CS Exit Ethiopia

Complete Study Notes

Computer Science Exit Examination

1 selected course | 18 November 2026

Selected courses

01. Data Structure and Algorithms

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.

Data Structure and Algorithms 6

How to use these study notes

START01 A practical study cycle

Start with a small set of questions before reading.

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

Use the question lookup to open the relevant section.

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

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

Then change one assumption or input and solve the variation.

This checks understanding beyond recognition.

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

Adjust the timing to your needs.

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

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

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

Keep the log short enough to review.

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

Explain how the same word changes meaning across contexts.

START02 What to emphasize

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

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

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

Allocate revision by both exam share and your error log.

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

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

Definitions: distinguish nearby concepts, not just their names.

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

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

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

Design choices: state the criterion.

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

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

START03 How to handle exam traps

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

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

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

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

Learn the correct rule and the smallest clarification needed.

Do not memorize an arbitrary key as a universal fact.

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

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

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

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

Revisit the reading sections behind recurring mistakes.

Data Structure and Algorithms

Complete Study Notes

DSA01 | Start with the data and the required operations

An abstract data type (ADT) defines values and operations; representation supplies implementation.

Arrays or linked nodes can implement the same LIFO stack.

Abstraction keeps relevant details: patient urgency and arrival order matter for service; hair colour does not.

Arrays, linked lists, stacks and queues are linear.

Trees model hierarchies; graphs model general connections.

A record grouping integer, character and floating fields is heterogeneous; an ordinary array has one element type.

Data Structure and Algorithms module

DSA02 | Arrays: indexed access, capacity and shifting

For equal-size elements, address(A[i]) = base + i × element_size under zero-based indexing.

This arithmetic gives constant-time positional access in the usual RAM cost model.

A row-major rectangular matrix uses base + (row × columns + column) × element_size.

A fixed array has a chosen capacity; the choice need not occur at compile time.

A resizable array allocates a larger block and copies when needed.

Inserting in the middle shifts later elements, so it costs linear time.

Appending into spare capacity is constant; a resizing append can be linear, though repeated geometric growth gives amortized constant append cost.

Example

Work it through

- Insert 9 at index 1 of [2,4,6] with a spare slot: shift 6 then 4 right, giving [2,9,4,6].

Data Structure and Algorithms module

DSA03 | Operations, records and memory ownership

Traversal processes every item; searching locates a target; insertion and deletion change membership; updating changes a stored value.

Sorting reorders items, while merging combines sequences.

To merge sorted [1,4] and [2,3], repeatedly emit the smaller front, producing [1,2,3,4].

Destroying a structure releases owned storage.

A C struct groups named fields, including arrays and pointers.

A singly linked node needs payload plus a same-type link.

C fields cannot have initializers inside the type declaration; modern C++ permits default member initializers.

In C++, match new with delete and new[] with delete[].

Repair links before freeing a node; never dereference it afterwards.

Data Structure and Algorithms module

DSA04 | Algorithm properties are separate obligations

Finiteness means termination after finitely many steps for every valid input.

Definiteness means each instruction has one interpretation; effectiveness means each step is executable.

Correctness means meeting the stated result for every valid input.

A finite program can still return the wrong answer.

A problem specifies inputs, outputs and assumptions.

Language independence concerns the procedure, not its implementation.

Efficiency needs a resource model.

An unfamiliar characteristic is not false merely because a teaching list omits it.

Data Structure and Algorithms module

DSA05 | Measure costs before naming their growth

Time counts chosen operations; space counts memory, including live recursive frames.

Distributed algorithms may also measure communication volume.

A single assignment is one unit in the elementary model, not literally one second.

Sequential blocks add costs; independent nested loops multiply their iteration counts.

Conditional worst-case cost includes the test and the more expensive branch.

Best, average and worst compare equal-size inputs; average needs a distribution.

For m operations each with mean cost f(n), expected total is m f(n), without requiring independence.

Exact coefficients depend on which elementary operations count.

Data Structure and Algorithms module

DSA06 | Upper, lower and tight bounds

For eventually nonnegative functions, f(n)=O(g(n)) means f(n)≤c g(n) for some positive c beyond a threshold k. Omega reverses the inequality; Theta requires both.

Little-o means f/g tends to zero; little-omega means it grows without bound.

These relations can describe best-case, average-case, worst-case or space functions.

For n≥1, $3n^2+4n+1 \le 8n^2$: c=8,k=1 witness $O(n^2)$.

Constants and lower-order terms disappear from tight polynomial growth.

Transitivity multiplies witnesses: f≤c g≤cd h beyond both thresholds.

Example

Work it through

- Increasing representative rates: 1, log n, n, n log n, n^2, n^3, 2^n, 3^n, n!, n^n. Logarithm bases differ only by a constant factor.

Exam tip

Exam trap

- A linear function is also $O(n^2)$. MCQs usually intend the tightest listed order; mixed O and Theta options may overlap.

Data Structure and Algorithms module

DSA07 | Count loops and account for hidden access costs

A loop i=1; i<n; i*=2 takes about $\log_2 n$ iterations.

Two independent n-iteration loops nested around constant work take $\Theta(n^2)$.

An n-level ordinary recursion with constant live storage per level needs Theta(n) stack space.

Indexed access is not constant for every List implementation.

Java LinkedList.get(i) walks from the nearer endpoint.

Summing get(0), get(2), get(4), … over a length-N list still gives $\Theta(N^2)$, because a linear number of middle indices each require linear traversal.

An iterator scans once in linear time.

An exact executable runtime requires valid code and an explicit cost convention.

Oracle Java SE 21: LinkedList

DSA08 | Understand the linked-node chain

A singly linked node contains data and next.

Head points to the first node; null ends a noncircular chain.

Empty means head==null.

Nodes allocated as needed grow dynamically, with pointer overhead and no direct index arithmetic.

To display or count nodes, follow next from head until null, processing each node.

Counting is Theta(n) unless size is maintained.

An array-linked list still follows links unless separately position-indexed.

Data Structure and Algorithms module

DSA09 | Update links without losing the rest of the list

Prepend with new.next=head; head=new.

To insert after a known predecessor p, set new.next=p.next before p.next=new.

Appending with only head requires a tail traversal; maintaining tail avoids it.

Delete the head by saving old=head, advancing head=head.next and releasing old in C++.

Deleting the tail of a singly linked list requires its predecessor, even when tail itself is known.

Handle empty and single-node lists separately; after removing the only node, set both head and tail to null.

To remove a known successor, bypass it before deallocation.

Clearing an owned n-node chain visits and releases every node.

Reverse a chain with prev=null, cur=head: save next=cur.next; set cur.next=prev; advance prev=cur and cur=next; finish with head=prev.

This is linear time and constant extra pointers.

Example

Work it through

- For 3→7→9, insert 5 after 3: first point 5 at 7, then 3 at 5. Reversing those assignments can lose access to 7→9.

Data Structure and Algorithms module

DSA10 | Doubly linked, circular and practical list choices

Doubly linked nodes add prev; head and tail enable constant-time operations at both ends.

Singly linked tail deletion still needs its predecessor.

A circular tail links back to head; stop traversal on returning to the start, handling empty separately.

Lists suit changing sequences when references to insertion positions are already available.

Finding a position is still linear.

Lists can hold polynomial terms (coefficient, exponent), directory or block links, child lists in nonbinary trees, and hash chains.

Browser history can use a doubly linked cursor.

A playlist may be circular; airline seat lookup requires a specified seat-record model, not an automatic linked-list answer.

Operation	Array	Singly linked list
Access index i	O(1)	O(n) worst
Insert at front	O(n) shifts	O(1)
Append	O(1) with spare capacity	O(1) with tail; O(n) head only
Delete tail	O(1)	O(n), predecessor needed

Data Structure and Algorithms module

DSA11 | Implement a stack with one consistent top convention

Stack policy is LIFO, equivalently FILO.

Push inserts, pop removes/returns top, and peek only observes.

With zero-based occupied slots, top=-1 is empty and top=C-1 is full.

Check before A[++top]=x or returning A[top--].

A count-based alternative uses size=0 and A[size++]=x. Do not combine its empty/full tests with the top-index convention.

Fixed-array push/pop are constant time.

Linked stacks achieve constant time by inserting and deleting at the head; using a singly linked tail for pop would require a predecessor traversal.

Exam tip

Exam trap

- Overflow is an attempted insertion into full storage; underflow is an attempted removal from empty storage. Merely checking isEmpty does not underflow.

Data Structure and Algorithms module

DSA12 | Trace stack operations and reversible actions

Write the stack bottom-to-top after every instruction and record popped values separately.

Push 1, push 2, pop emits 2 and leaves [1].

Push 1, push 2, pop, pop, pop then emits 2,1,1; push 2, pop emits 2.

The complete output is 2,2,1,1,2.

Calls nest on a runtime stack; the latest call returns first.

Bracket matching pushes opening brackets and matches each closing bracket against top.

Reversal pushes input then pops it.

Undo moves the latest action from undo to redo; redo moves it back.

A new edit after undo normally clears redo.

DFS and backtracking use the same latest-pending-choice discipline.

Data Structure and Algorithms module

DSA13 | Infix, postfix and prefix expressions

Infix places an operator between operands and uses precedence and parentheses.

Prefix places it first; postfix places it last.

Their fixed operator arity determines grouping.

To convert infix to postfix, output operands immediately and hold operators on a stack.

Pop higher-precedence operators before pushing a new operator; also pop equal precedence for left-associative operators.

Parentheses temporarily override this order.

To evaluate postfix, push operands.

At a binary operator, pop right operand first, then left, and push left op right.

Build an expression tree for either prefix (preorder) or postfix (postorder); a stack is useful but is not the only possible conversion representation.

Example

Work it through

- A+B*C-D/E becomes ABC*+DE/-. For 8 3 -, pop 3 then 8, producing 5. (A+B)*C becomes AB+C* and prefix *+ABC.

Data Structure and Algorithms module

University of Manchester: Infix, Postfix and Prefix

DSA14 | Queues preserve arrival order

An ordinary queue enqueues at rear, dequeues at front, and peeks at front.

This is FIFO or first-come, first-served.

A linked implementation with head and tail supports both operations in constant time: append at tail, remove at head.

The first insertion sets both pointers; removing the last item clears both.

With head alone, insertion must traverse to the tail.

FIFO suits equal-priority printer jobs, shared-resource waiting, asynchronous buffers and level-order exploration.

A play sequence can also use a queue.

Distinguish airline seat records from waiting requests; an application name alone does not force one representation.

Example

Work it through

- Enqueue 4,8; peek is 4, so a conditional removal only if peek>5 does nothing. Enqueue 1,6 then dequeue: [8,1,6].

Data Structure and Algorithms module

DSA15 | Circular buffers: logical order differs from physical slots

A simple noncircular array queue may report overflow when rear reaches the final index despite released slots at the front.

A circular queue reuses slots with modulo arithmetic.

Use a count convention: front points to the next removal, size counts items, and insertion slot is (front+size)%C. Empty means size==0; full means size==C.

Dequeue reads A[front], sets front=(front+1)%C and decreases size.

An alternative reserves one unused slot and uses next(rear)==front for full; its empty test and capacity differ.

State the convention before tracing.

Example

Work it through

- Full capacity-six [11,13,15,17,19,21], front=0: remove 11, then enqueue 9 into slot 0. Physical array: [9,13,15,17,19,21]; logical front-to-rear queue: [13,15,17,19,21,9]. A noncircular version can instead overflow.

Data Structure and Algorithms module

DSA16 | Deques, merging and splitting queues

A deque permits both-end insertion/removal without guaranteeing middle access.

Endpoint-linked operations are constant but middle indexing is linear.

A circular-array deque may support indexed access.

Dequeue names removal; deque names the structure.

Merging FIFO queues needs an explicit policy: append one sequence after another or alternate their fronts.

Splitting by a predicate preserves each subgroup's original order if each removed item is appended to its matching output queue.

A fixed-capacity buffer also needs a full-policy choice: reject, block or overwrite.

Capacity alone does not choose a policy.

Data Structure and Algorithms module

DSA17 | Priority queues serve by priority, with an explicit tie rule

A priority queue returns the smallest or largest priority key under its convention.

Urgent patients may precede earlier arrivals.

Insertions and extractions can interleave.

Binary heaps support constant peek and logarithmic insert/extract.

Arrays and lists can also implement priority queues: an unordered sequence gives cheap insertion but linear extremum removal; an ordered sequence reverses that tradeoff.

Equal priorities have no inherent FIFO guarantee.

If stable service is required, compare (priority, arrival_counter).

Choose the priority direction and tie rule before tracing.

Exam tip

Exam trap

- FIFO ordering among equal priorities requires an explicit stability policy. That is an additional policy, not a universal priority-queue rule.

Sedgewick and Wayne: Priority Queues

DSA18 | Tree vocabulary and the different binary shapes

A rooted tree has one parentless root and one parent per other node.

It is connected and acyclic, has n-1 edges, and one path between any pair.

Depth counts root-to-node edges; height is maximum depth.

A single-node tree therefore has height zero.

Leaves have no children.

Binary means at most two children.

Full means each node has zero or two children.

Complete means all earlier levels are filled and the last fills from left.

Perfect means all leaves share a level and every internal node has two children.

Balanced means height stays logarithmic under a stated rule; it does not require a perfect shape.

Data Structure and Algorithms module

DSA19 | BST search follows a global ordering invariant

With distinct keys, every key in a node's left subtree is smaller and every key in its right subtree is larger.

Immediate-child comparisons alone are insufficient: root 10, left child 5, and 5's right child 12 violate the root's restriction.

Duplicates require a separate count or placement policy.

Search compares at each node and follows one child until equality or null.

Insertion uses that same path to attach a new leaf.

Operations cost O(height): logarithmic for balanced trees, linear for a chain.

Best-case successful search is Theta(1), a root hit.

An optimal BST minimizes expected search cost using access probabilities; constructing it is a different problem from searching it.

Standard optimal-BST dynamic programming tries each possible root for each key interval: $O(n^2)$ intervals times O(n) roots gives $O(n^3)$ construction.

It minimizes probability-weighted search depth, not necessarily worst-case height.

Example

Work it through

- Insert 40,20,50,10,30,60: root children are 20 and 50; 20 has 10,30; 50 has right child 60. Height is two edges. Sorted insertions 1,2,3,4,5 instead form a right chain.

Data Structure and Algorithms module

University of Freiburg: Optimal Binary Search Trees

DSA20 | Tree traversal is a rule you can execute

Preorder visits node,left,right; inorder left,node,right; postorder left,right,node.

Apply recursively; null children do nothing.

Inorder returns sorted keys only for a BST.

Preorder supports copying, postorder supports deleting children before parents, and level order uses a FIFO queue.

For root 4, children 2 and 6, and leaves 1,3,5,7: preorder is 4,2,1,3,6,5,7; inorder is 1,2,3,4,5,6,7; postorder is 1,3,2,5,7,6,4.

Level order is 4,2,6,1,3,5,7.

Randomized visitation is possible; it is simply not one of these standard named depth-first orders.

Data Structure and Algorithms module

DSA21 | BST deletion preserves every subtree restriction

A leaf can be detached by clearing its parent link.

For one child, make the parent bypass the node and point to the child.

For two children, copy the inorder predecessor (left-subtree maximum) or successor (right-subtree minimum), then delete that replacement from its original location.

That second deletion has at most one child.

Deletion by merging is another approach.

Promote the right subtree and attach the entire old left subtree to the minimum node in the promoted right subtree; or use the symmetric left promotion.

Update root itself when deleting root.

Copying values and reconnecting subtrees can yield different valid shapes, so a trace must name its deletion method.

Data Structure and Algorithms module

DSA22 | Balanced trees prevent the sorted-input chain

AVL balance factor is height(left)-height(right); every node must have factor -1,0 or 1.

After insertion or deletion, update heights upward and rotate where necessary.

LL imbalance needs a right rotation; RR needs left; LR rotates the left child left then the node right; RL is symmetric.

Rotations preserve inorder key order.

Inserting 30,20,10 triggers LL repair: 20 becomes root with children 10,30.

Red-black colouring rules provide another logarithmic-height discipline.

Both prevent linear worst-case search; balanced trees also support ordered ranges unlike ordinary hashing.

Sedgewick and Wayne: Balanced Search Trees

DSA23 | A heap combines completeness with a weaker ordering

A binary max-heap is complete with each parent at least its children; a min-heap reverses this order.

Unlike BSTs, heaps do not order entire subtrees.

The root is maximum.

If all keys are distinct, the second largest is a root child.

With duplicate maxima, the second largest distinct value may be deeper: [9,9,9,8,7,6,5] is a valid max-heap and that value is 8 at index 3.

The minimum may be any leaf, requiring linear worst-case scanning.

Zero-based array representation: root index 0, left=2i+1, right=2i+2, parent=floor((i-1)/2) for i>0.

One-based representation uses children 2i,2i+1 and parent floor(i/2).

A child exists only if its index is inside the current heap size.

Exam tip

Exam trap

- The right-child formula 2i+1 assumes one-based indexing. For zero-based indexing with the root at index 0, the right-child formula is 2i+2.

Data Structure and Algorithms module

DSA24 | Restore a heap after insertion or removal

Insert at the next complete-tree position and swap upward while parent order is violated.

To remove the root, move the final element to root, reduce heap size, then swap downward with the more dominant child until order holds.

Each path has logarithmic length.

Building bottom-up by sinking internal nodes takes linear total time; repeated independent insertions give an O(n log n) construction bound.

Example

Work it through

- Max-heap [10,8,6,2,1,4,5]: remove 10, move 5 to root and sink, giving [8,5,6,2,1,4]. Remove 8, move 4 up then exchange with 6: [6,5,4,2,1].

Data Structure and Algorithms module

Sedgewick and Wayne: Priority Queues

DSA25 | Hash addresses, uniformity and random function choice

Direct addressing reserves a position for every possible key in a finite universe U, costing Theta(|U|) slots.

A hash table uses h(key) in 0..m-1 and stores only actual entries plus bucket capacity.

With h(k)=k mod 6, key 36 has home address 0; collisions may move its final storage position.

Determinism means repeated equal keys get the same address for the chosen function.

Uniformity means expected keys spread evenly.

Universal hashing chooses one function randomly from a suitable family for the table, independently of a fixed key set; it does not choose a fresh unrelated function for each lookup.

Random choice controls collision probability, not a promise of zero collisions.

MIT 6.006: Recitation4 Hashing

DSA26 | Resolve collisions and watch the load factor

Separate chaining stores colliding entries in a bucket list.

Load alpha=n/m is average chain length; expected operations are O(1+alpha) under suitable distribution assumptions, with O(n) worst cases.

Open addressing keeps entries inside the array and probes until a key or an available slot is found.

Linear probing tries (h(k)+j)%m. For size 7 and keys 10,17,24, all home addresses are 3 and occupied positions become 3,4,5.

Quadratic probing changes the step; double hashing uses h1(k)+j h2(k), modulo m, with a step relatively prime to m to cover all slots.

Neither method guarantees no collisions.

Deletion needs tombstones or reinsertion so searches do not stop prematurely.

At h(k)=3 in a seven-slot linear-probing table, addresses wrap 3,4,5,6,0,1,2.

A home address is not guaranteed to remain the final slot.

Sedgewick and Wayne: Hash Tables

DSA27 | Maps, sets and caches specify different contracts

A map associates unique keys with values; different keys may share values, and entries can be traversed.

A set forbids duplicate elements.

Null and iteration-order policies depend on implementation.

Lists preserve positions and generally allow duplicates; Java LinkedList also permits null.

A dictionary or compiler symbol table can use hashing for expected constant lookup, or a balanced tree for logarithmic lookup with ordering.

A bounded cache also needs eviction rules: FIFO uses a queue; LRU can combine hash lookup with a doubly linked recency list.

Move a hit to the recent end and evict the least-recent end when full.

No storage structure is universally best without operation and policy requirements.

Oracle Java SE 21: Map

Oracle Java SE 21: Set

Oracle Java SE 21: LinkedList

Oracle Java SE 21: LinkedHashMap

DSA28 | Linear search: a scan with an early stop

Compare the target with successive elements until equal or exhausted.

Unsorted data is allowed; sorted data is allowed too.

A first-element match costs Theta(1).

An absent target or last-position match needs n comparisons, giving Theta(n) worst time.

Uniformly likely successful positions average (n+1)/2 comparisons, still linear.

For a single unsorted query, sorting first is rarely justified.

Small lists may favour scanning because setup matters.

Sorting does not avoid examining all values for mean, variance or arbitrary parity counts.

Data Structure and Algorithms module

DSA29 | Binary search: maintain a shrinking ordered interval

On an ascending random-access sequence, set lo=0, hi=n-1.

Compute mid=lo+(hi-lo)//2.

Equality finishes; if target<A[mid], set hi=mid-1; otherwise set lo=mid+1.

Stop unsuccessfully when lo>hi.

Reverse comparisons for descending order.

Comparable keys and valid finite bounds matter; fixed allocation before population is not an extra algorithmic requirement.

Worst probes are floor($\log_2 n$)+1 for n>0: seven for 100 keys, about twenty for one million.

Iteration uses constant space; recursion uses logarithmic stack depth.

Linked midpoint traversal loses logarithmic total time.

Boundary variants retain a candidate rather than stopping at equality.

Example

Work it through

- Find 19 in [11,13,15,17,19,21]: mid=2 gives 15; search indices 3..5; mid=4 gives 19. Binary debugging needs an ordered good/bad boundary, not arbitrary fault locations.

Data Structure and Algorithms module

DSA30 | Search variants need their own assumptions

Interpolation search estimates the index from numeric values: lo+(hi-lo)×(key-A[lo])/(A[hi]-A[lo]), rounded to a valid index.

Guard equal endpoints and out-of-range targets.

On sorted, suitably uniform data its expected probes can be O(log log n); pathological distributions can require linear time.

Binary search gives a logarithmic worst bound without that distribution assumption.

Sorted-array ternary search compares two split points and keeps the relevant third, still logarithmic.

Jump search probes block boundaries then scans within a block.

Regular intervals could mean unit-step scanning or larger jumps.

Interpolation also needs sortedness, so prerequisite questions can overlap binary search.

University of Waterloo, Douglas Harder: Interpolation Search

University of Massachusetts: Running Time lecture

University of Waterloo, Douglas Harder: Jump Search

DSA31 | Compare sorting rules, not just algorithm names

Stable sorting preserves the input order of equal keys.

An in-place element algorithm uses a fixed amount of extra element storage; a recursive call stack may still matter when total auxiliary space is requested.

There is no single algorithm identified by stable or in-place alone.

State the implementation, input case and resource being compared.

Method	Best / average / worst time	Extra storage; usual stability
Bubble with early stop	n / n^2 / n^2	constant; stable
Selection	n^2 / n^2 / n^2	constant; unstable

Method	Best / average / worst time	Extra storage; usual stability
Insertion	$n / n^2 / n^2$	constant; stable
Quicksort	$n \log n / n \log n / n^2$	stack usually log n average, n worst; unstable
Ordinary array mergesort	n log n in all cases	linear; stable with left-first ties
Heapsort	O(n log n) bound in all cases	constant with iterative sink; unstable

Exam tip

Exam trap

- Big-O bounds can overlap. $n \log(n^2)=2n \log n$, so it has the same growth as n log n, unlike $n(\log n)^2$.

Data Structure and Algorithms module

DSA32 | Bubble sort: complete adjacent passes

For ascending order, compare each adjacent pair and swap only when left>right.

The largest remaining value moves to the final unsorted position after one left-to-right pass.

Shrink the next pass by one position.

Quadratic worst/average work comes from up to n(n-1)/2 pair comparisons.

Early stopping makes a sorted input linear: one no-swap pass finishes.

A fixed pass schedule stays quadratic.

Strict swaps preserve equal-key order.

Cocktail sort runs bubble passes in both directions and is stable with strict swaps.

Example

Work it through

- [4,1,3,2] → [1,4,3,2] → [1,3,4,2] → [1,3,2,4] after the first pass.

Data Structure and Algorithms module

DSA33 | Selection sort: minimize exchanges, not comparisons

At position i, scan the remaining suffix for its minimum, then exchange it with A[i].

There are n-1 meaningful placements and n(n-1)/2 comparisons regardless of initial order.

At most n-1 exchanges of distinct positions are needed.

A loop that also swaps the last item with itself can count n swap calls.

Selection suits expensive record moves with cheap key comparisons, not every large dataset.

It is unstable: exchanging first 2a with 1 in [2a,2b,1] gives [1,2b,2a].

Constant extra element storage is shared by several sorts.

Data Structure and Algorithms module

DSA34 | Insertion sort: extend a sorted prefix

Begin with the first element already sorted.

For each remaining item, save its value, shift larger preceding values right, and insert it into the gap.

That gives n-1 insertion passes.

On sorted input each pass needs one ordering check and no shifts: linear time.

Reverse order causes 1+2+…+(n-1) shifts: quadratic time.

Few inversions mean few shifts, favouring insertion for nearly sorted or small arrays.

Shift only strictly larger keys for stability.

A linked implementation saves array shifts but still must locate each insertion position.

Example

Work it through

- Insert 2 into prefix [1,4,7]: save 2, move 7 then 4 right, and write 2 after 1. The sorted prefix becomes [1,2,4,7].

Data Structure and Algorithms module

DSA35 | Shell sort: insertion across decreasing gaps

Choose a sequence of positive gaps ending at 1.

For each gap h, insertion-sort each subsequence of positions h apart.

The final h=1 pass finishes ordinary sorting after earlier long moves have reduced disorder.

An h-sorted array satisfies A[j]≤A[j+h] wherever both indices exist.

The O(n^(3/2)) bound depends on gaps; Knuth increments 1,4,13,… have that worst order.

Shell sort is generally unstable because distant moves can reorder ties; extra element storage is constant.

Example

Work it through

- For [5,4,3,2,1], gap 3 sorts positions (0,3) and (1,4), giving [2,1,3,5,4]. Gap 1 then produces [1,2,3,4,5].

Data Structure and Algorithms module

Sedgewick and Wayne: Shell.java

DSA36 | Quicksort: separate partition guarantees from exact traces

Choose a pivot, partition, then recursively sort the pieces.

Balanced partitions give n log n work; repeated extreme pivots sum n+(n-1)+... to quadratic work.

Expected n log n needs an input/random-pivot model.

Random and median-of-three choices have different tradeoffs; neither is universally best.

In the ten-distinct-value example, a final pivot 5 has five smaller values before it and four larger after it.

Internal orders depend on the partition algorithm; several candidate arrays can satisfy the invariant.

State the algorithm before predicting one exact array.

Quicksort normally rearranges in the same array but recursive storage can be linear in the worst case.

Data Structure and Algorithms module

DSA37 | Merge sort and internal versus external sorting

Split into halves, sort them, then merge by advancing the smaller current front.

With left-first tie handling, equal keys retain order.

Ordinary array merging needs a linear auxiliary buffer.

There are logarithmically many levels with linear merge work per level, giving n log n time even on sorted input unless an optimization skips merges.

Top-down mergesort recurses; bottom-up merges runs of widths 1,2,4,... iteratively.

In-place merging variants exist with different tradeoffs; linear merge names an operation, not a universally standardized sort variant.

Internal sorting fits the working dataset in memory.

External sorting builds sorted runs and merges buffered streams from disk, aiming to reduce I/O. Reverse order alone does not characterize maximum merge comparisons.

Data Structure and Algorithms module

Virginia Tech OpenDSA: External Sorting

DSA38 | Heapsort: repeatedly place the extreme at the end

For ascending in-place heapsort, build a max-heap over the array.

Exchange root with the final active element, shrink the active heap, and sink the new root.

The removed maximum now belongs in the sorted suffix.

Repeat until one active item remains.

Building with bottom-up sinking is linear; repeated extraction gives O(n log n) worst time.

Iterative sinking needs constant auxiliary storage.

The method is generally unstable because long root-to-end exchanges can reorder equal keys.

It is not uniquely the method for sorting by priorities: any comparison sort can compare a priority field.

Heap extraction is its distinctive mechanism.

Data Structure and Algorithms module

DSA39 | Radix sorting uses key digits, not arbitrary magic

LSD radix sorting handles fixed-width digit keys by stably sorting least-significant digit first, then moving left.

Stable intermediate passes preserve the ordering already established by less-significant digits.

With d digits and radix r, counting-based passes take O(d(n+r)) time and O(n+r) auxiliary storage.

MSD begins at leading digits and subdivides equal-prefix groups.

Strings are lexicographically comparable.

Radix methods require accessible digits and length/signed-value conventions, not arbitrary unstructured keys.

Large radix, long keys or small n may favour comparison sorting.

Sedgewick and Wayne: LSD.java

DSA40 | Graphs: represent the relationships you actually need

A graph has vertices V and edges E. Directed edges distinguish u→v from v→u; undirected edges represent two-way relationships.

Following accounts is directed; a mutual friendship relation is undirected.

Weighted edges carry costs; unweighted shortest-path questions count edges.

DAG expands to directed acyclic graph.

Trees are connected acyclic graphs, although graph data structures may also contain cycles.

A matrix uses Theta(V^2) space and constant edge lookup.

Adjacency lists use Theta(V+E) space and enumerate neighbours.

Dense simple graphs approach quadratic edge counts.

Examining every edge costs more on denser graphs; vertex-only operations need not inspect every edge.

Data Structure and Algorithms module

DSA41 | BFS: discover one distance layer at a time

Mark the source when enqueuing it.

Repeatedly dequeue a vertex, examine its neighbours, and enqueue each unvisited neighbour while marking it.

Store predecessor[v] when discovering v if a path must be reconstructed.

FIFO ensures vertices at distance d are explored before later distance layers.

BFS finds minimum-edge paths in unweighted graphs, not arbitrary minimum-weight paths.

With adjacency lists it costs O(V+E); scanning a matrix costs $O(V^2)$.

To traverse a disconnected graph, restart from each unvisited vertex.

Neighbour order affects tied traversal order but not minimum distances.

Example

Work it through

- Edges A-B, A-C, B-D, C-D; alphabetical neighbours: queue [A] → [B,C] → [C,D] → [D] → []. Discovery order A,B,C,D; D has distance two.

Sedgewick and Wayne: Undirected Graphs

DSA42 | DFS: finish a branch before returning

Mark a vertex, then recursively visit each unmarked neighbour before continuing with later neighbours.

An explicit stack can store pending work instead.

Both implement latest-pending-work first; a queue would change that discipline.

Marking prevents infinite repetition around graph cycles.

Using the same graph and alphabetical neighbours, recursive DFS can discover A,B,D,C. Its path is a valid reachability witness but not necessarily shortest.

Adjacency-list cost is O(V+E), with O(V) visited state and possible stack depth.

DFS supports cycle and connectivity analysis; a directed cycle test distinguishes active recursion vertices from already completed ones.

Data Structure and Algorithms module

DSA43 | Shortest paths and spanning trees solve different objectives

For nonnegative weights, standard Dijkstra initializes source distance zero and others infinity, repeatedly finalizes the unsettled vertex with smallest distance, and relaxes each outgoing edge: dist[v]=min(dist[v],dist[u]+w(u,v)).

A min-priority queue supports the next choice.

BFS is enough when every edge has equal cost; negative edges require another justified method, such as Bellman-Ford.

An MST minimizes the total cost of connecting all vertices of a connected undirected weighted graph.

Prim adds the cheapest edge crossing from its current tree to an outside vertex.

Kruskal processes edges by increasing weight and rejects those joining vertices already connected.

An MST need not give shortest source-to-vertex routes, and equal weights can permit several MSTs.

Example

Work it through

- Triangle weights AB=1, BC=2, AC=10: Prim or Kruskal selects AB,BC, total 3. Dijkstra from A gets B=1 and improves C from 10 to 3 through B.

Sedgewick and Wayne: Shortest Paths

Sedgewick and Wayne: Minimum Spanning Trees

DSA44 | Recognize two other tested algorithms through their rules

Kadane’s algorithm finds a contiguous maximum-sum subarray.

For a required nonempty answer, initialize ending=best=A[0]; update ending=max(x,ending+x), then best=max(best,ending).

It is linear time with constant working storage.

Initializing best=0 instead silently allows an empty answer and fails the nonempty all-negative case.

Knuth-Morris-Pratt (KMP) finds a pattern inside text.

Preprocess the pattern’s longest proper prefix that is also a suffix.

After mismatch, fall back using those prefix lengths rather than restarting comparisons from the beginning; the text pointer need not move backward.

Pattern preprocessing plus scanning takes O(pattern length+text length).

Example

Work it through

- For [-2,3,-1,4,-5], Kadane ending values are -2,3,2,6,1, so best is 6 from [3,-1,4]. Pattern ABAB has prefix lengths [0,0,1,2].

University of Alberta: Advanced Algorithms Lecture2

Jayadev Misra, University of Texas: Knuth-Morris-Pratt Algorithm

DSA45 | State trace data and representation assumptions

A trace needs the complete operation sequence, initial state, capacity and output orientation.

State these assumptions before calculating.

Evaluate all choices against the rules; a selected answer does not make other valid choices false.

Specify whether a question asks which structure can implement something or which is best under stated costs.

ADTs do not force one representation, and serial-access memory is not inherently LIFO.

A heap file is an unordered database record file, unrelated to a binary heap: insertion uses available space, scanning can find records, and ordered output requires sorting or an index.

It need not insert at the beginning.

Data Structure and Algorithms module

UC Berkeley CS186: Disk and Files

DSA-S01 | Best, average and worst compare equal-size inputs

Fix the input size before comparing cases. Best-case cost is the minimum resource use over legal inputs of that size; worst-case cost is the maximum. Average-case cost averages costs using a stated input distribution. These three input-case functions can each be described using upper, lower or tight asymptotic bounds; the case name does not select the bound notation.

For successful binary search with all stored positions equally likely, average probes have order Theta(log n). The probe decisions form a binary tree: only a bounded number of positions occur in shallow levels, while a constant fraction require logarithmic depth. Interval halving also gives the logarithmic upper bound. A distribution concentrating on an immediate midpoint hit would give a different average.

NIST Dictionary of Algorithms and Data Structures - Best case

NIST Dictionary of Algorithms and Data Structures - Worst case

NIST Dictionary of Algorithms and Data Structures - Average case

Sedgewick and Wayne - Algorithms, binary-search symbol-table average costs

DSA-S02 | Queues for deferred interrupt work

An interrupt service routine may need to record work quickly and defer its completion. Operating systems can queue that deferred work for later processing; Windows deferred procedure calls are one concrete example. This is a queue application, but it does not imply that every interrupt system uses a single FIFO service policy.

Microsoft - Introduction to DPC objects

DSA-S03 | Linked nodes and multiway tree branching

A linked binary-tree node stores a payload and two child pointers, one for the left child and one for the right child. A null child pointer means that child is absent; a root pointer gives access to the whole tree. In a dynamically allocated version, insertion can allocate a node and attach it through the appropriate child link.

A B-tree is a balanced multiway search tree. Its allowed branching factor can exceed two, so a general B-tree is not the binary-tree family. Balanced describes a height discipline; it does not by itself restrict every node to two children.

Data Structure and Algorithms module · page 9

Data Structure and Algorithms module · page 5

NIST Dictionary of Algorithms and Data Structures - B-tree

DSA-S04 | Recognize recursive divide-and-conquer structure

Divide and conquer reduces a problem to smaller instances of the same kind, solves the required smaller instances and obtains the original result from them. Mergesort recursively sorts two halves and merges; quicksort partitions and recursively sorts the resulting ranges. This course also classifies binary search as divide and conquer: an ordered comparison selects the only half that can contain the target. This is different from simply repeating adjacent swaps, choosing successive minima, or making stable LSD digit passes.

Data Structure and Algorithms module · page 11

Data Structure and Algorithms module · page 3