclusions
  - Bird usually flies; penguin exception

- Incomplete knowledge; no truth guarantee
- Blind search ≠ reasoning definition

Probability and uncertain models | Exam focus

- Probability and Bayes
  - $P(\neg A)=1-P(A)$
  - $P(A|B)=P(A\cap B)/P(B)$, $P(B)>0$
  - $P(H|E)=P(E|H)P(H)/P(E)$
  - Posterior; likelihood; prior; evidence
  - Exclusive exhaustive hypotheses
  - $P(E)=\Sigma_i P(E|Hi)P(Hi)$
- Uncertain graphical models
  - Bayesian network: directed acyclic graph
  - Conditional probability tables
  - Joint: product of parent-conditioned probabilities
  - Markov: next depends on current state
  - Entire-history dependence excluded by assumption
  - Uncertainty versus fuzzy membership
- Worked example Bayes numerical method
  - Prior 0.1; likelihood 0.8
  - Alternative likelihood 0.2
  - Evidence: $0.08+0.18=0.26$
  - Posterior: $0.08/0.26\approx 0.308$
  - Likelihood ≠ posterior

Fuzzy logic and vague categories | Exam focus

- Fuzzy membership versus probability
  - Graded membership: $0\leq\mu(x)\leq 1$
  - Warm/tall vague concepts
  - Warm degree 0.7 ≠ probability 70%
  - Boolean: true/false
  - Probability: uncertain events
- Fuzzy controller pipeline

- Fuzzify measurements
- Graded IF-THEN rules
- Combine outputs
- Defuzzify numerical action
- Fuzzy control: linguistic degrees
- RL: reward-based adaptation
- Complex/uncertain environment: insufficient method criterion

Supervised learning and prediction | Exam focus

- Supervised prediction
  - Labeled input-target mapping
  - Classification: discrete labels
  - Institution generations; spam/nonspam
  - Regression: numeric targets, prices
  - Predictive analytics: historical-pattern forecasts
  - Classification or regression
  - Target before method
- Decision trees and impurity
  - Internal attribute tests; branches; leaf predictions
  - ID3: high information gain
  - $H(S)=-\Sigma p_i \log_2 p_i$
  - Gain: parent minus weighted child entropy
  - $Gini=1-\Sigma p_i^2$
  - Pure class: Gini zero
  - Pruning against overfitting
  - Supervised neural networks, SVMs, trees

Unsupervised, Semi-supervised and Active Learning | Exam focus

- Unsupervised structure discovery
  - No target labels
  - Clustering: similar-example groups
  - Association: co-occurrence
  - Dimensionality reduction: feature compression

- High within-cluster similarity
- Low between-cluster similarity
- Appropriate distance
- K-means: nearest assignment ↔ centroid updates

• Semi-supervised and active learning
- Semi-supervised: labeled plus unlabeled
- Active: informative oracle-label queries
- Active learning still uses labels
- Exploratory learning: ambiguous terminology
- Unsupervised exploration versus active acquisition
- Feedback definition
- Summarization/time-series labels depend on training targets

Reinforcement learning | Exam focus

• Reinforcement learning interaction
- Observe → act → reward/new observation
- Policy improvement
- Expected cumulative reward objective
- Outcome reward ≠ correct-action label
- Delayed rewards
- Correct-looking action without immediate reward

• RL return and policy
- $Gt=rt+1+\gamma rt+2+\gamma^2 rt+3+\ldots$
- $0\leq\gamma<1$
- Policy: states/histories → action choices
- Exploration: information
- Exploitation: currently estimated reward
- RL⊂ML⊂AI
- No labels: RL or unsupervised
- Distinguishing reward-driven interaction

Neural networks and deep learning | Exam focus

• Neuron and activation functions

- z=Σiwixi+b; a=φ(z)
- x=(2,1), w=(0.5,−1), b=0.2
- Weighted sum: z=0.2
- Linear: z
- Sigmoid: 1/(1+e^(−z))
- tanh: bipolar
- ReLU: max(0,z)
- Nonlinear multilayer relationships

• Network architectures and training
- Feedforward: output-directed, no recurrent loops
- Recurrent: state across sequence steps
- Weight adjustment through gradients/backpropagation
- Backpropagation ≠ recurrent connection
- Deep learning: multiple representation layers
- Learned image, sound, text features
- CNN: shared filters across image locations
- ANN: brain-inspired abstraction
- Literal brain-simulation trap
- DL⊂ML⊂AI

Generalization and validation | Exam focus

• Generalization failures and controls
- Overfitting: training details, poor unseen performance
- Underfitting: limited or poorly fitted model
- Learning curve: performance versus training size/progress
- Complexity control
- Regularization; pruning
- More representative data

• Training, validation and test splits
- Training: parameter fitting
- Validation: model/settings selection
- Held-out test: final estimation
- k-fold: k models, k−1 training folds

- Remaining-fold evaluation; average scores
- Stratification: approximate label proportions
- Preprocessing fitted inside training folds
- Time-respecting splits
- Repeated test tuning: information leakage

Measure the right errors | Exam focus

• Classification confusion matrix
- Fixed positive class
- TP: true positives; TN: true negatives
- FP: false alarms; FN: missed positives
- Accuracy=(TP+TN)/N
- Precision=TP/(TP+FP)
- Recall=TP/(TP+FN)
- F1=2PR/(P+R)
- Averaging and zero-denominator conventions

• Metric calculation and selection
- TP=30, FP=10, FN=20, TN=40
- Accuracy=70%; precision=75%; recall=60%
- F1≈66.7%
- Rare-class failure hidden by accuracy
- Missed positives: recall
- Costly false alarms: precision
- MAE=mean$|y-\hat{y}|$
- MSE=mean$(y-\hat{y})^2$
- Squared errors emphasize large misses

Language processing tasks | Exam focus

• Natural language processing tasks
- Language interpretation and generation
- Translation; dialogue; summarization
- Question answering; text generation
- Speech recognition: audio → text

- Speech synthesis: generated speech
- Overlapping fields, distinct input/output tasks

• Language processing levels
- Tokenization: words; punctuation; subwords
- POS tagging: grammatical roles
- NER: people; organizations; locations
- Parsing: grammatical structure
- Semantics: meaning
- Discourse: sentence links
- Pragmatics: context and speaker intent
- Sentiment: evaluative tone
- Bank: lexical ambiguity
- Telescope: attachment ambiguity

Morphology, sounds and prosody

• Morphology and speech sounds
- Meaningful word parts
- un-happi-ness: prefix; stem; suffix
- cats: cat + plural -s
- Tokenization: processing boundaries
- Phonetics: physical speech sounds
- Phonology: sound patterns and contrasts

• Prosody and contextual meaning
- Stress; rhythm; timing
- Prominence and intonation
- Syllables and phrases
- Rising pitch: possible question
- Stress shifts: emphasis
- Dialogue and speech-processing cues
- Pragmatics: intended contextual meaning
- Pragmatics ≠ word-internal structure

Perception, vision and robotics | Exam focus

- Computer vision tasks and limits
  - Image/video interpretation
  - Classification: content names
  - Detection: object locations
  - Segmentation: pixel labels
  - Recognition: object/person identity
  - Occlusion; lighting changes; noise
  - Unfamiliar conditions; incomplete context
  - Face detection versus open-world perception
- Robotic sensing, planning and action
  - Mechanical; electrical; control; computing systems
  - Sensors; controller; power supply
  - Actuators and end effector
  - Legged; wheeled; tracked locomotion
  - Terrain; stability; efficiency trade-offs
  - Autonomous software ≠ robot
  - Humanoid shape ≠ general intelligence

Expert systems and choosing methods | Exam focus

- Expert system components
  - Specialized domain knowledge; equipment diagnosis
  - Domain expert: expertise
  - Knowledge engineer: elicitation and encoding
  - Inference engine: knowledge base/working facts
  - User interface: interaction
  - Explanation facility: why/how conclusions
- Method selection and responsible evaluation
  - Rules: inspectable, brittle, maintenance cost
  - Learned patterns: representative data and validation
  - Search: explicit action alternatives
  - Probability: uncertain evidence
  - Data/evaluation bias and unfairness

- Group-specific errors
- Robustness; privacy; human oversight
- Automation benefit assumption trap

Clarify search and logic questions

• Search-question clarifications
- A*: forward best-first, g+h
- No inherent backward frontier
- A*/IDA*: conditional optimality
- A*/IDA*: exponential worst cases
- Minimum heuristic = estimated closest
- LIFO = FILO

• Ambiguous search and validity stems
- Current-state information ≠ heuristic proof
- Estimated remaining cost criterion
- False: invalid iff unsatisfiable
- False: satisfiable implies valid
- Multiple false options
- Repair: all false or specified claim

Distinguish agent categories and components

• Agent-class naming conventions
- Five including learning
- Four decision architectures before learning
- Count requires convention
- Target-based may mean goal-based
- Unity-based ≠ utility-based

• Agent classification evidence
- Generic rule: memory unspecified
- Simple versus model-based reflex memory
- Unused goal ≠ goal-based policy
- Sensors/effectors: interaction only
- Interaction ≠ omniscience/intelligence

- Effector/actuator options: nonunique without distinction

Clarify overlapping learning and AI labels

- Overlapping learning labels
  - ANN; genetic search; RL improvement
  - Experience-based performance changes
  - Deep learning: architecture family
  - Supervised; unsupervised; reinforcement objectives
  - Trees and deep models: potentially supervised
  - Regime evidence: labels, rewards, unlabeled structure
- AI-label question traps
  - Face recognition: applied and narrow
  - Emulating intelligence and building learning systems
  - Humanlike problem-solving: insufficient expert-system criterion
  - Python; C++; Java implementations
  - Common language: nonunique answer
  - McCarthy: naming, not sole invention

Source Damage and Assumption Repairs

- Quantifier negation
  - $\forall x \exists y \exists t \neg F(x,y,t)$
  - Each person: failed fooling person/time
  - Equivalent: $\neg \exists x \forall y \forall t\ F(x,y,t)$
  - Nobody fools everybody all the time
  - Correct D; bold C wrong
  - C: stronger prohibition on every success
  - Read the formula carefully
- Implementation and task assumptions
  - Dijkstra/UCS: stated implementations
  - Queue initialization: nonintrinsic difference
  - Chess moves: sequential
  - Independently reset matches: episodic
  - Recommendation suitability requires requirements

- Graph relationships: support, not exclusivity

Computer Security

Revision Outlines

CIA and the property actually harmed | Exam focus

- CIA: property actually harmed
  - Confidentiality: restricted disclosure
  - Credit-card copying violates secrecy
  - Intact original ≠ preserved confidentiality
  - Integrity: unauthorized/improper change prevention
  - Altered balance; inserted false message
  - Availability: timely usable authorized service
  - Crashes; resource starvation; denied access
  - One event may harm multiple goals
- Identity, evidence and information assurance
  - Authentication: identity or claim establishment
  - Authorization: permitted actions
  - Accountability: responsible entities linked to actions
  - Reliable identity and protected logs
  - Non-repudiation: origin/receipt evidence
  - Signatures ≠ physically impossible denial
  - Assurance: confidence, risk management, protection
  - Detection; response; restoration; information lifecycle
  - Data at rest, in use, in transit
  - Accidental corruption and intentional attacks

Threat, vulnerability, risk and control | Exam focus

- Asset, threat, vulnerability, attack, control
  - Assets: data, software, hardware, service, reputation
  - Threat: potentially harmful circumstance
  - Vulnerability: exploitable weakness
  - Attack: actual attempt
  - Control: reduced likelihood or impact

- Software flaw ≠ active attack
- Risk: likelihood and consequence
- Unmeasured labels ≠ universally meaningful product

• Control purposes and categories
- Preventive: block; detective: reveal
- Corrective/recovery: restore safe operation
- Deterrent: discourage attempts
- Encryption: disclosure prevention
- Audit log: detection and accountability evidence
- Physical; technical; administrative categories
- Categories ≠ competing stages
- Firewall cannot replace policy or locks

Attack classifications and attacker intent

• Attack effects and 4D
- Interception: observe/copy
- Modification: alter
- Interruption: block
- Fabrication: false insertion
- Disclosure; deception; disruption; usurpation
- Usurpation: unauthorized control
- DoS: disruption and availability loss
- DoS need not obtain control

• Passive versus active attacks
- Passive: unchanged target exchange
- Sniffing; traffic analysis
- Active: inject, alter, replay, disrupt
- Passive/active ≠ offline/online
- Encrypted traffic still exposes timing/size
- Traffic analysis infers relationships
- Neither universally easy to prevent/detect

• Attacker intent and trust position
- White hat: authorization and scope

- Black hat: malicious or unauthorized
- Competitor: commercial secrets
- Hacktivist: cause
- Hacker label ≠ legality
- Insider/outsider: origin and trust position
- Origin ≠ attack mechanism

Malware: propagation, disguise and payload

• Malware replication and disguise
- Virus: attaches to program/document host
- Infected-host execution spreads virus
- Worm: independent replication, commonly networked
- Trojan: useful disguise, malicious behavior
- Trojan need not self-replicate
- Labels describe different properties
- Package may contain multiple components

• Malware payloads and defenses
- Spyware: monitoring; keylogger: keystrokes
- Adware: unwanted ads, possible tracking
- Ransomware: encryption/locking and extortion
- Logic bomb: trigger-dependent
- Backdoor: ordinary-access bypass
- Rootkit: hidden presence/privileged activity
- Zombie/bot: remote commands; botnet: collection
- Malware: umbrella, not replication method
- Update; restrict execution; isolate compromise
- Tested offline backups
- Antivirus never guarantees complete protection

Social engineering, spoofing and sessions

• Social engineering
- Phishing: trusted-source impersonation
- Secret theft or induced unsafe action

- Spear phishing: specific person
- Verify through independently known channel
- Shoulder surfing: observed entry
- Pretexting: invented disclosure reason
- Authorized monitoring ≠ necessarily attack

• Spoofing, replay and session attacks
- Masquerade: another identity
- ARP poisoning: local IPv4-to-MAC associations
- DNS spoofing/cache poisoning: false resolutions
- IP spoofing: false source address
- Intermediary: interception and possible alteration
- Replay: reuse valid old message
- Replay defenses: nonces, timestamps, sequence checks
- Session hijacking: stolen/predicted valid token
- Protect transport; regenerate IDs after login
- Restricted cookies; per-request server authorization

DoS, wireless attacks and layer-specific defenses

• Resource-exhaustion attacks
- DoS: exhaust or disable resource
- DDoS: multiple sources
- Bots; reflected/amplified responses
- SYN flood: incomplete-handshake state
- DNS flooding: query capacity
- NXDOMAIN: nonexistent-name resolver stress
- Fragment floods: reassembly/processing exhaustion
- Rate limits; SYN cookies; filtering
- Upstream capacity controls
- Defense depends on bottleneck

• Wireless attacks and defense layers
- Rogue AP: imitate/bypass approved service
- Bluesnarfing: Bluetooth information theft
- Bluejacking: unsolicited Bluetooth messages

- Phishing not wireless-specific
- Authenticated IPsec: IP impersonation resistance
- DNSSEC: DNS-data authentication
- TLS: application-transport authentication/protection
- Bounds-safe code: overflow prevention
- HTTPS cannot repair unsafe memory copy

Identification, authentication and factors | Exam focus

- Access-control sequence and direction
  - Identify → authenticate → authorize → record
  - Identification: username/claim, not proof
  - Identification omitted: authentication first available stage
  - User-to-host: user proves identity to host
  - Host-to-user: host proves identity to user
  - Arrow: prover → verifier
- Authentication factors
  - Knowledge: password/PIN
  - Possession: token/card
  - Inherence: biometric, fingerprint
  - Behavioral: typing rhythm, handwriting
  - Textbook category: something one does
  - One biometric ≠ three factors
  - Password plus PIN: one factor class
  - Card plus PIN: two factors
  - OTP factor depends on generation/delivery

Passwords and delegated authorization

- Password attacks and protection
  - Brute force: enumerate guesses
  - Dictionary: prioritize likely words
  - Credential stuffing: reuse leaked credentials
  - Online: server rate limits
  - Offline stolen verifiers: no server limits

- Short secrets vulnerable in either setting
- Unique long secrets; password manager
- Rate limiting; suitable MFA
- Salted slow hashes, not reversible plaintext
- Public salt defeats shared precomputed lookup

• OAuth authorization framework
- Client: limited access tokens
- Authorization server: authentication/owner approval
- Resource server: protected-data token validation
- Authorization server may delegate identity checks
- OAuth alone ≠ complete login identity protocol
- OpenID Connect: added identity layer

Access control and least privilege | Exam focus

• Subjects, objects and ACLs
- Subjects request operations on objects
- Object ACL: allowed subjects/actions
- Network ACL: address, protocol, port matching
- Direct effects: classification and filtering
- ACL ≠ guaranteed availability/malware detection

• Access-control models
- DAC: owner grants permissions
- Mandatory: enforced labels and policy
- Ordinary owners cannot override mandatory policy
- RBAC: permissions→roles; users→roles
- ABAC: subject/object/action/environment attributes
- Rule-based: explicit conditions
- Time or source-network rules

• Privilege restrictions
- Least privilege: needed rights, needed time
- Separation of duties: split sensitive steps
- Deny by default
- Mediate every access

- Revoke stale permissions
- SELECT allowed; UPDATE denied
- Access control supports integrity
- Business-state validity needs additional enforcement

Confidentiality and integrity models | Exam focus

- Bell-LaPadula confidentiality
  - Simple security: no read up
  - Star: no write down
  - Strong star: same-level writes only
  - TS>S>C>U
  - S subject may read C
  - S subject cannot write C
  - Compartments and DAC may further deny
- Integrity models
  - Strict Biba: no read down
  - Strict Biba: no write up
  - Prevent low-integrity input contaminating higher results
  - Clark-Wilson: certified well-formed transformations
  - Integrity checks; separation of duties
  - Clark-Wilson ≠ confidentiality model
  - MAC: enforcement category
  - Mandatory policy need not be Bell-LaPadula

Physical protection and resilient design

- Physical protection and threat models
  - Theft; unauthorized connections
  - Heat; humidity; water; surges; fire
  - UPS: temporary conditioned power
  - Faulty installation/battery: possible hazard
  - Disconnected machine: fewer network paths
  - Insiders; removable media; supply-chain attacks
  - Physical access remains a threat

- Most-secure claim requires threat model

• Resilient security design
  - Defense in depth: independent layers
  - Minimization: remove unnecessary software/services
  - Compartmentalization: limit blast radius
  - Fail securely: errors never silently grant
  - Uniformity: easier patching and prediction
  - Uniformity: correlated failures
  - Diversity: reduced common-mode compromise
  - Diversity: higher administration cost
  - Protection proportional to assets and threats

Cryptographic vocabulary and security assumptions

• Cryptographic vocabulary
  - Plaintext: original; ciphertext: encrypted representation
  - Key: algorithm input
  - C=E_k(P); P=D_k(C)
  - Symmetric five: plaintext, algorithm, secret key
  - Ciphertext and decryption algorithm
  - Cryptanalysis: unauthorized recovery or breaks
  - Cryptology: cryptography plus cryptanalysis
  - Steganography: hidden existence
  - Encryption: hidden meaning

• Cryptographic security assumptions
  - Known algorithm; protected keys
  - Implementation secrecy not main argument
  - b-bit key → 2^b possibilities
  - Uniform exhaustive-search average: half keyspace
  - Algorithm; key size; randomness
  - Mode and implementation matter
  - Symmetric ≠ inherently less secure
  - Speed ratios not fixed constants

Substitution, shift and affine ciphers

- Substitution versus transposition
  - Substitution: replace symbols
  - Transposition: rearrange positions
  - Monoalphabetic: one mapping throughout
  - Frequency leakage despite large nominal keyspace
  - Classical ciphers: teaching, not modern protection
- Caesar, multiplicative and affine arithmetic
  - Alphabet mapping: A=0…Z=25
  - Caesar: C=(P+k) mod26
  - Inverse: P=(C−k) mod26
  - k=3: A→D; HELLO→KHOOR
  - k=25: ATTACK→ZSSZBJ
  - Wrap negatives into 0-25
  - Multiplicative: C=kP mod26; gcd(k,26)=1
  - Decrypt with modular inverse of k
  - Affine: C=aP+b mod26; gcd(a,26)=1
  - Inverse relation, not repeated numeric multiplication

Vigenère, Playfair and Hill

- Vigenère and autokey
  - C_i=(P_i+K_i) mod26
  - Repeated or otherwise supplied shifts
  - Same letter may encrypt differently
  - Repeated short keys leak periodicity
  - Autokey extends initial key with message
- Playfair method
  - Keyed 5×5 grid; commonly merged I/J
  - Encrypt letter pairs
  - Specify duplicate fillers and padding
  - Same row: right
  - Same column: down
  - Rectangle: other column, original row

- Decrypt by reversed shifts
- Hill cipher and known plaintext
  - Letter vectors: C=KP mod26
  - Invertible K modulo26
  - gcd(det(K),26)=1
  - Linear structure
  - Enough independent known blocks reveal key
  - K=CP^-1 mod26
  - Claimed known-plaintext resistance: false
  - Polygraphic substitution ≠ modern strength

Transposition and one-time pads

- Transposition procedures and conventions
  - Rail fence: specified zigzag, read rails
  - Columnar: write rows, ordered columns
  - Symbols and counts preserved
  - Specify rows; column order; padding; direction
  - Inconsistent source-example conventions
  - Two rails ABCDEF: ACE and BDF
  - Ciphertext ACEBDF
- One-time pad conditions
  - XOR with independent uniformly random secret
  - Key at least message length
  - Never reuse key
  - Perfect secrecy requires every condition
  - Reuse: C1 XOR C2=P1 XOR P2
  - Vernam XOR ≠ automatically OTP
  - Predictable/reused keystream loses guarantee
  - Key delivery and storage costs

Symmetric keys, AES and distribution

- Symmetric keys and distribution
  - Shared secret; efficient bulk protection

- n users: n(n−1)/2 pairwise keys
- Trusted KDC changes architecture
- Session key: temporary communication context
- Master/key-encryption key: protects subordinate keys
- Physical distribution; authenticated protected channel
- Trusted third-party distribution
- Uncompromised old key protects new key
- Stolen old key prevents secrecy restoration

• AES and cipher construction

- AES block: 128 bits
- Keys: 128/192/256 bits
- Rounds: 10/12/14
- Substitution/permutation design
- SubBytes; ShiftRows; MixColumns; AddRoundKey
- Final round omits MixColumns
- Block cipher: mode and nonce/IV handling
- Authenticated encryption detects tampering
- Stream cipher: data plus keystream
- Nonce uniqueness when construction requires

DES and Feistel mechanics

• DES Feistel mechanics

- Block: 64 bits
- Key: 56 effective bits, eight parity
- 16 rounds; initial permutation
- L_i=R_(i−1)
- R_i=L_(i−1) XOR f(R_(i−1),K_i)
- Right-half expansion: 48 bits
- XOR with 48-bit subkey
- Eight S-boxes: six→four bits each
- Permutation: 32 output bits
- S-box row: outer input bits
- S-box column: inner four bits

- Feistel inversion and DES limitations
  - Nonlinear S-boxes resist simple linear descriptions
  - Same structure, reversed round keys
  - Round function need not be invertible
  - Final swap/permutation
  - DES and triple DES: legacy
  - Not choices for new protection
  - Three 56-bit keys ≠ 168-bit effective security

Public keys and hybrid protection

- Public-key operation direction
  - Confidentiality to B: B's authenticated public key
  - B decrypts with B's private key
  - A's public key targets A, not B
  - A need not possess B's private key
  - A signs: A's private signing key
  - Others verify: A's public key
  - Signing and encryption are distinct operations
  - Not every public-key algorithm encrypts
- Hybrid protection and Diffie-Hellman
  - Public keys require authentic identity bindings
  - Public-key secret establishment/encapsulation
  - Symmetric bulk-data protection
  - Unauthenticated DH permits intermediary substitution
  - Public p,g; private a,b
  - A=g^a modp; B=g^b modp
  - B^a=A^b=g^(ab) modp
  - Authenticate exchange; derive separate traffic keys
- Sender key-pair requirements
  - One-way confidentiality: recipient authenticated public key
  - Sender key pair not inherently needed
  - Sender signature requires sender key pair
  - Separate recipient role may require pair

RSA arithmetic and key-validity conditions

- Textbook RSA construction
  - Distinct primes p,q
  - n=pq; φ=(p−1)(q−1)
  - 1<e<φ; gcd(e,φ)=1
  - d=e^-1 modφ
  - Public (n,e); private includes d
  - 0≤m<n
  - c=m^e modn; m=c^d modn
  - Large exponents: modular repeated squaring
  - Real RSA requires standardized padding
  - Raw arithmetic ≠ secure deployment
- RSA valid versus defective examples
  - p=7,q=19 → n=133,φ=108
  - Smallest admissible e>1: 5
  - e=7 and other coprimes also valid
  - e=5 → d=65
  - 325≡1 mod108
  - p=3,q=13,e=3,m=5
  - 5^3 mod39=8
  - gcd(3,24)=3 → no inverse
  - Arithmetic result ≠ valid RSA key
  - State defect; reject claimed key pair

Hashes, MACs and signatures

- Hash versus keyed MAC
  - Arbitrary input → fixed-length digest
  - Deterministic; preimage and collision resistance
  - Collisions necessarily exist
  - Not mathematically unique fingerprints
  - Unkeyed hash can be attacker-recomputed
  - Hash alone needs trusted reference
  - MAC: integrity, shared-secret authentication

- Both holders can create MAC
- Limited third-party non-repudiation

• Signatures and password hashes
- Private-key message/encoded-digest signing
- Public-key verification
- Signing ≠ generic private-key encryption
- DSA/ECDSA counterexamples
- Origin/integrity: identity and custody assumptions
- Signature does not conceal content
- Passwords: salted, deliberately expensive hash
- MD5/SHA-1 history ≠ modern collision-resistant choice

Certificates, TLS and HTTPS

• Certificate trust validation
- Issuer signature binds key to subject
- Chain to trusted root
- Intended hostname; time validity
- Applicable key usage and revocation policy
- Genuine certificate ≠ honest business

• TLS and HTTPS protection boundaries
- TLS succeeds obsolete SSL
- Authenticated protected channel
- Handshake: parameters, authentication, key establishment
- Server authentication; optional client authentication
- Records: symmetric authenticated encryption
- Messages vary with TLS version
- ChangeCipherSpec narrative not universal TLS1.3
- HTTPS=HTTP over TLS
- HTTP paths, headers, content protected
- IP endpoints/timing metadata remain visible
- Compromised decrypted endpoint remains exposed

SSH, IPsec, email protection and SET

- SSH, IPsec and VPN
  - SSH: remote login, commands, SFTP
  - Host keys; multiple client-authentication methods
  - IPsec: network-layer IP protection
  - AH: integrity/origin authentication, no encryption
  - ESP: confidentiality and integrity possible
  - IKE: security associations and keys
  - Transport mode: upper-layer payload
  - Tunnel mode: inner IP, outer header
  - VPN: logical private connection
  - Security depends on authenticated tunnel
- Email protection and SET
  - PGP/OpenPGP: symmetric content protection
  - Public-key key handling and signatures
  - S/MIME: certificate-based signed/enveloped MIME
  - Separate/clear signatures need not conceal
  - Envelope: encrypted content, wrapped recipient keys
  - SET: Secure Electronic Transaction
  - Visa/Mastercard-led card-transaction specification
  - Certificates and signatures
  - SET ≠ payment processor or ordinary HTTPS
- S/MIME sign-then-encrypt construction
  - Sign MIME entity
  - Package content and signature
  - Encrypt signed entity with fresh symmetric key
  - Protect key copy per recipient
  - Recipient: recover key → decrypt → verify
  - Validated signer certificate/public key
  - Signing ≠ generic private-key encryption
  - Other nesting orders possible
  - Specify order and protection coverage

Firewalls and protected boundaries | Exam focus

- Firewall types
  - Traffic-policy enforcement
  - Stateless filtering: source/destination, protocol, ports
  - Stateful inspection: permitted connection context
  - Application proxy: terminate, relay, inspect
  - Circuit gateway: connection relay, commonly SOCKS
  - No full application-content interpretation
  - Textbook session/transport association
  - Circuit gateway need not encrypt
  - Gateway ≠ automatically VPN
- Firewall placement and limits
  - Host firewall: individual machine
  - Network firewall: segmentation and DMZ
  - DMZ: public-facing services
  - Explicit ordered least-privilege rules; appropriate logging
  - Stateful/next-generation devices also filter ports
  - Port-filter question nonexclusive
  - Bypass paths; insiders; uninspected encrypted content
  - Infected removable media
  - Hardware firewall ≠ fixed port count

Detection, prevention and diagnostic measures

- Detection versus prevention
  - IDS: observe and alert
  - IPS: enforce blocking, usually inline
  - Host sensors: host events
  - Network sensors: traffic
  - Signature detection: known patterns
  - Anomaly detection: baseline deviations, possible novelty
  - Neither detects every attack
  - Anomaly false positives
  - Signatures can sometimes match variants

- Diagnostic measures and obfuscation
  - TP: real attack detected
  - FN: attack missed
  - FP: innocent activity flagged
  - TN: innocent activity accepted
  - Recall=TP/(TP+FN)
  - Precision=TP/(TP+FP)
  - 90/100 spam; five innocent flags
  - Recall=90%; precision=90/95≈94.7%
  - Obfuscation: harder code understanding/analysis
  - No reliable secret protection or vulnerability removal

Memory safety and secure input handling | Exam focus

- Buffer overflows and defenses
  - Write beyond allocated bounds
  - Data/control corruption; possible code execution
  - Execution neither definition nor inevitable outcome
  - Eight-byte array ≠ eight characters plus zero
  - Bounds-check lengths and indexes
  - Consider integer overflow
  - Safe abstractions; validate library returns
  - Canaries; non-executable memory; address randomization
  - Exploit mitigation ≠ repaired underlying write
- Trust-boundary validation and patches
  - Expected type, length, range, structure
  - Server validation: clients controllable
  - Destination-specific output encoding
  - Validation ≠ query parameterization
  - Validation ≠ HTML-context encoding
  - Patches: known weakness repair
  - Zero-day may lack vendor fix
  - Segmentation and mitigations still useful

SQL injection, XSS and request forgery | Exam focus

- SQL injection prevention
  - Unsafe strings let values alter syntax
  - Tautology changes matching rows
  - Prepared statements bind data values
  - Allow-list nonparameterizable identifiers
  - Least-privilege application database account
  - Stored procedures/ORMs require safe SQL
  - Unsafe dynamic SQL remains vulnerable
- XSS mechanisms and defenses
  - Attacker script under victim-site origin
  - Stored: persistent; reflected: response-returned
  - DOM-based: unsafe browser sinks
  - Safe text insertion
  - Context-specific encoding
  - Careful allowed-HTML sanitization
  - CSP: added protection, not sole fix
- CSRF, cookie limits and fuzzing
  - Ambient credentials enable unwanted authenticated action
  - Unpredictable request tokens
  - Suitable cookie/origin controls
  - HttpOnly: restricted script cookie access
  - HttpOnly ≠ all XSS prevention
  - Fuzzing: unexpected-input fault discovery
  - Testing technique ≠ injection category

Races, testing and secure development | Exam focus

- Race conditions
  - Timing/interleaving-dependent shared-state correctness
  - Check-then-open: replacement between check/use
  - Two workers read100, each writes90
  - Lost debit
  - Protect whole invariant

- Suitable atomic operations, locks, transactions
- Unrelated-variable lock ineffective

• Security testing and development
- Vulnerability assessment: find/prioritize weaknesses
- Authorized penetration test: demonstrate consequences
- Static: inspect code; dynamic: execute
- Fuzzing varies inputs
- Host/network/database scanners: different exposure
- False results possible
- Correct → patch → test → recheck
- Testing ≠ proof of absence
- Threat modeling; review; least privilege
- Dependency management; safe error handling

Trusted OS requirements and reference mediation | Exam focus

• Trusted operating system requirements
- Enforce stated security policy
- Identification/authentication; process and memory isolation
- Controlled privileges; object-access enforcement
- Protected auditing; safe resource reuse
- Trusted boot/update paths; recovery behavior
- TCB: policy-critical components
- Smaller analyzable boundaries simplify assurance

• Reference monitor and assurance
- Abstract access-checking mechanism
- Complete mediation; tamper resistance
- Small/simple enough for analysis
- Security kernel: core enforcement implementation
- Firewall mediates traversing traffic only
- Firewall ≠ every host-memory access
- Assurance: justified confidence in implementation
- Functionality: claimed control behavior
- GUI/multitasking alone ≠ trust

Independent evaluation, certification and assurance | Exam focus

- Common Criteria evaluation roles
  - TOE: Target of Evaluation
  - Specified configuration and environment
  - Security Target: threats, assumptions, objectives
  - Functional and assurance requirements
  - Protection Profile: reusable product-category requirements
  - Evaluator: evidence, testing, vulnerability analysis
  - Certification body: oversight/review and decision
  - Sponsor commissions; developer supplies product/evidence
- Evaluation Assurance Levels and scope
  - EAL: assurance-requirement package
  - EAL1-7: increasing evaluation rigor
  - EAL ≠ universal security-function ranking
  - Higher EAL across scopes ≠ safer
  - CCRA: mutual recognition, not worldwide laboratory
  - Certificate: specified product/configuration and claims
  - Certification ≠ future-attack immunity

Database secrecy, permissions and key boundaries | Exam focus

- Database protection and permissions
  - Confidentiality; integrity; availability; accountability
  - Authenticate connections
  - Roles: necessary table/column/row privileges only
  - Restrict views; audit sensitive changes
  - Report permission ≠ underlying full-table access
  - Application authorization plus database rights
- Database encryption and replication limits
  - At-rest encryption: stolen storage/backups
  - Keys must remain separate
  - Transport encryption: connection protection
  - Neither blocks fully authorized queries
  - Neither guarantees correctness

- Access controls, encryption, protected transmission
- Secrecy question may have multiple answers
- Replication: availability plus malicious-update copying
- Separate recovery controls still needed

Database integrity and reliability mechanisms | Exam focus

- Database integrity mechanisms
  - Valid data and valid transitions
  - NOT NULL: mandatory values
  - UNIQUE/primary key: duplicate-identifier prevention
  - Foreign key: reference preservation
  - CHECK: declared predicates
  - Atomic transaction: transfer debit and credit
  - Concurrency control: safe interleavings
  - Roles/ACLs: who may update
  - Constraints/transaction logic: authorized update validity
- Logging, backup and reliability
  - Write-ahead logging: recovery records before data
  - Crash recovery
  - Backups; restore tests; separately protected logs
  - Deletion/corruption recovery
  - Replication: alternate service, not backup replacement
  - Redundancy needs appropriate failover
  - Independent failure handling
  - Confidentiality versus reliability versus integrity constraints

Inference control and multilevel databases | Exam focus

- Inference control
  - Permitted outputs plus knowledge reveal secrets
  - Two-person salary total minus known salary
  - Blocked direct SELECT still leaks indirectly
  - Restrict small groups and overlapping queries
  - Track query history

- Coarsen/suppress results
- Carefully designed noise
- Utility-leakage tradeoffs

• Multilevel database security
- Different item labels; mandatory policy
- Subject clearances; classifications; compartments
- Bell-LaPadula-style read/write restrictions
- Prevent direct downward confidentiality flow
- Trusted declassification: separately controlled action
- Views alone ≠ multilevel isolation proof
- Covert channels; query results; error messages
- Assess direct access and indirect inference

Policy, incident response and recovery objectives

• Policy versus procedure
- Policy: responsibilities and rules
- Procedure: operational steps
- Coordinate protection; reduce risk
- Acceptable use; access review
- Incident handling; recovery
- Policy ≠ merely passive system use
- Awareness: reporting and independent verification
- Enforcement and usable procedures

• Incident response and recovery objectives
- Prepare roles/tools
- Detect/analyze → contain → eradicate
- Restore verified service; learn from evidence
- Preserve useful protected logs
- Disaster recovery: restore IT
- Business continuity: sustain business functions
- Workarounds and alternate facilities
- RTO: recovery-time target
- RPO: tolerable lost-data age

- Backup schedule meets recovery point
- Restore tests demonstrate recoverability

Interpret security mechanisms and conditions precisely

• Flawed security-question interpretation
- Overlapping/missing mechanisms: no invented unique answer
- Asymmetric question: multiple wrong key directions
- Signature ≠ generic private-key whole-message encryption
- Body biometric ≠ three factors
- SSL/SSH both describe protected channels
- Remote-shell scope distinguishes SSH
- Malformed option labels unrecoverable from theory

• Missing distinctions in security questions
- Arbitrary code execution: multiple mechanisms
- Execution consequence ≠ uniquely buffer overflow
- Database requirements: availability and confidentiality
- ACL/RBAC both restrict unauthorized writes
- Neither alone enforces all integrity constraints
- Evaluation organizations depend on national scheme
- CCRA: recognition, not evaluator
- Supported answer requires missing distinction

Network and System Administration

Revision Outlines

Roles, principles and change discipline | Exam focus

- Administrative roles and qualities
  - Systems: hosts/accounts/storage/software/availability
  - Networks: connectivity/addressing/routes/devices/performance
  - Security: controls and policy
  - Overlapping account/patch/training duties
  - Job titles ≠ exclusive duties
  - Patience, communication, ethics
  - Technical understanding and organization
  - Beyond command knowledge
- Administration and measured change
  - Documentation, standardization, least privilege, automation
  - Record current state and intended change
  - Rollback and validation
  - Maintenance: service impact and urgency
  - Calendar date alone insufficient
  - Fault management: restore failures
  - Performance: measure service behavior
  - Configuration: track settings
  - Security: control access
  - Accounting: track usage

Network foundations for administrators

- Client/server and peer roles
  - Client: requests; server: provides
  - Peer-to-peer: both roles
  - Server may request DNS/backups
  - Dedicated server ≠ never client
  - Network OS ≠ dedicated-server-only

- Clients require networking support
- Centralized service software versus participant OS

• Forwarding, transport and diagrams
- Switching: LAN frames by MAC
- Routing: inter-network IP packets
- TCP: ordered reliable byte stream
- UDP: datagrams, no equivalent guarantee
- Ports: application endpoints, not connectors
- Diagram switches, links, subnets, gateways
- Service hosts and dependencies
- Topology shape ≠ security guarantee

Addresses, prefixes, gateways and loopback

• IPv4 addressing rules
- IPv4: 32 bits
- CIDR prefix determines network
- Network=address AND mask
- Remote subnet: route/gateway
- Private: 10/8,172.16/12,192.168/16
- Private ≠ directly Internet-routable
- IP/class label alone ≠ subnet

• Loopback meaning and test limits
- 127.0.0.1: conventional local IPv4 loopback
- 127/8 reserved
- Tests local protocol processing
- No cable/switch/remote reachability proof
- Local service use
- Multiple loopback descriptions may fit
- Multiple 127/8 choices may fit

• Worked example Subnet-boundary example
- 192.168.2.70/26
- Block size: 64 addresses
- Network .64; broadcast .127

- Ordinary hosts .65-.126
- Gateway requires appropriate reachable route
- NAT ≠ missing-route repair

DHCP exchange and lease lifecycle | Exam focus

- DHCP initial allocation
  - Addresses and configuration options
  - Discover→Offer→Request→ACK
  - Scope/pool and allocated leases
  - Exclusions and reservations
  - Mask, gateway, DNS options
  - Dynamic lease: specified interval
  - Lease ≠ forever
- Lease renewal and reuse
  - Renew before expiry
  - Failed renewal: later rebinding
  - Disconnected host ≠ perpetual renewal
  - Expiry: address eligible for reuse
  - Server-policy conditions
  - Reconnection: validate/renew/new allocation
  - Expiry alone ≠ persistent failure
  - Reservation: known client, predictable address
  - DHCP configuration retained
- Worked example Expired disconnected-client example
  - Expiry while other clients allocate
  - Old address may be reassigned
  - Validate configuration before reuse
  - No unvalidated old-address resumption

DHCP deployment and diagnosis | Exam focus

- DHCP deployment rules
  - Subnet-matching scope
  - Exclude manual server/router addresses

- Correct configuration options
- Relay: broadcasts across router
- Static and DHCP coexist
- Avoid address conflicts
- Less repetitive administration
- Availability/security: deployment-dependent
- Rogue-reply protection
- Location change ≠ guaranteed connectivity

• DHCP returning-client diagnosis
- Inspect current lease/address
- Server/relay reachability
- Free pool capacity and logs
- New devices: possible pool exhaustion
- Exhaustion requires evidence
- Link-local fallback: 169.254.2.33 example
- Same-link communication
- Not ordinary routed Internet access
- 0.0.0.0: unspecified
- 255.255.255.255: limited broadcast

DNS resolution and diagnostic evidence | Exam focus

• DNS record roles
- Forward: name→address
- A: IPv4; AAAA: IPv6
- Reverse: address→name
- PTR: reverse namespace
- CNAME: alias
- MX: mail exchangers
- NS: authoritative name servers
- Forward/reverse: independent existence
- Not necessarily perfect inverses

• DNS caching and diagnosis
- Resolver caching: TTL

- nslookup name
- Explicit DNS-server query
- Configured resolver versus authoritative answer
- Address ping succeeds, lookup fails
- Naming/configuration trouble suggested
- Ping by name: resolved address possible
- nslookup: direct DNS diagnostic
- Resolving includes forward resolution

Commands and observable state | Exam focus

- Address-configuration commands
  - Windows ipconfig: address/mask/gateway
  - ipconfig /all: complete adapters
  - Physical/MAC, DHCP, DNS information
  - /renew: request renewal
  - /release: dynamic address release
  - /flushdns: client DNS cache clearing
  - Linux ip addr: addresses
  - Linux ip route: routes
  - Persistent configuration: distribution-specific
- Connection and path diagnostics
  - netstat: endpoints/protocols/TCP states
  - netstat -ano: numeric addresses/ports/PIDs
  - Literal netstart ≠ netstat
  - Ping: IP reachability probes
  - traceroute/tracert: responding hops
  - TTL/hop-limit expiry
  - ICMP filtering: replies may fail
  - Missing reply ≠ service-down proof
  - nslookup: DNS inspection

Repeatable troubleshooting procedure | Exam focus

- Evidence-led troubleshooting sequence

- Symptom, scope, time, recent changes
- Configuration, baseline, logs
- Testable hypothesis
- Least disruptive explanation first
- Isolate failing component
- Targeted fix; verify user service
- Document results
- One relevant factor at a time
- Reboot may hide evidence
- Reboot ≠ diagnosis

• Troubleshooting approaches
- Bottom-up: physical link first
- Top-down: application errors first
- Divide-and-conquer: test middle layer
- Narrow working/failing boundary
- Divide-and-conquer ≠ random trial-and-error
- Reboot/update/reset/scan: context-dependent actions
- No fault evidence: no unique ranking

• Worked example Scope-based diagnosis examples
- One client fails: inspect client
- Link, address and policy first
- All name lookups fail
- Server IPs still reachable
- Inspect shared DNS/configuration first
- Cable replacement: unsupported by naming-only failure

Connectivity diagnosis by layers | Exam focus

• Layered connectivity checklist
- Power, cable/connector, link state
- NIC and driver state
- VLAN membership
- Address/prefix and local gateway
- Route, DNS, application port/authentication

- Recognized NIC: driver absence less likely
- Recognition ≠ working physical link
- Ethernet: commonly RJ45
- RJ11 telephone jack ≠ expected Ethernet

• Interpreting connectivity symptoms
- Local works, remote fails
- Inspect gateway/routes/filtering
- Application works, ping fails
- ICMP filtering plausible
- Port refused: destination reachable enough
- Inspect service/listener policy
- Timeout: loss/filtering/down host possible
- Timeout requires further evidence

• Worked example Post-replacement NIC example
- Recognized/enabled adapter; no link light
- Try known-good Ethernet cable
- Try known-good switch port
- Recognition ≠ one defective-cable diagnosis

Service catalog and remote access | Exam focus

• Service protocol roles
- DNS: names; DHCP: host configuration
- SMTP: mail send/relay
- IMAP/POP: mail retrieval
- FTP: named file-transfer protocol
- HTTP also transfers files
- File transfer ≠ exclusive FTP description
- HTTP/HTTPS: web resources
- SSH: encrypted remote login
- SFTP: file transfer over SSH
- SFTP ≠ FTP/FTPS

• VPN, media and service administration
- VPN: private connectivity over another network

- Secure tunnel: encryption/authentication
- Protocol/configuration conditions
- RTP: real-time timing/sequence information
- Often over UDP
- RTP ≠ timely-delivery guarantee
- Network printers: shared services
- Service accounts, permissions, configuration
- Monitoring and recovery

Routing, NAT and port translation

- Routing versus NAT/PAT
  - Routing: destination-prefix next-hop selection
  - NAT: address rewriting
  - PAT/NAPT: many-to-one address-and-port translation
  - Distinct bindings on one public address
  - Two private sockets→ports 40001,40002
  - Replies translated to correct clients
- NAT scope and limits
  - Private IPv4→public networks: common use
  - NAT ≠ gateway/route/firewall/authentication replacement
  - One-to-one NAT
  - Destination translation
  - NAT ≠ always many-to-one
  - Public LAN addresses: NAT not automatic
  - Unsolicited inbound service: explicit mapping possible
  - Filtering requirements

Proxies, gateways and QoS

- Proxy and gateway functions
  - Proxy: client/backend representative
  - Cache, filter, mediate application requests
  - Default IP gateway: next-hop router
  - Application gateway: possible protocol translation

- Gateway ≠ mandatory modem/router combination
- Functions may coexist on one machine

• QoS and mixed-category questions
- QoS: classify, queue, prioritize traffic
- Low-delay/limited-loss objectives
- Operates across layers
- Supports applications
- QoS ≠ exclusively application-layer service
- New-service installation: administrative action
- Email configuration: administrative action
- Proxies/agents: components
- Mixed taxonomy: unique exclusion needs scope

Workgroups, domains and directories | Exam focus

• Workgroup versus domain
- Workgroup: per-machine local accounts
- Local credentials and per-host sharing
- Domain: centralized identities/policies
- Directory infrastructure
- Administrative arrangement ≠ physical topology
- Recommended size ≠ hard protocol limit
- HomeGroup: historical Windows sharing
- HomeGroup ≠ AD domain

• Domain controllers and availability
- Controller: AD DS
- Domain authentication and directory replication
- Redundancy: dependencies/failure domains matter
- Two virtual controllers, one physical host
- Shared host failure ≠ independent availability
- Provisioning: addressing, DNS, time
- Supported OS/hardware
- Version minima ≠ universal architecture rules

LDAP and secure directory access

- LDAP operations and transport protection
  - Entries/attributes: distinguished names
  - Bind: session authentication
  - Search: matching entries
  - Compare: attribute-value test
  - Add/delete/modify: entry changes
  - Modify-DN: rename/move
  - Unbind: end session
  - StartTLS: upgrade connection where supported
  - LDAPS: TLS from setup
  - Protect credentials; verify server identity
- Directory ports and security caveats
  - LDAP: commonly TCP 389
  - LDAPS: commonly TCP 636
  - Acronym/port ≠ secure authentication
  - AD also uses DNS/Kerberos
  - LDAP ≠ entire directory product
  - LDAPS-retired-with-LDAPv2 claim: unreliable
  - Modern TLS-protected LDAP distinction

AD structure and delegation

- Directory objects and logical hierarchy
  - Objects: users/computers/groups/resources
  - Schema-defined attributes
  - Security principals: SIDs
  - Display name ≠ sole security identity
  - Domain: objects plus replicated database
  - Tree: contiguous DNS namespace domains
  - Forest: grouped trees
  - Shared schema/configuration
  - Global-catalog framework
- OUs, groups and schema restrictions

- OUs: object organization
- Delegation and Group Policy scopes
- OUs ≠ separate forests/security boundaries
- Groups: access assignment
- OUs: policy/delegation structure
- Schema changes: broad propagation, care
- Ordinary objects: deletion possible
- Schema-definition restrictions: special scope
- Schema restrictions ≠ all object restrictions

Sites, replication and trusts

• AD sites and replication
  - Sites: grouped IP subnets
  - Network locality model
  - Controller selection
  - Replication traffic management
  - Sites ≠ domains/OUs
  - Replicate changes among controllers
  - Intrasite notification-triggered pulls
  - Intersite schedules/link costs
  - Replication ≠ prior-good-state backup
• Trust direction and permissions
  - A trusts B
  - B identities authenticate for A resources
  - Permissions still required
  - Trust ≠ automatic file access
  - One/two-way direction
  - Transitive/nontransitive extension
  - Direction and transitivity: independent
  - Explicit trust ≠ fixed direction/transitivity
  - Trust type/configuration determines behavior
  - State direction before drawing arrows

Accounts, rights and password diagnosis | Exam focus

- Account lifecycle and access models
  - Named accounts
  - Only needed groups/rights
  - Disable departed users
  - Audit privileged activity
  - Authentication: verify account
  - Authorization: permitted operations
  - User right: remote logon
  - File permission: object-specific
  - Groups easier to review than individual grants
- Single-user password-change diagnosis
  - Inspect cannot-change-password account setting
  - Relevant policy/permission
  - Minimum age, password rules, account state
  - Plausible cause ≠ proven conclusion
  - One failure ≠ infrastructure-wide diagnosis
  - Administrators membership ≠ inherent change prohibition
  - Privileged reset ≠ user-initiated change

Windows permissions and effective access | Exam focus

- Windows permission evaluation
  - Read, write, read/execute
  - Modify and full control
  - Granular rights: delete
  - Inherited versus explicit entries
  - Group membership and deny entries
  - Object scope
  - Effective-access view: real account
  - Informal permission-label addition insufficient
  - Network access: share plus filesystem constraints
- Rights and access distinctions
  - Delete: valid Windows permission

- OS-unspecified Delete exclusion: false
- Modify ≠ full control
- Full control: permission/ownership capabilities
- Data access ≠ logon rights
- Read ≠ write
- Read ≠ parent-folder administration

Unix rwx and octal permissions | Exam focus

- Unix permission interpretation
  - Nine bits: owner/group/other triplets
  - Read=4; write=2; execute=1
  - Sum enabled bits per octal digit
  - Leading ls -l character: file type
  - Directory read: list names
  - Directory execute: traverse/search
  - Directory write: change entries
  - Other checks apply
  - Deletion: usually parent-directory permissions
  - Sticky-bit/ACL rules possible
- Worked example Octal and damaged-string examples
  - -rwxr--r--: ordinary file
  - Owner7/group4/other4 → 744
  - 755 → rwxr-xr-x
  - Damaged bank string: -rwxr-r-
  - Missing hyphens: no guessing
  - Confirm all nine bits before choice

Linux paths, filesystems and checks

- Linux directory roles
  - /: filesystem root
  - /root: administrator home
  - /etc: configuration
  - /home: ordinary user homes

- /var: variable data/logs
- /dev: device nodes
- /proc: kernel/process information
- /boot: boot material
- /tmp: temporary files
- Cleanup/mount layouts vary
- lost+found ≠ every partition guarantee
- Reboot ≠ every temporary directory erased

• Filesystems, checking and mounting
- ext2: Linux filesystem
- ext4: later filesystem
- HPFS: historical OS/2 association
- Swap: backing space, not ordinary filesystem
- fsck: filesystem-specific checks/optional repairs
- fsck ≠ init/halt/inittab
- Supported offline/safe repair procedure
- Mount: filesystem attached to directory
- Storage administration: partitioning/mounts/quotas/capacity

• Filesystem origin versus driver support
- Foreign filesystem access possible
- HPFS access: available driver required
- Origin ≠ Linux support
- Swap: paging area
- Native Linux filesystem: ext2
- Native-Linux scope must be explicit
- Broad Linux-use wording: ext2/HPFS ambiguity

Profiles, registry and persistent configuration

• User profile variants
- Local: settings on one client
- Roaming: server synchronization at logon/logoff
- Supported user settings/data
- Large profiles: slower logon/logoff

- Folder redirection: avoid repeated folder copying
- Central profile backup: policy-dependent
- Cached profile ≠ independent local security account

• Windows registry
- Hierarchical configuration database
- Keys, subkeys, typed values
- HKLM: machine settings
- HKCU: current-user view
- Registry ACLs
- Historical root handles: version-dependent
- Record/export relevant configuration before change
- Authorized change; verify service behavior

Performance monitoring and bottlenecks

• Performance diagnosis before hardware changes
- Baseline and interval sampling
- Workload comparison and symptom correlation
- CPU utilization/run queues: processing pressure
- Low free memory: weak evidence alone
- Reclaimable cache
- Persistent paging plus latency: memory pressure
- Storage capacity ≠ I/O latency
- Nearly empty disk: saturation possible
- Nearly full disk: light load possible

• Network measurements and collection methods
- Interface bytes/second
- Errors, drops, delay/loss
- Utilization versus link rate
- Capacity: maximum available rate
- Throughput: observed rate
- Logs: security and performance evidence
- Counters: measured events
- System calls/APIs/log parsers: collection routes

- Methods need not be exclusive

- Worked example Performance calculation and diagnosis examples
  - 80 MB/10 s=8 MB/s
  - Approximately 64 Mb/s before overhead
  - CPU 95% plus growing runnable queue
  - Suggested CPU pressure
  - Disk 95%-full: capacity action
  - Low latency ≠ sufficient free capacity

Monitoring, capture and assessment tools

- Monitoring-tool capabilities
  - Task Manager: processes/resource use
  - Resource Monitor/Performance Monitor: detailed counters
  - Wireshark: packet capture/dissection
  - Nmap: host/port/service discovery
  - Nessus: vulnerability/configuration assessment
  - Snort: detection and alerts
  - Snort inline: prevention possible
  - Snort: sniffing/logging roles
  - Packet capture: not exclusive
  - Wireshark: conventional analyzer choice
- Detection, prevention and evidence limits
  - IDS: detect/alert
  - IPS: inline blocking
  - Scan: potential weaknesses
  - Scan ≠ proven exploitation
  - Packet/open port/alert: context required
  - Usage monitoring ≠ access control
  - Usage monitoring ≠ encryption

Backups, restore and continuity | Exam focus

- Backup purpose and independence
  - Recover previous data

- Deletion, compromise, corruption
- Failed upgrades/migrations; hardware loss
- Authorization: current access, different purpose
- Mirror/replica copies deletions/corruption
- Replica ≠ versioned independent backup
- Protected separate failure domains
- Retention and restore tests

• Backup types and recovery objectives
- Full: chosen dataset
- Incremental: since last backup
- Differential: since last full
- Incremental restore: full plus subsequent incrementals
- Differential restore: full plus latest differential
- RPO: tolerable lost-data time
- RTO: tolerable restoration time
- Compression: reduced storage
- Scheduling: operation timing
- Restore testing: recovery usability

RAID layouts, capacity and selection

• RAID layout mechanisms
- RAID0: striping, capacity/throughput
- RAID0: no disk-failure tolerance
- RAID1: mirroring
- Two-disk mirror: one disk's usable capacity
- Two-disk mirror: survives one disk failure
- RAID2: bit striping/Hamming redundancy
- RAID3: byte striping/dedicated parity
- RAID4: block striping/dedicated parity
- RAID5: block striping/distributed parity
- RAID6: distributed dual parity
- RAID6: two disk failures tolerated
- RAID10: striped mirrored pairs

- RAID10 survival: failed-disk placement dependent

• RAID capacity and selection rules
  - N equal disks, each capacity D
  - RAID0=ND
  - Paired RAID1/10≈ND/2
  - RAID5=(N−1)D
  - RAID6=(N−2)D
  - Hardware controller resources/cache: possible offload
  - Hardware ≠ universally faster software RAID
  - Rebuild bandwidth and parity-write overhead
  - University/bank size ≠ level selection
  - Striping/parity ≠ uniquely RAID5
  - Distributed block parity: RAID5 distinction

• Worked example Four-disk capacity example
  - Four 2 TB disks
  - RAID0=8 TB; RAID10=4 TB
  - RAID5=6 TB; RAID6=4 TB
  - Metadata ignored
  - Failure tolerance, workload, capacity, recovery
  - RAID layouts ≠ backups

Security policy development and purpose | Exam focus

• Security-policy lifecycle
  - Required behavior, scope, accountability, enforcement
  - Assets and service objectives
  - Assess threats and weaknesses
  - Choose controls
  - Exceptions and incident procedures
  - Approve and communicate
  - Monitor and periodic review
  - CIA: confidentiality, integrity, availability
  - No single universal primary objective

• Policy requirements and implementation

- Password controls and approved encryption
- Restricted administrative access
- Backups and patch deadlines
- Firewall rules: partial implementation
- Unrestricted wireless conflicts with internal control
- Isolated intentional guest service: exception
- Passive system use ≠ policy purpose
- Coordinated action and reduced risk
- Policy ≠ all attacks fail

Hardening, controls and vulnerabilities | Exam focus

• Hardening and risk
  - Remove unnecessary services
  - Supported patches
  - Restricted accounts/permissions
  - Appropriate port filtering
  - Unused known-port blocking: possible hardening
  - Blocking useful ports: availability destroyed
  - Weak passwords/unpatched exploitable software: vulnerabilities
  - Attacks exploit weaknesses
  - Risk: likelihood with impact
• Control functions and patch discipline
  - Preventive: restrict harmful actions
  - Detective: expose harmful actions
  - Recovery: restore service
  - Accounts/backups/configuration/traffic analysis: possible controls
  - Stated objective determines control relevance
  - Vulnerability scan: weakness patterns, not bandwidth
  - Test/stage patches
  - Prepare rollback; deploy; verify
  - Testing/installation: both administrator duties

Encryption, hashes and script secrets

- Cryptographic purpose distinctions
  - Encryption: confidentiality
  - Correct key/nonce use
  - Authenticated encryption: integrity/authenticity too
  - Unspecified encryption ≠ both guarantees
  - Password hash: one-way verification
  - Hash ≠ reversible credential store
  - Recoverable credential encryption: protected separate key
- Script-secret handling
  - No reusable secrets in source/logs
  - Protected credential service
  - Restricted runtime source
  - Environment variables/configuration files: storage/transport
  - Neither automatically encrypted
  - Service authentication: actual credential/token often required
  - One-way hash ≠ usable replacement
  - Verification versus recoverable service credentials

Incidents, evidence and reports | Exam focus

- Incident-response lifecycle and reporting
  - Prepare; detect/analyze
  - Contain; remove cause
  - Restore service; learn
  - Investigation: reactive response
  - Patching/backups: normally preventive/preparatory
  - Timestamps and affected systems
  - Evidence, impact, actions
  - Responsible people and recovery results
  - Report may include policy violations
  - Incident scope broader than violations
- Attacks and evidence handling
  - DoS: availability harm

- DDoS: distributed sources
- Phishing: user deception
- Brute force: credential attempts
- Interception: communications read/altered
- Correlate logs by time/identity/context
- Preserve evidence before resets
- Suspicious event ≠ confirmed compromise
- Record facts and uncertainty

Segmentation, NAC and remote administration | Exam focus

- Segmentation and admission control
  - Separate system groups
  - Routing/filtering: allowed flows
  - VLAN membership ≠ complete security
  - NAC: device/user admission assessment
  - Reject/quarantine unauthorized/noncompliant devices
  - Authentication: prove identity
  - Access rules: allowed resources
- Protected remote administration
  - SSH/secured management services
  - Authentication and protection
  - Restricted source access; auditing
  - VPN: defined protected-connectivity model
  - VPN ≠ packet analyzer
  - Firewall: policy-based permit/block
  - Firewall filtering ≠ inherent encryption
  - Management route/firewall changes: connectivity-loss risk
  - Plan for remote lockout

Scripting inputs and reusable operations | Exam focus

- Administrative script building blocks
  - Repeatable automation
  - Scripts ≠ inherently faster/secure

- Python, shells, PowerShell
- HTML: markup
- Define inputs and outputs
- Variables: state
- if/elif/else: decisions
- Loops: repetition
- Functions: reuse
- Python indentation: blocks
- ==: comparison; =: assignment

• Predictable and safe reruns
- Compute desired state
- Compare current state
- Change only differences
- Small dry run before mutation
- Read-only reporting: avoid unnecessary privilege
- Report failures
- Meaningful exit status
- No unconditional success output

• Worked example Threshold-function trace
- def warn_percent(values, limit)
- Loop unpacking: name,percent
- Condition: percent>=limit
- Inputs: root=92,data=40,limit=90
- Only root meets threshold
- print: space-separated output
- root high 92

Python subprocess command launching | Exam focus

• subprocess module and run contract
- Python module ≠ universal shell command
- run: child launch and result
- capture_output: stdout/stderr collection
- text=True: text decoding

- timeout: bounded wait
- check=True: nonzero exit raises
- Argument list: avoid shell interpretation
- List arguments versus shell-string quoting

• Command versus language abstraction
- Echo: print
- grep: text search
- ipconfig/ifconfig: network reporting/configuration
- Neither general Python child-process abstraction
- Generic command question: language required

• Worked example subprocess example and failures
- subprocess.run: ip addr show
- text=True; capture_output=True
- check=True; timeout=5
- print(result.stdout)
- Linux; iproute2 installed
- Missing program: OSError
- Nonzero status: CalledProcessError
- Timeout: TimeoutExpired

File I/O, parsing and updates | Exam focus

• Configuration-processing tools
- File I/O: obtain/persist configuration
- String operations: parse values
- Regex: recognize patterns
- Broad useful-concept question: multiple answers
- JSON/INI: format-specific parser
- Avoid blind global substring replacement
- Explicit text encoding
- with: closure even after failure

• Configuration-update sequence
- Validate desired output
- Preserve known-good copy

- Temporary file: same filesystem
- Atomic target replacement where supported
- Check permissions and syntax
- Reload affected service
- Verify; retain rollback
- Repeated append ≠ idempotent

• Worked example JSON validation example
- with open: settings.json, encoding=utf-8
- json.load(handle)
- port=config["port"]
- isinstance(port,int) required
- 1<=port<=65535
- Invalid type/range: ValueError
- Output: validated port and value
- Reporting only; no system-file change

Arguments and targeted error handling | Exam focus

• Command-line parsing
- Named typed values and usage
- Python argparse
- Positional arguments: required inputs
- Flags: options
- Validate ranges and paths
- Unchecked input ≠ executable shell text

• Worked example Argument/error-handling trace
- ArgumentParser and parse_args
- logfile: Path
- --limit: int, default 10
- limit<1 → p.error
- read_text: UTF-8
- [:limit]: characters, not lines
- Catch OSError/UnicodeError
- Exception skips remaining try

- p.exit(1): read-failed message
- Specific catches expose unexpected programming errors

Regex analysis and log rotation | Exam focus

- Regex contracts and log rotation
  - Text patterns, not encryption/compression/rotation
  - search: match anywhere
  - fullmatch: entire string
  - Raw strings: backslash-heavy patterns
  - Capture needed fields only
  - Explicit unmatched/malformed-line handling
  - Rotation: bounded size, historical segments
  - Parsing and rotation: complementary
- Log-analysis evidence
  - Count requested event
  - Correlate timestamps and hosts
  - No matches ≠ unreadable log
  - Security and performance uses
  - Neither purpose universally exclusive
- Worked example Regex status-count example
  - re.compile: status=(\d{3})\b
  - Three digits and word boundary
  - pattern.search per line
  - if match: group(1)
  - counts.get(code,0)+1
  - 200 and 503: one each
  - Noise deliberately ignored

CPU and memory measurements | Exam focus

- Interval CPU measurement
  - Cumulative counters: two samples required
  - Lifetime total ≠ current utilization
  - Linux /proc/stat: CPU categories

- Define idle categories
- Avoid guest-field double counting
- Busy_fraction=1−Δidle/Δtotal
- Chosen nonbusy: idle+iowait
- Iowait interpretation limitations

• Available-memory measurement
- /proc/meminfo: MemAvailable
- Estimate without swapping
- MemFree excludes reclaimable memory
- Windows Performance Counters/OS APIs
- Analogous metrics, different interfaces
- System calls/counters/log parsing: complementary
- Only-technique claim requires platform/API

• Worked example Measurement calculation example
- Δtotal=1000 ticks; Δidle=250
- Busy≈75%
- MemTotal=8192 MiB; MemAvailable=2048 MiB
- Approximate unavailable/used=75%
- MemAvailable ≠ exact-allocation success guarantee

Backup scheduling and restore verification | Exam focus

• Recurring-job scheduling
- Unix: cron/systemd timers
- Windows: Task Scheduler
- User crontab: minute,hour,day-of-month,month,day-of-week,command
- Absolute paths and known environment
- Protected inputs and logs
- Bounded runtime
- Lock unsafe overlapping jobs
- Interactive PATH/CWD/credentials: not assumed

• Backup verification and tool purposes
- Compression: data packaging
- Scheduling: operation timing

- Check exit status and archive contents
- Restore sample into isolated destination
- Command success ≠ usable backup proof
- Script version control: review aid
- Version control ≠ data backup

• Worked example Cron entry interpretation
- 0 2 * * *
- Daily 02:00, scheduler timezone
- Runs as crontab owner
- /usr/bin/python3 /opt/backup.py /srv /bak
- Source/destination arguments
- >>/var/log/bak.log: append output
- 2>&1: stderr joins stdout
- Command kept on one line
- Paths/permissions/logging/program: preconfigured
- User crontab: no username field
- System crontab: username possible

Constructing a bounded backup script | Exam focus

• Backup-script structure and limitations
- Separate parsing, preconditions, archive creation, reporting
- Source directory required
- Destination outside source
- Unique output names
- Failed partial ≠ completed backup
- Application consistency and capacity checks
- Retention and independent protected copies
- Teaching script: directory packaging
- No silent scheduled-job installation

• Worked example Backup-script trace and verification
- argparse source/destination: Path
- resolve(): absolute source/destination
- is_dir(): existing directories required

- dst==src or src in dst.parents
- Reject destination inside/equal source
- UTC timestamp name; microseconds
- shutil.make_archive: gztar, root_dir=src
- OSError: remove partial .tar.gz
- unlink(missing_ok=True)
- p.exit(1): failure report
- print completed archive path
- Verify archive members
- Restore into different empty directory
- Compare selected files

Automata and Complexity Theory

Revision Outlines

Symbols, alphabets, words and length | Exam focus

- Alphabet, word and language
  - Σ: finite symbol set
  - String: finite ordered symbol sequence
  - $L \subseteq \Sigma^*$
  - Empty word $\varepsilon=\lambda$; length zero
  - $\varnothing$: no words
  - $\{\varepsilon\}$: one word
  - ε and $\varnothing \neq$ numeric 0
- Length versus word powers
  - Occurrences, not distinct symbols
  - $|caddba|=6$
  - $|caddba|^2=36$
  - $|(caddba)^2|=12$
  - Exponent placement trap
  - Choice repair requires changed expression

Concatenation, powers and reversal

- Concatenation, powers and reversal
  - cddc·ababb=cddcababb
  - Order generally matters
  - $\varepsilon x=x\varepsilon=x$
  - $x^0=\varepsilon$; x^n: n copies
  - $caddba^R=abddac$
  - $(xy)^R=y^Rx^R$
  - $(x^R)^R=x$
- Language concatenation and powers
  - $LM=\{xy:x\in L,y\in M\}$
  - $L^0=\{\varepsilon\}$; $L^{n+1}=L^nL$

- {e,fgh}{c,d}={ec,ed,fghc,fghd}
- fghe absent
- $L^0=\varepsilon$: missing-braces type error
- Case-sensitive symbols
- Uppercase/lowercase transcription requires declared repair

Language Operations and Empty-Word Cases | Exam focus

• Alphabet closures and word counts
- Σ^n: exactly n symbols
- $|\Sigma|=k \Rightarrow |\Sigma^n|=k^n$
- $\Sigma^0=\{\varepsilon\}$
- $\Sigma^*=\bigcup n\geq 0\Sigma^n$
- $\Sigma^+=\bigcup n\geq 1\Sigma^n$
- Nonempty alphabet: $\varepsilon\notin\Sigma^+$
- $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$
- $\{a,b\}^2=\{aa,ab,ba,bb\}$

• Language operations and empty cases
- $L^*=\bigcup n\geq 0L^n$; $L^+=LL^*$
- $\varepsilon\in L \Rightarrow \varepsilon\in L^+$
- Alphabet positive-closure distinction
- $\varnothing^*=\{\varepsilon\}$; $\varnothing^+=\varnothing$
- $\{\varepsilon\}^*=\{\varepsilon\}$
- Union: either set
- Intersection: same word in both
- Intersection ≠ concatenation
- Complement Σ^*-L: fixed alphabet

Grammars generate; automata recognize | Exam focus

• Grammar versus recognizer
- $G=(V,\Sigma,P,S)$
- Finite variables; disjoint terminals
- Productions; start $S\in V$
- Left-side replacement

- L(G)={w∈Σ*:S⇒*w}
- Grammar: language generation
- Automaton: word acceptance/language recognition
- Automata recognize languages, not grammars

• Language equivalence and notation
- Equivalent grammars: equal generated languages
- Rule counts/names irrelevant
- Common word insufficient
- Equal alphabets insufficient
- One-way inclusion insufficient
- RE denotes language, not word set
- Formal examples: precisely specified sets
- Natural-language example trap

Chomsky Hierarchy and Epsilon Conventions | Exam focus

• Chomsky model correspondences
- Type 3 REG ↔ finite automata
- Type 2 CFG ↔ NPDA
- Type 1 CSL ↔ nondeterministic LBA
- Context-sensitive/noncontracting grammars
- Type 0 unrestricted ↔ Turing recognizers
- Unrestricted α→β: variable in α
- Meaningless arbitrary rewrite trap

• Expressiveness hierarchy and ε exception
- REG⊊CFL⊊CSL⊊DECIDABLE⊊RE
- Optional S→ε; S absent from RHS
- Strict noncontracting rules cannot generate ε
- Stronger models include weaker languages
- Capable-family questions: potentially nonunique
- Least sufficient versus exact equivalent family
- Expressiveness ≠ runtime ranking

DFA definition and acceptance | Exam focus

- DFA formal definition
  - $M=(Q,\Sigma,\delta,q_0,F)$
  - Finite nonempty Q; $q_0 \in Q$
  - $F \subseteq Q$, possibly empty
  - Total $\delta: Q \times \Sigma \to Q$
  - One successor per state/symbol pair
  - One-transition-machine misconception
  - Empty F recognizes $\varnothing$
  - State names ≠ acceptance status
- DFA acceptance and memory
  - Entire input consumed
  - Ending state in F
  - Earlier accepting visit insufficient
  - ε accepted iff $q_0 \in F$
  - Rejecting sink: all-symbol self-loops
  - Sink still finishes finite input
  - Finite state-encoded memory
  - Finite memory ≠ no memory

Finite-State Recognizer Design and Traces | Exam focus

- Suffix-invariant DFA design
  - Binary ending 01
  - q_0: no useful suffix
  - q_1: suffix 0; q_2: suffix 01
  - Only q_2 accepts
  - (0,1): $q_0 \to (q_1, q_0)$
  - (0,1): $q_1 \to (q_1, q_2)$
  - (0,1): $q_2 \to (q_1, q_0)$
  - 10101: $q_0, q_0, q_1, q_2, q_1, q_2$
  - Final q_2: accepted
- Length-at-least-two DFA
  - States: 0, 1, ≥2 symbols

- Every symbol advances count stage
- Third state: accepting loop
- q_2 naming arbitrary
- Missing diagram/meanings: no numbered answer
- Test ε and shortest boundary words
- Test overlaps
- Sample agreement ≠ proof

Minimum States for Prefixes and Suffixes | Exam focus

- Fixed-suffix DFA minimum
  - Valid word length n: n+1 states
  - Longest suffix matching pattern prefix
  - Overlapping-prefix fallback transitions
  - Distinguishable prefixes via continuations
  - Lower-bound proof
  - Unnecessary rejecting-sink trap
- Fixed-prefix DFA minimum and alphabet
  - n matched-prefix stages
  - Accepting continuation and rejecting sink
  - n≥1, mismatches possible: n+2
  - Unary alphabet: potentially unreachable sink
  - Pattern symbol outside Σ: empty language
  - Empty-language minimum: one rejecting state
  - Starting abc over {a,b}: one state
  - Five requires alphabet containing a,b,c

NFA and epsilon transitions | Exam focus

- NFA and ε-NFA
  - $\delta: Q \times \Sigma \to P(Q)$
  - Zero, one, many successors
  - ε-NFA domain: $Q \times (\Sigma \cup \{\varepsilon\})$
  - ε moves consume nothing
  - NFA convention varies

- Existential finite accepting path
- Complete word consumed
- ε moves before/after input symbols
- Other paths may get stuck

• NFA versus DFA power
- Both recognize exactly REG
- Nondeterminism: potentially smaller description
- DFA successor as singleton set
- ε ≠ input character
- ε ≠ permission for unread input

Subset construction with epsilon closure | Exam focus

• Subset construction with ε-closure
- ε-closure(T): T plus ε-reachable states
- Start: ε-closure($\{q_0\}$)
- Symbol step: ε-closure($\bigcup_{q \in T} \delta(q,a)$)
- Accepting subset intersects F
- Reachable subsets only
- Empty subset: rejecting sink if reached
- n-state NFA: at most 2^n states
- Exactly-2^n trap

• Subset-construction example
- $\Sigma=\{a\}$; start p; final r
- $\delta(p,a)=\{p,r\}$; $\delta(r,a)=\varnothing$
- Start $\{p\}$
- $a \rightarrow \{p,r\}$; further a self-loop
- Accepted language a^+
- Added ε edge $p \rightarrow r$
- Start already accepting; ε accepted

Equivalence, minimization and diagram checks | Exam focus

• DFA minimization
- Discard unreachable states

- Initial accepting/nonaccepting partition
- Split by transition target blocks
- Repeat until stable
- Stable equivalence classes merged
- Equivalent: every continuation agrees
- Current acceptance alone insufficient

• Equivalence and diagram checks
- Reachable product-state pairs
- Exactly one accepting: distinguishing word
- No disagreement: language equality
- Complete partial diagram with sink
- RE test: ε and shortest cycles
- Accepting start plus ab return
- Equivalent expression must include ε,ab

Regular-expression notation and precedence

• Classical RE syntax and precedence
- ∅; ε; symbols; union; concatenation; star
- Union: A|B or A+B
- Postfix r^+: one-or-more
- Distinct uses of +
- Parenthesize conflicting conventions
- Star > concatenation > union
- r?=r|ε
- r^+=rr*

• RE interpretation examples
- (a|b)*a: ending a
- 0*10*: exactly one 1
- a*(b|c): a's then one b/c
- aab accepted; aaa,aaabc rejected
- (0|ε)(0|1)*: all binary words
- ε alternative permits any suffix
- Programming backreferences may exceed REG

- Wildcard dot: additional notation

Regular Expressions and Representation Conversions | Exam focus

- Build regular expressions
  - At least one a: (a|b)*a(a|b)*
  - Exactly one a: b*ab*
  - At least two zeros: (0|1)*0(0|1)*0(0|1)*
  - Nonadjacent zeros allowed
  - 010 distinguishes from substring 00
  - S→aS|ε generates a*
- RE-automaton conversions
  - RE→ε-NFA fragments
  - Union: parallel alternatives
  - Concatenation: connected fragment ends
  - Star: ε bypass and repetition
  - Then subset construction
  - Automaton→RE: state elimination
  - $r_{ij} \mid r_{ik}(r_{kk})^*r_{kj}$
  - New start/final nodes when required
  - Whole-language preservation

Regular grammars and closure constructions | Exam focus

- Regular grammar orientations
  - Right-linear: A→aB, A→a, permitted ε
  - Left-linear: A→Ba
  - Consistent orientation: REG
  - Arbitrary mixing may be nonregular
  - S→aA|ε, A→Sb: a^nb^n
  - A→aBA: CFG, not regular form
- Regular constructions and closure
  - Variables → NFA states
  - A→aB: a edge

- A→a: fresh final state
- A→ε: accepting A
- Union; concatenation; star; reversal
- Intersection; difference; complement
- Reversal: reverse edges, swap start/finals
- ε-NFA handles reversed starts
- Complement: complete DFA, swap acceptance
- Product states: intersection/difference

Regular pumping lemma: use the quantifiers | Exam focus

• Regular pumping lemma
  - $\exists p \geq 1$; $\forall s \in L$, $|s| \geq p$
  - $s=xyz$; $|xy| \leq p$; $|y| \geq 1$
  - $\forall i \geq 0$: $xy^iz \in L$
  - Nonregular proof: arbitrary p, chosen witness
  - Every allowed split broken
  - One bad split insufficient
  - Necessary condition ≠ regularity proof
• Pumping a^nb^n and infinitude
  - Witness a^pb^p
  - Every split: $y=a^k$, $k \geq 1$
  - i=0: fewer a's than b's
  - Contradiction excludes every DFA
  - n-state DFA: long path repeats state
  - Accepted cycle: indefinite repetition
  - Infinite iff accepted word length ≥n

CFG derivations, sentences and parse trees | Exam focus

• CFG productions and sentential forms
  - $A \to \alpha$; $A \in V$; $\alpha \in (V \cup \Sigma)^*$
  - One variable on left
  - Weaker set-membership options also true
  - Sentential form: any S-derived string

- Includes terminal sentences and S

• Derivations and parse trees
  - Leftmost: leftmost variable replacement
  - Rightmost: rightmost variable replacement
  - Not opposite input-scanning directions
  - Root S; expanded internal variables
  - Terminal leaves; optional ε leaves
  - Left-to-right yield
  - Tree: derivation structure
  - Derivation-graph path: different object

Design and validate context-free grammars | Exam focus

• Matched-structure CFG design
  - Recursive rule plus stopping base
  - $S \to aSb \mid \varepsilon$: $a^n b^n$, $n \geq 0$
  - n recursions: n matched pairs
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
  - $n \geq 1$: base ab replaces ε
• CFG construction and proof
  - $a^n b^m$, $n > m \geq 0$
  - $S \to aSb \mid A$; $A \to aA \mid a$
  - S: matched pairs; A: extra a's
  - Balanced parentheses: $S \to (S)S \mid \varepsilon$
  - Soundness: preserved intended property
  - Completeness: decompose desired words
  - Several examples ≠ language equality

Ambiguity and precedence | Exam focus

• CFG ambiguity
  - One word, two distinct parse trees
  - Equivalent: two distinct leftmost derivations
  - Same-tree expansion orders insufficient
  - $E \to E+E \mid E*E \mid id$

- id+id*id: (id+id)*id versus id+(id*id)

• Precedence and ambiguity distinctions

- E→E+T|T; T→T*F|F; F→id|(E)
- Multiplication before addition
- Left recursion: left association
- Uniqueness: last-operator split and induction
- S→SS|ε: ambiguous ε
- S→A|B; both branches derive ab
- A→ab; B→abB|ε
- Ambiguous grammar ≠ inherently ambiguous language
- Alternative grammar may remove ambiguity

Simplification, CNF and membership | Exam focus

• Grammar simplification

- Null A→ε; unit A→B
- Mark nullable variables
- Omit nullable occurrences through alternatives
- Fresh start preserves ε
- Unit-chain reachability copies nonunit productions
- Remove nongenerating, then unreachable
- Nullable deletion: nonempty-word loss trap

• CNF and CYK membership

- A→BC or A→a
- Optional isolated $S_0 \to \varepsilon$
- Long-rule terminals replaced by variables
- Binary variable-sequence splitting
- $A \to aBC \Rightarrow A \to XC_1$, $X \to a$, $C_1 \to BC$
- CYK: substring variable sets
- Combine splits using A→BC
- Fixed grammar: $O(n^3)$
- n>0: n terminal, n−1 binary rules

Greibach form and left recursion | Exam focus

- Greibach normal form
  - $A \to aA_1 \ldots A_k$, $k \geq 0$
  - Leading terminal, then only variables
  - ε: isolated fresh start
  - Fresh start absent from right sides
  - Length $n>0$: n leftmost rule applications
  - Each step adds one prefix terminal
- GNF conversion choices
  - Remove ε, unit, useless rules
  - Order variables; substitute earlier-leading rules
  - Remove left recursion
  - Substitute until terminal-leading
  - CNF-first: optional algorithm choice
  - $A \to A\alpha \mid \beta \Rightarrow A \to \beta R$, $R \to \alpha R \mid \varepsilon$
  - Further ε normalization
  - No direct or indirect left recursion
  - First-step question requires named algorithm

PDA configuration and acceptance | Exam focus

- PDA structure and transitions
  - $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
  - Finite control plus stack
  - State,input/ε,top → new-state,replacement pairs
  - Stack-top orientation before traces
  - $X \to X$: unchanged stack
  - ε input: no consumption
  - $X \to \varepsilon$: pop
  - Distinct ε roles
- PDA configurations and acceptance
  - State; unread input; stack
  - Final-state mode: consumed input, final state
  - Empty-stack mode: consumed input, empty stack

- NPDA modes: same language family
- Conversion, not same unmodified machine
- DPDA acceptance conventions need care

PDA Construction and Determinism | Exam focus

- PDA for $a^n b^n$
  - Push X per a
  - Switch to b phase
  - Pop X per b
  - End input at bottom marker
  - Reject a after b phase
  - Reject b without X
  - Explicit ε handling for n=0
  - Stack counting ≠ random-access input storage
- Mirror versus copy and determinism
  - ww^R: push, guess midpoint, match pops
  - Binary unmarked midpoint: CFL, not DCFL
  - wcw^R: visible switch, DPDA sufficient
  - Unary even palindromes: even length, REG
  - ww: same-forward-order copy
  - Binary copy language ≠ palindrome language
  - Single-stack comparison limitation

CFL closure and counterexamples | Exam focus

- CFL closure with regular intersection
  - Union; concatenation; star; reversal
  - CFL∩REG: context-free
  - PDA×DFA control; original stack
  - Consumed symbol updates both states
  - ε move leaves DFA unchanged
  - Not necessarily regular
  - $a^n b^n \cap a^* b^* = a^n b^n$
- CFL nonclosure boundaries

- Arbitrary intersection not closed
- $a^n b^n c^m$ and $a^m b^n c^n$: CFL
- Intersection $a^n b^n c^n$: non-CFL
- Complement/arbitrary difference not closed
- CFL−REG: CFL
- Intersect with regular complement
- Failed proposed PDA ≠ impossibility proof

CFL pumping proof and copy versus mirror | Exam focus

- CFL pumping lemma
  - CFL assumption; some pumping length p
  - Every sufficiently long $s \in L$
  - $s=uvxyz$; $|vxy| \le p$; $|vy| \ge 1$
  - $\forall i \ge 0$: $uv^i xy^i z \in L$
  - v and y pumped together
  - Either empty, not both
  - Non-CFL proof: arbitrary pumping length
  - Every allowed decomposition defeated
- Three-count and copy-language boundaries
  - Witness $a^p b^p c^p$
  - Window cannot contain a and c
  - At most two blocks affected
  - At least one count changes
  - i=0 or 2
  - Third count fixed or order broken
  - $a^n b^n c^n$: non-CFL
  - Binary ww: non-CFL
  - ww^R grammar: $S \to aSa \mid bSb \mid \varepsilon$
  - Calling ww palindrome: contradictory

Context-sensitive rules and linear bounded automata | Exam focus

- Context-sensitive and noncontracting rules

- αAβ→αγβ; γ≠ε
- Context-preserving variable replacement
- Complete-side length nondecrease
- Isolated ε exception
- Unrelated $|\beta| \geq |\alpha|$ comparison trap

• LBA and CSL decidability
- Length-n input: O(n) tape cells
- CSL ↔ nondeterministic LBA
- $a^n b^n c^n$: verify a*b*c* order
- Repeated one-a,b,c marking
- Missing/excess symbols rejected
- Marks confined to input region
- m cells: $\leq |Q| m |\Gamma|^m$ configurations
- Finite-graph acceptance reachability
- Looping branches allowed
- Every CSL decidable

Language Classification and Boundaries | Exam focus

• Language-family classification
- a*b* and $a^n b$: REG
- $a^n b^n$: CFL, nonregular
- Binary ww^R: CFL
- Binary ww: non-CFL
- Both mirror/copy: decidable
- $a^n b^n c^n$: CSL, non-CFL, decidable
- Finite languages: regular
- Stronger models recognize lower-family languages

• Unary prime-length boundary
- $\{a^p : p \text{ prime}\}$: nonregular
- Choose prime q>pumping length p
- Pump piece length k≥1
- Pump exponent q+1
- Length q(k+1): composite

- Unary CFL ⇒ regular
- Prime-length language: non-CFL
- Bounded-workspace divisor test: CSL
- Listed examples: none nonrecursive

TM components, transitions and configurations | Exam focus

- Turing machine components
  - $M=(Q,\Sigma,\Gamma,\delta,q_0,q\text{accept},q\text{reject})$
  - Finite state/tape alphabets
  - $\Sigma\subseteq\Gamma$; blank $\square\in\Gamma-\Sigma$
  - Module blank$\notin\Gamma$: wrong
  - Finite control; read/write head; unbounded tape
  - Input: initial tape content
  - Separate input organ/stack unnecessary
- TM transitions and instantaneous descriptions
  - $\delta(q,a)=(p,b,L)$
  - Read a, write b, left, state p
  - Halting states: no moves
  - ID u q av: scans a
  - Left: $u=u'c \Rightarrow u'\ p\ cbv$
  - Right: ub p v
  - Unused tape: show blank
  - Diagram-specific state/transition conventions

TM Design, Variants and Limits | Exam focus

- TM decision design and halting
  - $a^nb^nc^n$: mark triples, reset scans
  - Verify no unmatched symbols
  - Progress: additional marks each round
  - Finite rounds
  - ww: reject odd length
  - Mark/counter midpoint
  - Corresponding-half symbol comparison

- Writable tape beyond single-stack access
- Acceptance test and all-input halting proof

• TM variants and Church-Turing thesis
- Off-line: read-only input, read/write work
- Multitape; multihead; two-way-infinite tape
- Nondeterministic TM: same recognition power
- Different simulation costs
- Two unbounded stacks simulate tape
- Turing completeness: general TM simulation
- No polynomial-speed implication
- Finite runs need finite cells
- Effective procedures: Church-Turing thesis
- Not proved runtime theorem

Deciders, recognizers and complements | Exam focus

• Deciders versus recognizers
- Decider: halts on every input
- Recursive = decidable
- Recognizer: members halt-accept
- Nonmembers: halt-reject or loop
- RE = recognizable = semidecidable
- Loop ≠ rejecting halt
- Undecidable may remain recognizable

• Complements and computable functions
- DECIDABLE ⊊ RE
- L and complement RE ⇒ decidable
- Dovetail recognizers until acceptance
- Decider complement: swap outcomes
- Recognizer final-state flipping fails on loops
- Total computable functions
- Recursive function ≠ recursive self-calls
- Partial functions undefined on nonhalting computations

Halting, diagonalization and Rice’s theorem | Exam focus

- HALT and diagonalization
  - HALT={⟨M,w⟩:M halts on w}
  - Recognizable: simulate, accept any halt
  - Assume halting decider H
  - D(x): loop if H(x,x) predicts halt
  - Otherwise halt
  - D on own description: contradiction
  - HALT: RE, undecidable
  - Complement HALT not RE
  - Both recognizable would yield decider
- Rice’s theorem and limits
  - Nontrivial recognized-language semantic property
  - Some RE languages satisfy, others not
  - Empty language; contains ε; ≥3 words
  - Undecidable language properties
  - State count: syntactic
  - Symbol-writing: machine behavior
  - Nonlanguage properties require separate proof
  - Undecidable: no general decider
  - Currently unknown algorithm ≠ undecidability

Decidable and Undecidable Questions | Exam focus

- Decidable automata/grammar questions
  - DFA membership: simulation
  - NFA membership: reachable sets
  - FA emptiness: reachable final state
  - DFA equivalence: product acceptance disagreement
  - CFG membership: parsing/CYK
  - CFG emptiness: generating-variable marking
  - Halting procedures
  - Arbitrary TM membership distinction
- Undecidable questions and syntax scope

- TM acceptance: RE-complete, undecidable
- HALT: undecidable
- General CFG ambiguity/equivalence: undecidable
- Fixed CFG-word membership: decidable
- Two trees establish particular ambiguity
- No universal ambiguity decider follows
- FORTRAN: syntax versus semantics versus computation
- Language name alone insufficient

Computable reductions for undecidability | Exam focus

- Computable many-one reduction
  - $A \leq_m B$: total computable f
  - $x \in A$ iff $f(x) \in B$
  - B decider plus f decides A
  - Known-undecidable A → target B
  - $B \leq_m A$ alone: no B-hardness proof
- HALT$\leq_m$A_TM construction
  - Given $\langle M,w \rangle$, construct N
  - N ignores input, simulates M(w)
  - Accept on any M halt
  - Rejecting halts included
  - $\langle N,\varepsilon \rangle$ accepted iff M(w) halts
  - Finite-description construction terminates
  - Internal simulation may loop
  - Both yes/no directions
  - Computable mapping, not resemblance

Resources and Input Encoding | Exam focus

- Resource measures and asymptotic bounds
  - Time: computation steps
  - Space: used cells/memory
  - Read-only input inclusion convention
  - Deterministic decider: length-n maximum

- Bounded NTM: every branch bounded
- Big-O: asymptotic upper bound
- Worst/average/best-case functions all eligible
- Θ: matching lower bound

• Growth and encoding size
- $1 < \log n < n$
- $n < n \log n < n^2$
- $n^2 < 2^n < n!$
- Fixed polynomial n^k versus exponential 2^n
- Binary N: Θ(log N) bits
- O(N) not polynomial in binary length
- Pseudopolynomial: numeric magnitude dependence
- Huge polynomial degree: potentially impractical

P, NP and polynomial verification | Exam focus

• P and NP definitions
- Classes of decision languages
- P: deterministic polynomial-time decider
- NP: nondeterministic polynomial-time decider
- Polynomial verifier and polynomial-size certificate
- Yes-input: some accepting certificate
- No-input: no accepting certificate
- NP ≠ non-polynomial

• CLIQUE verification and $P \subseteq NP$
- Certificate: k vertices
- Check distinctness and every pair-edge
- Polynomial in graph encoding
- P verifier ignores certificate
- P=NP: open
- NP ≠ definitely hard/outside P
- Finding/verifying intuition, not exponential-search necessity

Polynomial Reductions and Consequences | Exam focus

- Polynomial reduction consequences
  - $A \leq_p B$: polynomial f
  - $x \in A$ iff $f(x) \in B$
  - Solving B solves A
  - B at least A-hard
  - $B \in P \Rightarrow A \in P$
  - A NP-hard $\Rightarrow$ B NP-hard
  - $A \in P \nRightarrow B \in P$
  - Same-class inference trap
  - Polynomial reductions compose
- Hardness direction and mapping bounds
  - Known-hard Y → new target X
  - X→Y: reversed-proof trap
  - Polynomial mapping time and instance size
  - Undecidability: computable reduction sufficient
  - NP-completeness: arbitrary computability insufficient
  - Explicit reduction and resource bound

NP-Hardness, Completeness and Decision Formulations | Exam focus

- NP-hard versus NP-complete
  - NP-hard: every NP language $\leq_p X$
  - NP-hard need not lie in NP
  - NP-hard may be undecidable
  - NP-complete: NP membership plus NP-hardness
  - NP-complete lies inside NP
  - Finite exhaustive certificates: exponential decision
  - Multiple true source options
  - Exponential necessity unproved
- Canonical complexity examples
  - SAT: satisfying Boolean assignment
  - Cook-Levin: SAT NP-complete

- CNF-SAT and 3-SAT: NP-complete
- TSP decision: tour cost ≤B
- Standard finite encoding: NP-complete
- Shortest-tour optimization: NP-hard
- NP-complete requires decision formulation
- Fixed 2-colorability: P
- Fixed k≥3 colorability: NP-complete

3-SAT to CLIQUE Hardness Proof | Exam focus

- 3-SAT→CLIQUE construction
  - CLIQUE∈NP: pairwise-edge certificate check
  - m clauses
  - Vertex per literal occurrence
  - Edges: different clauses, noncomplementary literals
  - Target clique size m
  - Polynomial construction time and space
- 3-SAT→CLIQUE equivalence and correction
  - Satisfying assignment: true literal per clause
  - Consistent literals form clique
  - m-clique: one occurrence per clause
  - No complementary chosen literals
  - Assign selected literals true
  - Extend remaining variables arbitrarily
  - Both implications
  - Source formula: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_3$
  - (true,true,false) satisfies all clauses
  - ¬false=true: source correction

PSPACE, quantifiers and space reuse | Exam focus

- PSPACE and memory reuse
  - PSPACE=$\cup_k$SPACE(n^k)
  - Deterministic polynomial workspace
  - NP⊆PSPACE: enumerate/check witnesses sequentially

- Exponential candidates, reused memory
- Savitch: NSPACE(s)⊆DSPACE(s^2)
- Appropriate s≥log n
- NPSPACE=PSPACE
- No P=NP implication

• TQBF and quantifier evaluation
- Fully quantified Boolean truth
- TQBF: PSPACE-complete
- ∃x: either branch
- ∀x: both branches
- DFS assignments and recursion state
- Polynomial space, potentially exponential time
- ∀x∃y(x↔y): true
- ∃y∀x(x↔y): false
- Membership plus every-PSPACE-language hardness
- Stated reduction qualifier

Known Hierarchy and Open Separations | Exam focus

• Complexity containment hierarchy
- L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE
- L/NL: logarithmic workspace
- EXPTIME=∪kTIME(2^(n^k))
- Upper-bound class includes P
- Membership ≠ exponential necessity
- Hierarchy: P⊊EXPTIME
- Hierarchy: PSPACE⊊EXPSPACE
- Not all adjacent separations established

• Open separations and collapse limits
- P versus NP: open
- NP versus PSPACE: open
- PSPACE versus EXPTIME: open
- coNP: complements of NP languages
- P⊆NP∩coNP

- NP=coNP: open
- NP-complete in P ⇒ P=NP
- No all-decidable polynomial collapse
- Undecidable remains undecidable
- Cryptography/easiness claims require class and encoding

Unusual Bank Grammar Derivations

- Unusual right-linear grammar invariant
  - S→11S|0A|1A; A→0|1|0S
  - At most one rightmost variable
  - Terminal word still sentential form
  - S⇒11S⇒1111S⇒111111S⇒1111110A⇒11111101
  - Return-to-S pairs: 11,00,10
  - Prefix invariant over length guessing
- Unusual grammar membership repair
  - A→aB|bA|bBa; B→bB|cA|b
  - Start variable required
  - Assume start A
  - Remove required a/b prefix
  - Remove b…a wrapper
  - B: match b, or recurse bB/cA
  - Shorter residual guarantees termination
  - All four offered words fail
  - Marked choice contradicted; no option works

Source Defects and Missing Information

- Formula and diagram prerequisites
  - Exact readable language and alphabet
  - Arbitrary state labels
  - Transition diagram and initial state
  - Accepting markings or state meanings
  - Theory alone cannot identify labels
- Specific source defects

- Accepting start and ab-return paths
- Required ε and ab absent together
- Trace strings from the initial state
- Forward copy ww versus palindrome ww^R
- Capable/contains: least-family or exact-class qualifier
- Most difficult: reduction qualifier
- GNF first step: named algorithm
- Overlapping options: no invented unique key

Compiler Design

Revision Outlines

Translation, Interpretation and Compilation Timing

- Compiler translation contract
  - Source → equivalent target representation
  - Defined observable behavior preserved
  - Machine code; assembly; bytecode; source targets
  - Transpiler: source-to-source compiler
  - Language semantics versus unstated programmer intention
- Interpretation, AOT and JIT
  - Interpreter: executes program representation
  - Line-at-a-time misconception
  - Bytecode compilation then interpretation/JIT
  - AOT: before execution
  - JIT: during execution, runtime profiles
  - Speed: startup, optimization, workload
  - No universally faster method

Tools, host, target and incremental work

- Translation toolchain
  - Preprocessor: inclusion and macros
  - Assembler: assembly → object code
  - Linker: objects and libraries
  - External-symbol resolution and relocation
  - Loader: executable mapping and start
  - Compilation alone ≠ ready executable
- Host, target and compilation organization
  - Cross-compiler: host differs from target
  - Multiple targets: appropriate backends
  - Retargetable: changeable target descriptions
  - Incremental: affected fragments and dependent units

- Incremental ≠ changed-line replacement only
- One/multipass: traversal counts, not targets

Compiler phases and shared services | Exam focus

- Conventional compiler pipeline
  - Lexical → syntax → semantic analysis
  - IR generation → optimization → target generation
  - Scanning: characters → tokens
  - Parsing: structure
  - Semantics: declarations and types
  - Synthesis: implementation code
  - Phase: logical activity
  - Pass: traversal, possibly combined activities
- Compiler organization and services
  - Front end: source-language rules
  - Middle end: IR analysis/transformation
  - Backend: target details
  - Shared symbol tables and error reporting
  - No mandatory seventh final phase
  - Optimization at multiple levels
  - Post-instruction-selection optimization possible
  - Standard six-phase final: code generation

Token, lexeme and pattern

- Token, lexeme and pattern
  - Token: category plus optional attribute
  - (ID,symbol-entry); (INT_LITERAL,42)
  - Lexeme: actual characters total or 42
  - Pattern: token-class membership specification
  - Scanner output: ordered token stream
  - Unordered-set trap
- Token classes and scanner boundaries
  - Keywords; identifiers; literals; operators; punctuation

- Usually skipped whitespace/comments
- Retained line/column positions
- Indentation/newlines may carry tokens
- Parse trees: parser responsibility
- Scanning may intern identifier spellings
- Scope/types: later analysis

• Worked example Five-token declaration
- int x = 42;
- keyword; ID; assignment; integer; semicolon

Regular expressions for lexical structure | Exam focus

• Identifier regular expressions
- Concatenation; alternation r|s
- Kleene r*: zero-or-more
- r+: one-or-more
- Letter(Letter|Digit)*
- Single letter allowed; leading digit forbidden
- C-like ASCII underscore convention
- [A-Za-z_][A-Za-z0-9_]*
- Language-specific identifier rules

• Equivalent patterns and nesting limits
- Letter(Letter+|Digit*)* = Letter(Letter|Digit)*
- Repeated letter/digit blocks
- Equivalent-option trap
- Regular patterns: ordinary tokens
- Arbitrary balanced nesting: grammar/stack

Construct and run a scanner automaton | Exam focus

• Scanner-generator automata pipeline
- RE → Thompson NFA → subset DFA
- Optional minimization
- Concatenation: final-to-start ε edge
- Alternation: branching ε paths

- Star: skip and repetition ε paths

• Scanner subset construction
  - ε-closure: zero-or-more ε reachability
  - DFA state: NFA-state set
  - Transition: ε-closure(move(set,a))
  - Start: NFA start closure
  - Repeated unseen-subset discovery
  - Accepting subset contains NFA final
  - Token identity and rule-priority annotations
  - Empty subset: dead state

Longest match and counting tokens

• Maximal munch and rule priority
  - Longest matching lexeme
  - Equal-length flex matches: earliest rule
  - ifx: identifier, not if+x
  - <= and ++: single operators
  - Identifier recognition then keyword-table lookup

• Loop-header token count
  - for (int i = 0; i<=10; i++)
  - for,(,int,i,=,0,;
  - i,<=,10,;,i,++,)
  - Fourteen tokens irrespective of spacing
  - Increment: one token
  - Comments: no ordinary post-preprocessing parser token
  - Choices 4,5,9,10: correct result omitted

Locate the error by evidence

• Compiler-error classification
  - Lexical: no legal token formation
  - Invalid character; unterminated literal
  - Syntax: illegal token arrangement
  - Missing delimiter

- Semantic: name, scope, type constraints
- Wrong algorithms may compile
- All-logical-errors detection trap

• Malformed fro-line diagnosis
- fro and lowercase o: legal identifiers
- Misspelled keyword ≠ lexical error
- Mismatched syntax: definite
- Undeclared o: possible semantic diagnostic
- Compiler-dependent recovery order and diagnostics
- No silent fro→for or o→0 repair

CFGs, derivations and trees | Exam focus

• CFG vocabulary and derivations
- G=(N,T,P,S)
- Nonterminals; terminals; productions; start
- A→α: one nonterminal rewrite
- Sentential form: any S-derived string
- Sentence: terminals only
- Leftmost/rightmost: corresponding remaining nonterminal

• Parse tree versus AST
- Parse tree: production structure
- Root S; internal nonterminals
- Terminal leaves; optional ε leaves
- Left-to-right yield: input
- AST: removed scaffolding and punctuation
- Operands: leaves; operators: internal nodes
- Parser may construct either

Ambiguity, precedence and associativity | Exam focus

• Grammar ambiguity
- One terminal string, two parse trees
- Equivalent: two leftmost derivations
- Same-tree derivation order insufficient

- E→E+E|E*E|id
- id+id*id: alternative grouping

• Precedence, associativity and dangling else
- E→E+T|T; T→T*F|F; F→(E)|id
- Multiplication precedence; left association
- a−b−c: (a−b)−c
- Dangling else: alternative nested-if attachments
- Nearest-unmatched-if policy or grammar rewrite
- Leftmost derivation alone ≠ disambiguation

Top-down parsing | Exam focus

• Top-down parsing
- Start S → input expansions
- Usually leftmost derivation
- Recursive descent: procedure per nonterminal
- General descent may backtrack
- Predictive: lookahead choice without backtracking
- LL(1): Left-to-right, Leftmost, one lookahead

• Nonrecursive predictive parsing
- Explicit grammar-symbol stack
- Parsing-table consultation
- Same predictive language class
- Recursive/table-driven: linear for suitable fixed grammar
- Names alone do not rank efficiency

Left Recursion and Prefix Factoring | Exam focus

• Left-recursion elimination
- Immediate A→Aα
- A→Aα|β; β not A-leading
- A→βA′; A′→αA′|ε
- Generated language preserved
- Changed tree shape
- Semantic actions preserve intended association

- Indirect cycles through other nonterminals
- Ordinary predictive descent needs removal

• Left factoring and parser suitability
- $A \to \alpha\beta \mid \alpha\gamma \Rightarrow A \to \alpha A'$
- $A' \to \beta \mid \gamma$
- Common-prefix choice postponed
- Factoring ≠ guaranteed unambiguity/LL(1)
- LR handles suitable left recursion
- Unmodified descent: infinite recursion before input

Compute FIRST and FOLLOW | Exam focus

• FIRST of a sequence
- Beginning terminals
- ε when sequence vanishes
- Add each FIRST excluding ε
- Continue across nullable symbols
- ε only if all nullable

• FOLLOW fixed-point computation
- Next terminals after A
- End marker $, never ε
- $\$ \in FOLLOW(S)$
- $B \to \alpha A \beta$: FIRST(β)−{ε} into FOLLOW(A)
- Nullable β: FOLLOW(B) into FOLLOW(A)
- Repeat until no growth

• Worked example FIRST/FOLLOW worked grammar
- $S \to AB$; $A \to a \mid \varepsilon$; $B \to b$
- FIRST(A)={a,ε}
- FIRST(S)={a,b}
- FOLLOW(A)={b}
- FOLLOW(B)={$}

Build and use an LL(1) table | Exam focus

• LL(1) table construction

- A→α entries under FIRST(α) terminals
- Nullable α: entries under FOLLOW(A)
- Multiple productions per cell: conflict
- SELECT=FIRST(α)−{ε}
- Nullable: add FOLLOW(A)
- Pairwise disjoint SELECT sets
- Two nullable alternatives also conflict

• Predictive stack trace
- Initialize $S
- Terminal match and pop
- Nonterminal row plus lookahead
- Push selected RHS reversed
- ε production: no input consumed
- Accept: stack/input both $
- A→a|ε: a-lookahead selects a
- b∈FOLLOW(A): b-lookahead selects ε

• Exam trap LL(1) conflict traps
- Useful ambiguous grammars not LL(1)
- Overlapping FIRST
- FIRST/FOLLOW conflicts
- Left recursion
- A-and-B-only key: insufficient exclusion

Bottom-up actions and handles | Exam focus

• Bottom-up parsing and handles
- Input reduced toward S
- Reverse rightmost derivation
- Handle: next correct RHS occurrence
- Context determines valid reduction
- Arbitrary matching RHS ≠ handle

• Shift-reduce actions and GOTO
- Shift: next token and state
- Reduce: RHS → LHS, then GOTO

- Accept: complete; error: invalid situation
- A→XYZ: pop three symbols/states
- Exposed q → push A,GOTO[q,A]
- Go ≠ separate standard ACTION
- GOTO: essential table machinery

Construct LR(0) item states | Exam focus

- LR(0) augmentation and closure
  - S′→S
  - Item A→α·β: production progress
  - Dot before B triggers B→·γ
  - Repeat CLOSURE additions
- LR(0) canonical collection
  - GOTO(I,X): advance dots, take closure
  - Initial CLOSURE({S′→·S})
  - Nonempty GOTOs for all grammar symbols
  - Iterate until no new states
  - Canonical collection: item-set collection
  - GOTO alone ≠ canonical collection
- Worked example LR(0) state example
  - S→a
  - I0={S′→·S,S→·a}
  - GOTO(I0,a)={S→a·}
  - GOTO(I0,S)={S′→S·}
  - Complete item set: one state
  - One-state-per-item trap

LR families and conflicts | Exam focus

- LR parser families
  - SLR(1): LR(0) states, FOLLOW reductions
  - Canonical LR(1): context-specific item lookaheads
  - LALR(1): merge equal LR(0) cores
  - Combined lookaheads

- Merging may introduce reduce/reduce conflicts
- LR(0)⊆SLR(1)⊆LALR(1)⊆LR(1)
- Increasing capability, often larger tables

• Parser conflicts and limits
- Shift/reduce: consume or reduce
- Reduce/reduce: two reductions
- SLR/LALR conflict ≠ ambiguity proof
- LR(1) handles LR(1) grammars
- Not every unambiguous CFG
- Suitable expression conflicts: precedence/associativity
- Parser renaming ≠ universal ambiguity repair

Parser limits and operator grammars | Exam focus

• Nesting and deterministic parser limits
- LL; LR; recursive descent: suitable nesting
- Explicit/implicit stack supports unbounded nesting
- LL and LR: tables possible
- Deterministic LL/LR: unambiguous grammars
- Ambiguous grammar: rewrite or disambiguation
- Generalized method: GLR

• Operator grammar conditions
- No adjacent RHS nonterminals
- Usually no ε productions
- Operator precedence: conflict-free relations additionally
- Right-linear grammars also lack adjacent nonterminals
- Property alone: nonunique category
- Regular patterns: chiefly lexical
- CFGs: chiefly nested syntax

Symbol tables and scope | Exam focus

• Symbol-table contents and scope operations
- Names/identities → attributes
- Type; scope; width; storage offset

- Parameter signature; constants; linkage
- Hashing: expected O(1) insertion/lookup
- Stack of lexical scope tables
- Lookup nearest scope first
- Insert current-scope declarations
- Scope exit removes visibility

• Symbol-table timing and implementation
- Scanner may intern spelling
- Semantics binds declaration/type
- First interaction: conventional, implementation-dependent
- Punctuation need not have entries
- Literals may use separate table
- Hash table: implementation, not semantic role

Names, Types and Advanced Features | Exam focus

• Member-call phase responsibilities
- obj.method(x)
- Scanner: identifiers and punctuation
- Parser: member-call structure
- Semantics: receiver, accessibility, argument types
- Overloading: signatures using static types
- Overriding: possible runtime implementation selection

• Scope, closures and generics
- Declaration identity beyond spelling
- Inner int x shadows outer float x
- Closures: captured lexical free variables
- Lifetime beyond creating frame
- Generics: parameter constraints/instantiation checks
- Symbol/type information beyond CFG
- Surface syntax may remain context-free

• Worked example object.resize(3) worked call
- Tokens: id,dot,id,(,integer,)
- Call→Expr . id (Args)

- Expr→id; Args→Expr|integer|ε
- Call⇒Expr.id(Args)⇒id.id(integer)
- Bind object, resize, 3
- Parser: shape only
- Semantics: scope, accessible member, compatible overload
- Only resize(): parses, argument/type failure

Semantic checking: static and dynamic | Exam focus

- Static semantic checks
  - Before execution
  - Compatible types; valid names
  - Restricted-scope duplicate declarations
  - Argument counts; return compatibility
  - Control-flow placement
  - C/C++ break: loop or switch
  - Annotated AST or typed IR
  - No mandatory unique output-tree name
- Dynamic checks and static proofs
  - Runtime-dependent bounds/null checks
  - Some accesses statically provable safe/faulty
  - Universal static-memory-check impossibility trap
  - Unreachable-code rules language-dependent
  - Reject specified cases, warn, or eliminate

Type rules, inference and compatibility | Exam focus

- Type-checking operation rules
  - Strict integer mod: two integers
  - Indexing: array plus integer index
  - if: Boolean condition
  - Assignment: equal/convertible types
  - Calls: parameter count and compatibility
  - Returns: result-type match
- Type inference and error distinctions

- Constraints derive types
- Integer x, numeric +, promotion rules
- Static inference without annotations
- Operator overloading: multiple meanings
- Overloading ≠ replacement for checking
- Ordinary C++/Java int x="hello": mismatch
- Zero division/bounds ≠ automatic type mismatch

Conversions need a named language

- Conversions and precision
  - Coercion: permitted implicit conversion
  - Cast: explicit conversion request
  - Widening: often increased range
  - Java int→float: widening, possible rounding
  - Java int→long: exact
  - Narrowing: range/fraction loss
- Language-specific conversion cases
  - C++ int x=3.14: 3 if representable
  - C++ int x{3.14}: narrowing rejection
  - Floating→integer: fraction discarded
  - C++ unrepresentable truncated value: undefined behavior
  - Java double→int assignment: explicit permitted cast
  - C++/Java char/int: ordinary promotion
  - Universal mixed-type ban trap

Attributes and Syntax-Directed Definitions | Exam focus

- Syntax-directed definitions and attributes
  - CFG with grammar-symbol attributes
  - Semantic rules on productions
  - Annotated parse tree: computed attribute values
  - Tree ≠ inherited/synthesized attribute
  - Synthesized: children → parent
  - Inherited: parent/sibling context

- Attribute evaluation examples and classes
  - E→E1+T; E.val=E1.val+T.val
  - Synthesized expression value
  - Inherited declaration type environment
  - S-attributed: synthesized only
  - Natural LR reduction-time evaluation
  - Suitable LL evaluation also possible
  - L-attributed: restricted left-to-right dependencies
  - Dependency cycles may prevent evaluation

Translation actions, ASTs and postfix | Exam focus

- Syntax-directed translation schemes
  - Executable actions embedded in productions
  - Actions during source translation
  - Not during compiler generation
  - Attribute results: types; values; AST pointers; code
  - Input-to-output translation
  - Beyond input-language recognition
- AST construction and postfix evaluation
  - Identifier: mkleaf(id,entry)
  - Number: mkleaf(num,value)
  - Operator: mknode(op,left,right)
  - a−4+c AST: +(−(a,4),c)
  - Postorder postfix: a 4 − c +
  - Stack: push operands
  - Pop right, then left
  - Stack-machine usefulness
  - Infix-to-postfix not universally mandatory

IR and Three-Address Code

- Intermediate-representation purpose
  - Source analysis separated from target implementation
  - Optimizer/backend reuse

- Easy generation, analysis, transformation, lowering
- Portability architectural benefit
- Possible target-dependent details
- AST; graphs; bytecode; TAC

• TAC and instruction records

- Ordinary arithmetic: one operation maximum
- Two sources; one destination
- t1=c*d; t2=b+t1; a=t2
- Quadruple: (op,arg1,arg2,result)
- Triple: position references, no result field
- Indirect triples: separate ordering list
- Easier rearrangement
- Trees: expressions; DAGs: shared subexpressions
- Quadruples/triples: instruction records

Control flow, arrays and SSA | Exam focus

• Control-flow and array translation

- if a<b goto Ltrue; goto Lfalse
- Labels join control paths
- Short-circuit AND: true-left permits right
- Short-circuit OR: false-left permits right
- Eager evaluation may change side effects
- Address: base+(i−L)*w
- Element width w; lower bound L
- Fixed-width contiguous-storage assumption

• Static single assignment

- Distinct name per definition
- Join: x3=φ(x1 from B1,x2 from B2)
- Incoming value follows actual predecessor
- φ ≠ arithmetic average
- SSA: IR naming discipline
- Coexisting control-flow graph

Basic Blocks, Flow Graphs and Dominators

- Basic blocks and leaders
  - Maximal straight-line instruction sequence
  - Entry at beginning; exit at end
  - Leaders: first instruction
  - Every jump target
  - Instruction after each branch
  - Block continues until next leader
- Flow graphs and dominators
  - CFG vertices: basic blocks
  - Edges: possible next transfers, fall-through
  - D dominates N: every entry-to-N path
  - Backedge N→H; H dominates N
  - H: natural-loop header
  - Loop-analysis structure
  - Expression DAG: value dependencies
  - DAG ≠ arbitrary control flow

Data-flow questions and equations

- Forward data-flow analyses
  - Reaching definitions: may-analysis
  - Assignments reaching without replacement
  - OUT=GEN∪(IN−KILL)
  - IN: predecessor-OUT union
  - Available expressions: must-analysis
  - Computed on every relevant path
  - Unchanged operands
  - Available-expression join: intersection
- Liveness and definite assignment
  - Liveness: backward may-analysis
  - Possible read before future redefinition
  - OUT[B]=⋃successors IN
  - IN[B]=USE[B]∪(OUT[B]−DEF[B])

- Fixed-point iteration
- Register allocation and dead-value removal
- Definite assignment: initialized on every path
- Reaching definitions alone insufficient

Folding, Propagation and Dead Code

• Folding, propagation and elimination
- Constant folding: 3*4→12
- Constant propagation: x=3; y=x+2→y=3+2
- Then fold
- Copy propagation: source remains unchanged
- Dead code: unreachable or unused computation
- Necessary-side-effect exclusion

• Combined optimization worked case
- const x=8,y=16; if(x>y){z=x+y;}
- Propagate constants
- Fold 8>16 to false
- Remove unreachable branch
- Compile-time evaluation plus dead-code elimination
- Overlapping optimization choices
- Unused volatile read may remain necessary
- Effectful calls may remain necessary

Reuse, Strength Reduction and Peepholes

• Reuse, identities and strength reduction
- CSE: reuse earlier expression value
- Unchanged operands and relevant memory
- DAG sharing
- Valid integer identity: x+0→x
- Strength reduction: equivalent cheaper operation
- Unsigned x*8→x<<3
- Valid width and shift count
- Actual saving depends on target costs

- Peephole optimization
  - Short instruction windows
  - Redundant load/store removal
  - Machine-idiom replacement
  - Jump-chain shortening
  - jump L1; L1→L2 ⇒ jump L2
  - Local optimization: broader category
  - Peephole: concrete method
  - Overlapping choices require requested level

Loop Invariants, Induction and Fusion

- Loop invariants and induction variables
  - Unchanged expression inputs
  - Preheader hoisting: newly executed paths safe
  - Induction: loop-invariant increment
  - i=i+1
  - Repeated t=4*i
  - Equivalent running t increased by 4
  - Reduced repeated multiplication
- Loop transformation contrasts
  - Fusion/jamming: compatible bounds, safe dependencies
  - Unrolling: replicated body
  - Reduced branch overhead, exposed parallelism
  - Remainder iterations and code growth
  - LICM; fusion; unrolling; strength reduction
  - Distinct transformations
  - Machine-dependent/independent taxonomy
  - Local/global/loop taxonomy
  - Classification requires criterion

Optimization legality and aliasing | Exam focus

- Optimization legality and aliasing
  - Defined observable behavior preserved

- Required side effects and exceptions
- Aliases: same memory possibility
- *p=5 may invalidate q-load
- Reuse requires proven nonaliasing/equivalent proof
- Calls may invalidate memory facts

• Unsafe transformation boundary cases
- Potentially trapping division hoist
- Zero-iteration loop trap
- Floating reassociation: rounding/NaN changes
- Signed C/C++ overflow and shifts
- Unsigned modular arithmetic distinction
- Mathematical identity ≠ legal rewrite
- Dependence analysis: required memory-access order
- Fusion/scheduling legality

Target code generation

• Target code-generation tasks
- Checked IR → target form
- Instruction selection: target operations
- Register allocation: value locations
- Scheduling: dependency-respecting order
- Storage layout: offsets and ABI
- Register assignment: actual physical names

• Target instruction example and variants
- t=a+b
- LOAD R1,a; LOAD R2,b
- ADD R3,R1,R2; STORE t,R3
- Combined load/arithmetic or register-held t
- Machine idioms; register availability; instruction order
- Execution-time effects
- Absolute code; relocatable object; assembly; bytecode
- TAC normally intermediate
- Cannot-ever-be-target universal trap

Descriptors and getReg | Exam focus

- Register and address descriptors
  - Register descriptor: current held values
  - Address descriptor: all current value locations
  - Memory and multiple registers
  - Register-only assignment: stale memory copy
  - Code-generator bookkeeping
  - Descriptors ≠ source types
- getReg code-generation sequence
  - Select operand/result registers
  - Necessary loads → operation → descriptor update
  - Prefer suitable existing register
  - Reuse dead operand
  - Otherwise spill victim
  - Store live value lacking current alternative
  - Block exit: preserve successor-needed values

Interference graphs and spilling

- Liveness, interference and spilling
  - Live: possible use before redefinition
  - Edge: values need distinct coexisting registers
  - Registers as colors
  - Nonoverlapping live ranges may share
  - Insufficient register fit: spill to memory
  - Insert loads/stores
- Register-allocation complexity limits
  - General fixed k≥3 coloring: NP-complete
  - Optimal assignment/spill minimization: generally hard
  - Practical heuristic allocators
  - Restricted graphs/SSA: different properties possible
  - Not every allocation algorithm NP-complete
  - Decision-model versus optimization distinction

Runtime Storage and Calling Conventions | Exam focus

- Runtime storage and frames
  - Static: entire execution lifetime
  - Stack: nested calls and recursion
  - Heap: beyond-call lifetimes
  - Frame parameters and return address
  - Saved registers; locals; temporaries
  - Target-dependent sizes, alignment, offsets
  - Two-byte int: module example
  - Not C++ guarantee
- Calling convention and escape analysis
  - Argument/result locations
  - Caller/callee-saved registers
  - Stack alignment and frame responsibilities
  - Escape: outlive/access outside allocation region
  - Proven nonescape: stack allocation possible
  - Scalar replacement: allocation elimination
  - Preserved identity and lifetime semantics

Inlining and dynamic dispatch | Exam focus

- Inlining benefits and limits
  - Suitably renamed function-body copy
  - Preserved argument evaluation, returns, side effects
  - Reduced call overhead
  - Exposed optimization opportunities
  - Increased code size/cache pressure
  - Recursive expansion cannot be unlimited
- Dynamic dispatch and devirtualization
  - Overridden method: receiver runtime class
  - Common per-class virtual-method table
  - Object pointer to table
  - Load selected entry
  - Implementation strategy, not universal requirement

- Name mangling: linker-symbol distinctions/overloads
- Mangling ≠ runtime method selection
- Proven receiver type: devirtualization

Recovery and compiler-construction tools

- Error recovery methods
  - Panic mode: skip to synchronizing token
  - Semicolon or closing brace
  - Phrase-level: local insertion/deletion
  - Location and cause reporting
  - Avoid diagnostic cascades
  - Lexical, syntax, semantic error detection
  - Error handling: shared service
- Compiler-construction generators
  - Lex/flex: patterns/actions → scanner
  - Yacc: Yet Another Compiler Compiler
  - Yacc/Bison: grammar/actions → parser
  - Definitions; %% rules; %% user code
  - Classic Yacc: LALR
  - Bison: additional LR/generalized modes
  - Scanner generator ≠ CFG parser replacement

Expression Translation Through the Compiler | Exam focus

- Expression pipeline trace
  - a=b+c*d;
  - Scanner: names and operators
  - Parser: assignment with +(b,*(c,d))
  - Semantics: declarations and operand/result types
  - float a; int b,c,d;
  - Possible result conversion before store
  - Language arithmetic rules
- Expression IR to target trace
  - t1=c*d; t2=b+t1

- a=convert_if_needed(t2)
- Constant operands: folding
- Repeated unchanged expressions: sharing
- Target instruction and register selection
- Same a spelling, distinct scoped declarations
- Entering representation and added information
- Evidence for newly diagnosable errors

Missing and Conflated Lexical Answers

- Lexical-answer repairs
  - Primary output: token stream
  - Symbol table/AST/assembly: invalid substitutes
  - Side structures versus primary output
  - Smallest meaningful sequence: token/lexeme conflation
  - Required abstraction unspecified
  - Loop header: 14 despite missing option
- Grouping units and storage roles
  - Characters → token classes: lexical
  - Tokens → expressions/statements: syntax
  - Vague syntactic-class stem needs units
  - Symbol tables: identifiers and attributes
  - Not necessarily every token kind
  - Trees: hierarchical constructs
  - Symbol tables: declaration metadata

Overlapping Parser Choices

- Overlapping parser choices
  - Shift-reduce and LR: both bottom-up
  - Negative top-down question: multiple answers
  - Recursive/nonrecursive predictive: both potentially efficient
  - Nesting/tables: not uniquely LR
  - LR choice ≠ ambiguity removal
  - Leftmost derivation ≠ disambiguation policy

- LL(1) key and precedence repairs
  - Overlapping FIRST; left recursion; ambiguity
  - All rule out useful LL(1) grammar
  - A-and-B-only key invalid
  - Ask all conditions or conflicting cell
  - Precedence/operator-precedence: concrete conflict and policy
  - Universal shift-reduce solution trap

Semantic and Translation Claim Repairs

- Semantic activity versus data structure
  - Type checking: activity
  - Symbol table: data structure
  - Both support semantic analysis
  - Semantic rules compute attributes
  - Types; values; code: nonexhaustive examples
  - Symbol records: names and types
- Semantic and pipeline claim repairs
  - Zero division/bounds ≠ clean type-error list
  - Indentation/syntax mistakes ≠ clean type errors
  - Incompatible operand plus named language
  - Conventional final phase: code generation
  - Later target optimization possible
  - Annotated AST or typed IR
  - Representation versus annotation versus phase
  - Forced unique tree-name trap