MockAttempt

TECH ACADEMY • OFFICIAL COURSE SYLLABUS BROCHURE

MockAttempt Academy Pvt. Ltd. • Live Architect Mentorship, AI Assessments & Client Production

Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: DSAJ • **Updated:** 01 Nov 2026
https://mockattempt.com/courses/dsa-with-java-training

DSAJ • DATA STRUCTURES, ALGORITHMS & FAANG CODING PREP | **BEGINNER TO COMPETITIVE & PRODUCT-COMPANY DSA (JAVA)**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

DSA with Java Training

Crack FAANG & top product-company coding rounds with Java 21 DSA: Big-O Asymptotic Analysis, Two-Pointer Arrays, Singly/Doubly Linked Lists, ArrayDeque Monotonic Stacks, Circular/Deque Windows, HashMap Internals, Binary Search, Merge/Quick Sort, BST/AVL Trees, PriorityQueue Heaps, BFS/DFS & Kruskal/Prim MST — with 10×50 MCQ Mock Tests per topic and Spoken Voice + MockAttempt AI Interviews at ₹4,999.

Duration: 15–17 Weeks • Live + Hands-On Projects (222 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

Fixed Non-Negotiable Tuition Fee: Base Fee **₹4,999** + 18% GST (₹900) = **Total Incl. GST: ₹5,899**

Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots): Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 19–21 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 21–23 Weeks Weekend Track)

Core Technologies & Engineering Stack: Java 21 LTS (Records & Generics) • Big-O, Amortized & Space Complexity • Arrays, Two Pointers & Linked Lists • ArrayDeque Stacks, Circular Queues & Deques • HashMap (TreeBin), Binary Search & Sorting • Binary Trees, BST, AVL & Binary Heaps • Graph BFS/DFS, Union-Find & Kruskal/Prim MST

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Derive tight Big-O, Big-Omega, Big-Theta, and amortized bounds while understanding JVM CPU cache locality vs pointer-chasing heap overhead
2. Solve product-company interview patterns across Two Pointers, Fast/Slow Linked Lists, Monotonic Stacks, Sliding Window Deques, and Binary Search on Answer
3. Implement self-balancing AVL rotations, O(N) Floyd Heapify / Top-K PriorityQueues, iterative/recursive Graph BFS/DFS, and Union-Find / Prim Minimum Spanning Trees
4. Clear 200 Timed Mock Exams (10,000 Java DSA MCQs) and 200 Spoken Voice + MockAttempt AI Coding & Algorithmic Viva Rounds
5. Asymptotic Time & Auxiliary Space analysis (`Big-O`, amortized complexity, and recursion stack trade-offs).
6. Core linear & hierarchical structures: Arrays, Two Pointers, Sliding Window, Linked Lists, Stacks, Monotonic Queues, Trees, Tries, and Heaps.
7. Graph & Optimization mastery: BFS/DFS, Topological Sort, Union-Find, Dijkstra, Greedy proofs, and Dynamic Programming.
8. Java Collections internals (`HashMap` collision trees, `PriorityQueue`, `ArrayDeque`) and clean bug-free coding speed.

Eligibility & Prerequisites: Basic familiarity with Java syntax (primitive types, loops, methods, classes) — generics, collections, and recursion are built up systematically • Any laptop/PC with JDK 21 installed and IntelliJ IDEA Community / VS Code • No prior DSA with Java industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED MODULE & TOPIC-WISE SYLLABUS (16 MODULES • 42 LIVE TOPICS)

MODULE 1 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Algorithmic Thinking, JVM Memory Layout & Asymptotic Complexity (`O`, `Ω`, `Θ`)

Understand Abstract Data Types (ADTs) vs concrete Java data structures, primitive contiguous arrays vs boxed heap objects, and rigorous Big-O, Big-Omega, Big-Theta, and Amortized complexity analysis.

Objectives: ✓ Choose optimal data structures based on problem constraints (`N <= 10^5` demands `O(N log N)` or `O(N)` — never `O(N^2)`) • ✓ Compare contiguous primitive `int[]` CPU L1/L2 cache locality against boxed `Integer` object header overhead on the JVM • ✓ Compute worst-case, best-case, average-

case, and amortized complexities across nested loops and halving recurrences

1.1 DSA Foundations in Java — ADTs vs Implementations, Primitive Cache Locality & Judge Constraints [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply DSA Foundations in Java — ADTs vs Implementations, Primitive Cache Locality & Judge Constraints in real DSA with Java systems — specifically: Why...

1.2 Asymptotic Notation — Big-O (`O`), Big-Omega (`Ω`), Big-Theta (`Θ`) & Amortized Analysis [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Asymptotic Notation — Big-O (`O`), Big-Omega (`Ω`), Big-Theta (`Θ`) & Amortized Analysis in real DSA with Java systems — specifically: Formal grow...

Module 1 Hands-On Capstone Project (100 Marks): Constraint-to-Complexity Feasibility Estimator & Amortized Dynamic Array Profiler — Build a Java 21 benchmark & constraint analyzer that maps input size `N` (`10`, `20`, `10^3`, `10^5`, `10^7`) to maximum non-TLE complexity classes under a `10^8` operations/sec judge budget...

MODULE 2 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Contiguous Arrays (Two Pointers, Prefix Sums) & Singly Linked Lists

Exploit `O(1)` random-access contiguous arrays with in-place reversal, Prefix Sums, and Two Pointers, and master pointer manipulation in Singly Linked Lists (Dummy Head, Fast/Slow Tortoise-Hare, In-Place Reversal).

Objectives: ✓ Rotate arrays in `O(N)` time and `O(1)` auxiliary space via the 3-reversal trick • ✓ Manipulate Singly Linked Lists without `NullPointerException` using sentinel Dummy Nodes (`new ListNode(0)`) • ✓ Detect cycles (`Floyd's Tortoise & Hare`), find middle nodes, and reverse linked lists iteratively in `O(1)` extra space

2.1 Arrays in Java — `base + i * stride` Random Access, 3-Step Rotation & Two-Pointer Technique [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Arrays in Java — `base + i * stride` Random Access, 3-Step Rotation & Two-Pointer Technique in real DSA with Java systems — specifically: Understa...

2.2 Singly Linked List — Pointer Rewiring, Sentinel Dummy Head, Floyd's Cycle & In-Place Reversal [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Singly Linked List — Pointer Rewiring, Sentinel Dummy Head, Floyd's Cycle & In-Place Reversal in real DSA with Java systems — specifically: Master...

Module 2 Hands-On Capstone Project (100 Marks): In-Place Array Rotation Engine & `O(1)` Space Singly Linked List Palindrome Checker — Implement an `O(N)` time, `O(1)` auxiliary-space `k`-step right array rotator alongside an `O(N)` time, `O(1)` space Singly Linked List palindrome verifier using Fast/Slow midpoint discovery...

MODULE 3 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Doubly Linked Lists (`O(1)` LRU Eviction) & Monotonic Stacks (`ArrayDeque`)

Perform `O(1)` arbitrary node splicing with bidirectional sentinel Doubly Linked Lists (powering LRU Caches) and solve Next Greater Element, Balanced Parentheses, and Histogram problems using `ArrayDeque` Stacks.

Objectives: ✓ Use `head` and `tail` sentinel nodes in a Doubly Linked List to eliminate every `null` check during `remove(node)` and `addFirst(node)` • ✓ Explain why `Deque<T> stack = new ArrayDeque<>()` MUST replace legacy `java.util.Stack<T>` in modern Java interviews • ✓ Solve Next Greater Element / Daily Temperatures / Largest Rectangle in Histogram in `O(N)` amortized time with a Monotonic Stack

3.1 Doubly Linked List — `prev`/`next` Sentinel Splicing, `O(1)` Node Deletion & LRU Cache Design [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Doubly Linked List — `prev`/`next` Sentinel Splicing, `O(1)` Node Deletion & LRU Cache Design in real DSA with Java systems — specifically: Why st...

3.2 Stack in Java — LIFO `ArrayDeque` vs Legacy `Stack`, Balanced Brackets & `O(N)` Monotonic Stack [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Stack in Java — LIFO `ArrayDeque` vs Legacy `Stack`, Balanced Brackets & `O(N)` Monotonic Stack in real DSA with Java systems — specifically: Impl...

Module 3 Hands-On Capstone Project (100 Marks): `O(1)` LRU Cache (HashMap + Sentinel Doubly Linked List) & Monotonic Next-Greater Engine — Implement an interview-standard `LRUCache(capacity)` supporting `get(key)` and `put(key, val)` in strict `O(1)` time by pairing a `HashMap<Integer, DNode>` with a sentinel-guarded Doubly Lin...

MODULE 4 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Ring Buffers (`CircularQueue`) & Double-Ended Queues (`Deque` Sliding Window Max)

Eliminate wasted front capacity in array queues via modulo wrap-around (`CircularQueue`), and master Double-Ended Queues (`Deque`) to solve `O(N)` Sliding Window Maximum and 0-1 BFS.

Objectives: ✓ Implement fixed-capacity Ring Buffers (`(tail + 1) % cap` or explicit `size` counter) with zero element shifting • ✓ Understand how Java's `ArrayDeque` uses a power-of-two circular array with bitmask `(tail + 1) & (elements.length - 1)` • ✓ Solve Sliding Window Maximum in `O(N)` linear time (beating `O(N log K)` heap solutions) using a Monotonic Decreasing `Deque`

4.1 Circular Queue (Ring Buffer) — Modulo Wrap-Around `(idx + 1) % k` & Full vs Empty Disambiguation [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Circular Queue (Ring Buffer) — Modulo Wrap-Around `(idx + 1) % k` & Full vs Empty Disambiguation in real DSA with Java systems — specifically: Why...

4.2 Double-Ended Queue (`Deque`) — `offerFirst`/`offerLast`, 0-1 BFS & `O(N)` Sliding Window Maximum [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Double-Ended Queue (`Deque`) — `offerFirst`/`offerLast`, 0-1 BFS & `O(N)` Sliding Window Maximum in real DSA with Java systems — specifically: Pus...

Module 4 Hands-On Capstone Project (100 Marks): Lock-Free Style Ring Buffer Queue & `O(N)` Sliding Window Maximum Analytics Engine — Build a bounded `CircularQueue` supporting `enQueue`, `deQueue`, `Front`, `Rear`, `isEmpty`, and `isFull` in strict `O(1)` time using a `size` counter, alongside an `O(N)` Monotonic `Deque`...

MODULE 5 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Java `HashMap` Internals (TreeBins, Load Factor) & Binary Search on Answer

Master Java 8+ `HashMap` internals (XOR hash spreading, load factor `0.75`, collision chaining -> Red-Black `TreeNode` at threshold 8) and write bug-free lower/upper-bound Binary Search & Binary Search on Answer.

Objectives: ✓ Explain Java `HashMap` bucket indexing `(n - 1) & hash`, load factor `0.75`, and why `TREEIFY_THRESHOLD = 8` prevents `O(N)` HashDoS attacks • ✓ Uphold the `equals()` / `hashCode()` contract using immutable keys (`record` / `String` / `Integer`) • ✓ Write overflow-safe `mid = low + (high - low) / 2` boundary searches and monotonic **Binary Search on Answer** (`Koko Eating Bananas` / `Split Array`)

5.1 Hash Table & Java `HashMap` — `(h ^ (h >>> 16))`, Load Factor `0.75` & Red-Black `TreeNode` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Hash Table & Java `HashMap` — `(h ^ (h >>> 16))`, Load Factor `0.75` & Red-Black `TreeNode` in real DSA with Java systems — specifically: How `Has...

5.2 Binary Search — Overflow-Safe Midpoint, Lower/Upper Bound & Binary Search on Answer [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Binary Search — Overflow-Safe Midpoint, Lower/Upper Bound & Binary Search on Answer in real DSA with Java systems — specifically: Go far beyond si...

Module 5 Hands-On Capstone Project (100 Marks): Prefix-Sum Subarray Target Counter (`HashMap`) & Monotonic Binary Search Capacity Solver — Solve Subarray Sum Equals `K` (including negative integers where two-pointer sliding window fails!) in `O(N)` using a prefix-sum frequency `HashMap`, paired with an `O(N log(sum))` Binary Se...

MODULE 6 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Divide-and-Conquer Sorting — Stable `Merge Sort` & Partitioned `Quick Sort`

Master guaranteed `O(N log N)` stable Merge Sort (plus Inversion Counting) and cache-friendly in-place Quick Sort (Lomuto, Hoare, 3-Way Dutch National Flag) alongside why JDK uses Dual-Pivot Quicksort vs TimSort.

Objectives: ✓ Implement stable Merge Sort (`<=` left-tie rule) with a single preallocated scratch buffer and count array inversions in `O(N log N)` • ✓ Defeat `O(N^2)` adversarial sorted/all-equal Quick Sort inputs using randomized pivots and Dijkstra's 3-Way Partitioning • ✓ Explain why `Arrays.sort(int[])` uses Dual-Pivot Quicksort while `Arrays.sort(Object[])` uses **TimSort** (stability requirement!)

6.1 Merge Sort — Guaranteed `O(N log N)` Divide & Conquer, Stability (`<=`) & Inversion Counting [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Merge Sort — Guaranteed `O(N log N)` Divide & Conquer, Stability (`<=`) & Inversion Counting in real DSA with Java systems — specifically: Recursi...

6.2 Quick Sort — Lomuto vs Hoare vs 3-Way Dutch Flag Partitioning & Randomized Pivot Safety [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Quick Sort — Lomuto vs Hoare vs 3-Way Dutch Flag Partitioning & Randomized Pivot Safety in real DSA with Java systems — specifically: Understand w...

Module 6 Hands-On Capstone Project (100 Marks): `O(N log N)` Inversion Counter (Merge Sort) & 3-Way Quickselect `K`-th Largest Engine — Implement a stable Merge Sort pass counting exact out-of-order pairs `(i < j && a[i] > a[j])` in `O(N log N)`, paired with a randomized 3-Way Dutch National Flag Quickselect finding the `k`-...

MODULE 7 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Hierarchical Trees — Binary Tree Traversals, Diameter/LCA & Binary Search Trees (BST)

Master recursive and iterative DFS/BFS Binary Tree algorithms (Preorder, Inorder, Postorder, Level-Order, Diameter, Lowest Common Ancestor) and Binary Search Tree invariants (`(min, max)` validation, Hibbard deletion, `k`-th smallest).

Objectives: ✓ Write recursive and `ArrayDeque`-backed iterative Preorder, Inorder, Postorder, and Level-Order BFS tree traversals • ✓ Compute tree height, diameter, and Lowest Common Ancestor (LCA) in a single `O(N)` bottom-up postorder pass • ✓ Validate BSTs strictly via `(Long min, Long max)` subtree range bounds (never just local `left < node < right`!) and implement Hibbard 2-child successor deletion

7.1 Binary Tree — DFS Traversals, Level-Order BFS, `O(N)` Postorder Diameter & Lowest Common Ancestor [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Binary Tree — DFS Traversals, Level-Order BFS, `O(N)` Postorder Diameter & Lowest Common Ancestor in real DSA with Java systems — specifically: Ma...

7.2 Binary Search Tree (BST) — Global Subtree Bounds, Sorted Inorder, Hibbard Delete & `TreeMap` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Binary Search Tree (BST) — Global Subtree Bounds, Sorted Inorder, Hibbard Delete & `TreeMap` in real DSA with Java systems — specifically: Exploit...

Module 7 Hands-On Capstone Project (100 Marks): Single-Pass Binary Tree Diameter/LCA Solver & Range-Bounded BST Validator — Implement a Binary Tree analyzer computing exact Diameter + Lowest Common Ancestor in `O(N)` time, paired with a strict `(long min, long max)` BST validator that handles `Integer.MIN_VALUE`...

MODULE 8 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Self-Balancing `AVL Trees` (LL/RR/LR/RL Rotations) & Binary `Heaps` (`PriorityQueue`)

Prevent `O(N)` skewed-BST degeneration permanently with height-balanced **AVL Trees** (`|bf| <= 1`, 4 rotation cases) and master array-backed **Binary Heaps** (`O(N)` Floyd `buildHeap`, Java `PriorityQueue`, Top-`K`, Two-Heap Median).

Objectives: ✓ Compute AVL Balance Factors `bf = height(left) - height(right)` and apply LL (Right), RR (Left), LR (Left-Right), and RL (Right-Left) `O(1)` rotations • ✓ Index Complete Binary Trees inside flat arrays (`parent = (i - 1) / 2`, `left = 2*i + 1`, `right = 2*i + 2`) with zero object-node overhead • ✓ Solve Top-`K` Elements in `O(N log K)` and Streaming Median in `O(log N)` using Java `PriorityQueue` Min/Max Heaps

8.1 AVL Tree — Balance Factor `{-1, 0, +1}`, LL/RR/LR/RL `O(1)` Rotations & `1.44 log2(N)` Height Bound [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply AVL Tree — Balance Factor `{-1, 0, +1}`, LL/RR/LR/RL `O(1)` Rotations & `1.44 log2(N)` Height Bound in real DSA with Java systems — specifically:...

8.2 Binary Heap & Java `PriorityQueue` — `O(N)` Floyd Heapify, Top-`K` Min-Heap & `Integer.compare` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Binary Heap & Java `PriorityQueue` — `O(N)` Floyd Heapify, Top-`K` Min-Heap & `Integer.compare` in real DSA with Java systems — specifically: Unde...

Module 8 Hands-On Capstone Project (100 Marks): Self-Balancing AVL Tree Dictionary & Two-Heap Streaming Median Finder — Implement an AVL Tree supporting sequential sorted insertions `1..15` while maintaining strict `|bf| <= 1` height `<= 4`, paired with a `MedianFinder` maintaining a Max-Heap (lower half) and...

MODULE 9 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Graph Traversals — Breadth-First Search (`BFS`) & Depth-First Search (`DFS` / Topological Sort)

Represent sparse graphs cleanly with Adjacency Lists (`List<List<Integer>>`), compute unweighted shortest paths & multi-source wave propagation via **BFS**, and detect cycles, connected components & Topological Order via **DFS**.

Objectives: ✓ Choose Adjacency Lists (`O(V + E)` space) over Adjacency Matrices (`O(V^2)` memory explosion when `V = 10^5`) • ✓ Mark vertices `visited` AT ENQUEUE TIME (never wait until dequeue!) in BFS to prevent exponential duplicate queue insertions • ✓ Detect directed cycles using 3-color DFS states (`0 = unvisited`, `1 = visiting/on-stack`, `2 = visited`) and compute Topological Sort

9.1 Breadth-First Search (BFS) — Unweighted Shortest Path, Multi-Source BFS & Enqueue-Mark Rule [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Breadth-First Search (BFS) — Unweighted Shortest Path, Multi-Source BFS & Enqueue-Mark Rule in real DSA with Java systems — specifically: Explore...

9.2 Depth-First Search (DFS) — Connected Components, 3-Color Directed Cycle Detection & Topological Sort [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Depth-First Search (DFS) — Connected Components, 3-Color Directed Cycle Detection & Topological Sort in real DSA with Java systems — specifically:...

Module 9 Hands-On Capstone Project (100 Marks): Multi-Source Grid Shortest-Wave BFS & Directed Course-Prerequisite Cycle Detector — Implement a Multi-Source BFS grid solver (`Rotting Oranges` / `01 Matrix` wave distance) marking cells visited on enqueue, paired with a 3-color Directed Graph DFS cycle detector and Topolog...

MODULE 10 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Minimum Spanning Trees — `Kruskal's` (Disjoint Set Union) & `Prim's` (`PriorityQueue`)

Connect all `V` vertices of a weighted undirected graph at minimum total edge cost (`V - 1` edges, zero cycles) using edge-greedy **Kruskal's Algorithm + Union-Find (`O(α(V))`)** and vertex-cut **Prim's Algorithm**.

Objectives: ✓ Implement Disjoint Set Union (DSU / Union-Find) with both **Path Compression** and **Union by Rank/Size** for `O(α(V))` inverse-Ackermann speed • ✓ Execute Kruskal's MST in `O(E log E)` and detect disconnected graphs when `edgesAdded < V - 1` • ✓ Execute Prim's MST using a Min-Heap `PriorityQueue` (`O(E log V)`) and know when dense graphs (`E ≈ V^2`) favor unheaped Prim (`O(V^2)`)

10.1 Kruskal's Algorithm & Disjoint Set Union — Cut Property, Path Compression & `O(E log E)` MST [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Kruskal's Algorithm & Disjoint Set Union — Cut Property, Path Compression & `O(E log E)` MST in real DSA with Java systems — specifically: Sort al...

10.2 Prim's Algorithm — Vertex-Cut Min-Heap Growth, Lazy `inMst` Filtering & Dense vs Sparse Choice [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Prim's Algorithm — Vertex-Cut Min-Heap Growth, Lazy `inMst` Filtering & Dense vs Sparse Choice in real DSA with Java systems — specifically: Grow...

Module 10 Hands-On Capstone Project (100 Marks): Minimum-Cost Fiber Network Planner Comparing Kruskal (DSU) & Prim (Min-Heap) — Build both Kruskal's (Union-Find with Path Compression + Union by Size) and Prim's (`PriorityQueue` cut expansion) MST solvers verifying identical minimum total cable cost and returning `-1`...

MODULE 11 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Advanced Graph Algorithms: Shortest Paths, Topo Sort, MST & Union-Find

Production-grade DSA with Java Training engineering module covering Advanced Graph Algorithms: Shortest Paths, Topo Sort, MST & Union-Find with hands-on labs and capstone build.

Objectives: ✓ BFS/DFS State Modeling, Cycle Detection & Kahn's Topological Sort — Purpose: Solve course-schedule, build-system order, and grid shortest-path interview problems. • ✓ Dijkstra (PriorityQueue), 0-1 BFS & Bellman-Ford Weighted Paths — Purpose: Compute minimum-cost network and delivery routes with optimal O((V+E) log V) Java code. • ✓ Disjoint Set Union (Path Compression + Union by Rank) & Kruskal MST — Purpose: Answer dynamic connectivity and clustering queries in near-constant O(alpha(N)) time. • ✓ Multi-Source BFS & Bipartite Graph Coloring — Purpose: Tackle rotten-oranges, flood-spread, and conflict-partitioning FAANG graph rounds.

11.1 BFS/DFS State Modeling, Cycle Detection & Kahn's Topological Sort [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply BFS/DFS State Modeling, Cycle Detection & Kahn's Topological Sort in real DSA with Java systems — specifically: Solve course-schedule, build-syste...

11.2 Dijkstra (PriorityQueue), 0-1 BFS & Bellman-Ford Weighted Paths [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Dijkstra (PriorityQueue), 0-1 BFS & Bellman-Ford Weighted Paths in real DSA with Java systems — specifically: Compute minimum-cost network and del...

11.3 Disjoint Set Union (Path Compression + Union by Rank) & Kruskal MST [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Disjoint Set Union (Path Compression + Union by Rank) & Kruskal MST in real DSA with Java systems — specifically: Answer dynamic connectivity and...

11.4 Multi-Source BFS & Bipartite Graph Coloring [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Multi-Source BFS & Bipartite Graph Coloring in real DSA with Java systems — specifically: Tackle rotten-oranges, flood-spread, and conflict-partit...

Module 11 Hands-On Capstone Project (100 Marks): Low-Latency Route Planner & Dependency Cycle Detector in Java — Design, implement, test, and document 'Low-Latency Route Planner & Dependency Cycle Detector in Java' applying every production concept from Module 11 (Advanced Graph Algorithms: Shortest Pa...

MODULE 12 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Dynamic Programming Mastery: 1D/2D, Knapsack, Intervals, Trees & Bitmask

Production-grade DSA with Java Training engineering module covering Dynamic Programming Mastery: 1D/2D, Knapsack, Intervals, Trees & Bitmask with hands-on labs and capstone build.

Objectives: ✓ Top-Down Memoization -> Bottom-Up Tabulation -> Space Optimization — Purpose: Convert exponential O(2^N) recursion into crisp O(N*W) DP systematically every time. • ✓ 0/1 & Unbounded Knapsack, Subset Sum & Coin-Change Pattern Family — Purpose: Recognize capacity-allocation problems immediately in OA coding assessments. • ✓ LCS, Edit Distance, LIS (O(N log N) Patience) & Palindromic DP — Purpose: Master string/subsequence DP asked across Amazon, Google, and Microsoft interviews. • ✓ Interval DP, DP on Trees & State-Machine Stock Trading Problems — Purpose: Handle cooldown/transaction-capped trading and tree diameter/robber rounds confidently.

12.1 Top-Down Memoization -> Bottom-Up Tabulation -> Space Optimization [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Top-Down Memoization -> Bottom-Up Tabulation -> Space Optimization in real DSA with Java systems — specifically: Convert exponential O(2^N) recurs...

12.2 0/1 & Unbounded Knapsack, Subset Sum & Coin-Change Pattern Family [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply 0/1 & Unbounded Knapsack, Subset Sum & Coin-Change Pattern Family in real DSA with Java systems — specifically: Recognize capacity-allocation prob...

12.3 LCS, Edit Distance, LIS (O(N log N) Patience) & Palindromic DP [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply LCS, Edit Distance, LIS (O(N log N) Patience) & Palindromic DP in real DSA with Java systems — specifically: Master string/subsequence DP asked ac...

12.4 Interval DP, DP on Trees & State-Machine Stock Trading Problems [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Interval DP, DP on Trees & State-Machine Stock Trading Problems in real DSA with Java systems — specifically: Handle cooldown/transaction-capped t...

Module 12 Hands-On Capstone Project (100 Marks): Algorithmic Portfolio Optimizer & Sequence Alignment Suite — Design, implement, test, and document 'Algorithmic Portfolio Optimizer & Sequence Alignment Suite' applying every production concept from Module 12 (Dynamic Programming Mastery: 1D/2D, Knaps...

MODULE 13 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Tries, Segment Trees, Fenwick Trees (BIT) & String Pattern Matching

Production-grade DSA with Java Training engineering module covering Tries, Segment Trees, Fenwick Trees (BIT) & String Pattern Matching with hands-on labs and capstone build.

Objectives: ✓ Prefix Tries & Bitwise XOR Tries — Purpose: Power O(L) search autocomplete, spellcheck, and maximum-XOR subarray queries. • ✓ Fenwick Tree (BIT) & Segment Tree with Point Update / Range Query — Purpose: Maintain live prefix sums and range minimums across millions of streaming updates. • ✓ Rolling Hash (Rabin-Karp), KMP Prefix Function & Z-Algorithm — Purpose: Match substrings in strict linear O(N + M) time without worst-case quadratic TLEs. • ✓ Sweep-Line Intervals, Difference Arrays & Coordinate Compression — Purpose: Count peak concurrent meetings/servers cleanly over large timestamp ranges.

13.1 Prefix Tries & Bitwise XOR Tries [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Prefix Tries & Bitwise XOR Tries in real DSA with Java systems — specifically: Power O(L) search autocomplete, spellcheck, and maximum-XOR subarra...

13.2 Fenwick Tree (BIT) & Segment Tree with Point Update / Range Query [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Fenwick Tree (BIT) & Segment Tree with Point Update / Range Query in real DSA with Java systems — specifically: Maintain live prefix sums and rang...

13.3 Rolling Hash (Rabin-Karp), KMP Prefix Function & Z-Algorithm [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Rolling Hash (Rabin-Karp), KMP Prefix Function & Z-Algorithm in real DSA with Java systems — specifically: Match substrings in strict linear O(N +...

13.4 Sweep-Line Intervals, Difference Arrays & Coordinate Compression [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Sweep-Line Intervals, Difference Arrays & Coordinate Compression in real DSA with Java systems — specifically: Count peak concurrent meetings/serv...

Module 13 Hands-On Capstone Project (100 Marks): Autocomplete Prefix Engine & Live Range-Query Analytics in Java — Design, implement, test, and document 'Autocomplete Prefix Engine & Live Range-Query Analytics in Java' applying every production concept from Module 13 (Tries, Segment Trees, Fenwick Trees...

MODULE 14 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Greedy Proofs, Monotonic Queues, Concurrency DSA & Machine-Coding LLD

Production-grade DSA with Java Training engineering module covering Greedy Proofs, Monotonic Queues, Concurrency DSA & Machine-Coding LLD with hands-on labs and capstone build.

Objectives: ✓ Monotonic Stack & Deque (Sliding Window Max, Largest Histogram) — Purpose: Turn brute-force O(N*K) window/span problems into single-pass O(N) solutions. • ✓ Heap Design Patterns: Two-Heaps Median, K-Way Merge & Top-K Frequent — Purpose: Stream live medians and merge sorted shards using Java PriorityQueue. • ✓ Doubly-Linked + HashMap O(1) LRU/LFU Cache & Token-Bucket Rate Limiter — Purpose: Ace the most frequent SDE-1/SDE-2 low-level machine-coding interview rounds. • ✓ Complexity Trade-Off Communication & Edge-Case Dry-Run Protocol — Purpose: Verbalize invariants, Big-O bounds, and integer-overflow guards like a top 5% candidate.

14.1 Monotonic Stack & Deque (Sliding Window Max, Largest Histogram) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Monotonic Stack & Deque (Sliding Window Max, Largest Histogram) in real DSA with Java systems — specifically: Turn brute-force O(N*K) window/span...

14.2 Heap Design Patterns: Two-Heaps Median, K-Way Merge & Top-K Frequent [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Heap Design Patterns: Two-Heaps Median, K-Way Merge & Top-K Frequent in real DSA with Java systems — specifically: Stream live medians and merge s...

14.3 Doubly-Linked + HashMap O(1) LRU/LFU Cache & Token-Bucket Rate Limiter [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Doubly-Linked + HashMap O(1) LRU/LFU Cache & Token-Bucket Rate Limiter in real DSA with Java systems — specifically: Ace the most frequent SDE-1/S...

14.4 Complexity Trade-Off Communication & Edge-Case Dry-Run Protocol [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Complexity Trade-Off Communication & Edge-Case Dry-Run Protocol in real DSA with Java systems — specifically: Verbalize invariants, Big-O bounds,...

Module 14 Hands-On Capstone Project (100 Marks): Thread-Safe LRU/Rate-Limiter Machine-Coding Round in Java — Design, implement, test, and document 'Thread-Safe LRU/Rate-Limiter Machine-Coding Round in Java' applying every production concept from Module 14 (Greedy Proofs, Monotonic Queues, Concurren...

MODULE 15 OF 16 • 14 LIVE + LAB HOURS • 3 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Advanced Graph, Trie, Segment Tree & Bit-Manipulation Patterns in Java

Self-explanatory list-and-bullet production mastery module covering Advanced Graph, Trie, Segment Tree & Bit-Manipulation Patterns in Java in DSA with Java Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Prefix Tries, Bitwise XOR Tries & String Rolling-Hash / KMP Pattern Matching — Purpose: Solve dictionary search, IP routing, and linear-time substring problems in O(L) / O(N) time. • ✓ Disjoint Set Union (Union-Find with Path Compression), Bridges & Topological Cycles — Purpose: Detect dynamic network connectivity, dependency deadlocks, and minimum spanning trees rapidly. • ✓ Segment Trees, Fenwick (BIT) Trees & Monotonic Queue/Stack Optimization — Purpose: Drop range-sum/min updates from O(N) to O(log N) and sliding-window extrema to O(N). • ✓ Build, verify, and document 'Autocomplete Trie + Range-Query Segment Tree Benchmark Engine in Java 21' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Prefix Tries, Bitwise XOR Tries & String Rolling-Hash / KMP Pattern Matching [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Prefix Tries, Bitwise XOR Tries & String Rolling-Hash / KMP Pattern Matching in real DSA with Java systems — specifically: Solve dictionary search...

15.2 Disjoint Set Union (Union-Find with Path Compression), Bridges & Topological Cycles [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Disjoint Set Union (Union-Find with Path Compression), Bridges & Topological Cycles in real DSA with Java systems — specifically: Detect dynamic n...

15.3 Segment Trees, Fenwick (BIT) Trees & Monotonic Queue/Stack Optimization [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Segment Trees, Fenwick (BIT) Trees & Monotonic Queue/Stack Optimization in real DSA with Java systems — specifically: Drop range-sum/min updates f...

Module 15 Hands-On Capstone Project (100 Marks): Autocomplete Trie + Range-Query Segment Tree Benchmark Engine in Java 21 — Architect, implement, test, and document 'Autocomplete Trie + Range-Query Segment Tree Benchmark Engine in Java 21' applying every bullet-wise engineering concept from Module 15 (Advanced Gr...

MODULE 16 OF 16 • 14 LIVE + LAB HOURS • 3 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Competitive Edge-Case Mastery, Complexity Proofs & FAANG Java Coding Rounds

Self-explanatory list-and-bullet production mastery module covering Competitive Edge-Case Mastery, Complexity Proofs & FAANG Java Coding Rounds in DSA with Java Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Recognizing Patterns Instantly: Mapping Problem Constraints (N <= 10^5) to Exact Big-O Targets — Purpose: Choose between DP, Greedy, Binary Search on Answer, or Two Pointers within 2 minutes of reading a prompt. • ✓ Integer Overflow Guards, Off-by-One Invariants & Iterative Stack Conversions in Java — Purpose: Eliminate StackOverflowError and long-overflow WA failures on adversarial hidden test cases. • ✓ Communicating Brute-Force -> Optimal Trade-Offs Aloud in Live SDE Interviews — Purpose: Structure dry runs, complexity proofs, and clean modular Java helper methods under time pressure. • ✓ Build, verify, and document 'Timed 45-Minute Multi-Constraint Problem Solver & JUnit Stress-Tester' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Recognizing Patterns Instantly: Mapping Problem Constraints (N <= 10^5) to Exact Big-O Targets [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Recognizing Patterns Instantly: Mapping Problem Constraints (N <= 10^5) to Exact Big-O Targets in real DSA with Java systems — specifically: Choos...

16.2 Integer Overflow Guards, Off-by-One Invariants & Iterative Stack Conversions in Java [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Integer Overflow Guards, Off-by-One Invariants & Iterative Stack Conversions in Java in real DSA with Java systems — specifically: Eliminate Stack...

16.3 Communicating Brute-Force -> Optimal Trade-Offs Aloud in Live SDE Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Communicating Brute-Force -> Optimal Trade-Offs Aloud in Live SDE Interviews in real DSA with Java systems — specifically: Structure dry runs, com...

Module 16 Hands-On Capstone Project (100 Marks): Timed 45-Minute Multi-Constraint Problem Solver & JUnit Stress-Tester — Architect, implement, test, and document 'Timed 45-Minute Multi-Constraint Problem Solver & JUnit Stress-Tester' applying every bullet-wise engineering concept from Module 16 (Competitive Ed...

3. LIVE CLIENT PRODUCTION PROJECT, GLOBAL CERTIFICATIONS & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: AlgoSearch High-Speed Autocomplete, Graph Route Planner & LRU Cache Engine in Java • **Confirmed Role:** Java Algorithms, Data Structures & Low-Latency Systems Intern

Domain & Production Infra: High-Scale Search & Routing Core Platform Team (Java 21 LTS + Custom Trie/Heap/Graph/LRU Engines + JMH Benchmarks + JUnit 5)

- Implemented O(L) prefix autocomplete using compressed Tries and top-K min-heap frequency ranking in Java 21.
- Engineered weighted shortest-path routing with Dijkstra/A* priority queues and thread-safe O(1) LRU caching.
- Benchmarked time and heap-memory complexity across 1M+ synthetic inputs, eliminating GC bottlenecks.
- Documented Big-O trade-offs and passed rigorous edge-case stress tests modeled on FAANG/product interviews.

Aligned Global Vendor Certifications: **CCDSA-ADV** (Certified Data Structures & Algorithms Proficiency Exam) • **1Z0-830** (Oracle Certified Professional: Java SE 21 Developer)

Target Job Roles: Software Development Engineer — SDE-1 / SDE-2, Product Backend Engineer (Java), Algorithms & Core Platform Engineer, Member of Technical Staff (MTS)

Realistic Annual CTC Scope: **Product Fresher / SDE-1 (0–1 Yr):** ₹8.0 LPA – ₹18.0 LPA **Product SDE-2 (1–3 Yrs):** ₹16.0 LPA – ₹32.0 LPA **Senior SDE / Staff Product Architect (3–6+ Yrs):** ₹30.0 LPA – ₹60.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 98765 43210 • Web: https://mockattempt.com/courses/dsa-with-java-training

Try 2 Free Topic-Wise 50-MCQ Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.