# DSA PROFESSIONAL PROGRAM – 2025 EDITION

*"Think in logic, code with precision."*

**Market Demand Note**

Strong knowledge of data structures and algorithms (DSA) is a core requirement for technical roles in top companies. It’s essential for solving complex computational problems, optimizing performance, and clearing coding interviews.

Duration: 12 Weeks | Mode: Online/Offline

## 📘 DSA Professional Program — 2025 Edition

**Key Tools & Technologies:**
Python, Java, or C++ (learner's choice), LeetCode, HackerRank, Visual Studio Code, GitHub.

### Learning Objectives

By the end of this program, learners will be able to:

1. Understand and implement core data structures from scratch.
2. Solve problems using efficient algorithms.
3. Apply complexity analysis to select optimal solutions.
4. Build a strong foundation for interviews and competitive programming.
5. Translate real-world problems into structured solutions.

### Table of Contents

### Detailed Content

Week 1: Introduction to DSA

- Big O, Big Θ, Big Ω: Formal definitions, how to analyze worst/average/best cases.
- Problem-solving: Step-by-step approach, edge cases, dry runs.
- Pseudocode: Writing clear, language-agnostic logic before coding.

- Coding Standards: Readable, modular, commented code.
- Hands-on: Solve simple problems, analyze time/space complexity, write pseudocode.

Industry Relevance: Complexity analysis is a staple in coding interviews and system design.

---

Week 2: Arrays & Strings

- Array operations: Insert, delete, search, reverse, rotate.
- String manipulation: Substrings, palindromes, anagrams, encoding/decoding.
- Sliding window: Fixed/variable size, max sum, longest substring with K distinct.
- Hands-on: Implement in-place array operations, solve string puzzles, optimize with sliding window.

Real-World Use: Array/string ops are foundational for data processing, parsing, and competitive coding.

---

Week 3: Linked Lists

- Singly linked lists: Traversal, insertion, deletion, cycle detection.
- Doubly linked lists: Advantages, operations.
- Circular linked lists: Use cases (e.g., round-robin scheduling).
- Hands-on: Implement all variants, solve problems like merge two sorted lists, detect cycle.

Interview Focus: Linked lists test pointer manipulation and memory management.

---

Week 4: Stacks & Queues

- Stack operations: Push, pop, peek, isEmpty, applications (parsing, undo/redo).
- Infix-to-postfix: Shunting-yard algorithm, evaluation.
- Queue types: Simple, circular, deque, priority queue (min/max heap intro).
- Hands-on: Build a stack-based calculator, simulate a call center with queues.

System Design: Stacks/queues underpin OS schedulers, caches, and event systems.

---

Week 5: Recursion & Backtracking

- Recursion basics: Base case, recursive case, call stack visualization.
- Tail vs. head recursion: Optimization, language support.
- Backtracking: Generate all subsets, permutations, N-Queens, Sudoku solver.
- Hands-on: Solve classic problems, compare iterative vs. recursive solutions.

Why It Matters: Recursion is key for tree/graph traversals, DP, and combinatorial problems.

---

Week 6: Trees

- Binary trees: Properties, traversals (pre/in/post-order, level-order).
- BST: Insert, delete, search, validate BST.
- Heap/Priority Queue: Insert, extract-min/max, heapify.
- Hands-on: Implement traversals, BST operations, heap sort.

Applications: Trees are core for databases, filesystems, and hierarchical data.

---

Week 7: Graphs

- Representations: Adjacency list vs. matrix, space/time tradeoffs.
- Traversals: BFS (shortest path in unweighted graphs), DFS (connectivity, cycles).
- Shortest path: Dijkstra (weighted), Floyd-Warshall (all pairs), Bellman-Ford.
- Union-Find: Disjoint set, path compression, cycle detection.
- Hands-on: Code BFS/DFS, find shortest paths, detect cycles.

Industry Use: Graphs model networks, maps, dependencies, and social connections.

---

Week 8: Searching & Sorting

- Searching: Linear, binary, interpolation (when to use each).
- Sorting: Quicksort (partition, pivot selection), mergesort (divide-and-conquer), heapsort, counting sort (non-comparison).
- Hands-on: Implement and benchmark sorts, solve search-based problems.

Optimization: Knowing sort/search complexities is critical for scalable systems.

---

Week 9: Hashing & Advanced DS

- Hash tables: Insert, delete, search, load factor, rehashing.
- Collision handling: Chaining, open addressing, double hashing.
- Tries: Prefix search, autocomplete.
- Disjoint sets: Union, find, applications in graph algorithms.
- Hands-on: Build a hash map, spell checker with trie, Kruskal's MST with union-find.

Real-World: Hashing is everywhere—databases, caches, compilers.

---

Week 10: Dynamic Programming

- Memoization: Top-down, caching.
- Tabulation: Bottom-up, space optimization.
- Classic problems: 0/1 Knapsack, Longest Increasing Subsequence (LIS), Coin Change.
- Hands-on: Solve DP problems, compare recursive vs. iterative approaches.

Interview Staple: DP separates strong candidates in coding rounds.

---

Week 11: Problem Solving & Mock Interviews

- Practice: Mixed problems from LeetCode, HackerRank, Codeforces.
- Mock interviews: Simulate technical screens, whiteboarding, code reviews.
- Feedback: Time complexity analysis, code readability, edge cases.
- Hands-on: Solve under time pressure, present solutions, get peer feedback.

Job Prep: Mock interviews build confidence and identify gaps.

---

Week 12: Capstone & Final Assessment

- Project: Choose a capstone idea (e.g., library system, route optimizer, autocomplete).

- Implementation: Use multiple DSA concepts, optimize for time/space.
- Presentation: Demo, explain design choices, discuss tradeoffs.
- Assessment: Code quality, documentation, performance, presentation.
- Portfolio: GitHub repo with code, README, demo.

Next Steps: Guidance on advanced topics (A*, segment trees, persistent DS), interview prep, open-source contributions.

---

## Capstone Project Ideas

- Library Management System: Implement catalog search (hash/trie), reservations (queue), recommendations (graph similarity).
- Route Optimization Tool: Use Dijkstra/A* for shortest path, visualize with adjacency list/matrix.
- Autocomplete Search: Build a trie for prefix search, integrate with priority queue for ranking.

---

## Tools & Practice Platforms

- Languages: Python (recommended), C++, Java
- IDEs: VS Code, PyCharm, Jupyter Notebook
- Practice: LeetCode, HackerRank, Codeforces, AtCoder
- Version Control: GitHub (for projects and collaboration)
- Interview Prep: Pramp, Interviewing.io, CodePair

---

## Industry Trends & Market Relevance (2025)

- Algorithms: Still the #1 filter in tech hiring (FAANG, startups, product companies).
- System Design: DSA knowledge is foundational for designing scalable systems.
- Competitive Coding: Active in India (CodeChef, LeetCode contests), valued by recruiters.
- Open Source: Contributions to projects using advanced DSA are a plus.

- Portfolio: Completed capstone projects are a differentiator for internships and jobs.