A structured 3-month roadmap for learning Data Structures and Algorithms (DSA) in 2026

Month 1: Foundations and Core Concepts

The first month focuses on establishing a strong base in programming fundamentals, basic data structures, and algorithmic complexity.

- **Week 1: Programming Basics & Arrays**
  - Topics: Review a programming language (Python, Java, C++, etc.), understand syntax, control structures, and functions. Learn the basics of arrays and strings.
  - Tasks: Practice array traversal, string manipulation, and solving basic problems involving these concepts.
- **Week 2: Algorithmic Complexity and Searching/Sorting**
  - Topics: Learn time and space complexity using Big-O notation. Study fundamental searching algorithms (Linear and Binary Search) and basic sorting algorithms (Bubble Sort, Selection Sort, Insertion Sort).
  - Tasks: Analyze the complexity of simple code snippets and implement the covered algorithms.
- **Week 3: Linked Lists, Stacks, and Queues**
  - Topics: Understand linear data structures: Singly and Doubly Linked Lists, Stacks, and Queues. Learn their operations (insertion, deletion, peek) and implementations.
  - Tasks: Solve classic problems like "Valid Parentheses" using a stack, or implementing a queue.
- **Week 4: Hashing and Hash Tables**
  - Topics: Master hash tables and hash maps, including collisions, hashing functions, and their use in efficient lookups.
  - Tasks: Practice problems that utilize hash maps to reduce time complexity.

Month 2: Intermediate Topics

The second month moves into non-linear data structures and more complex problem-solving techniques.

- **Week 5: Trees and Binary Search Trees (BST)**
  - Topics: Learn tree data structures, including binary trees and Binary Search Trees (BSTs). Explore different traversals (In-order, Pre-order, Post-order, BFS, DFS).
  - Tasks: Implement tree traversals and solve problems like finding the height of a tree or checking if a tree is a valid BST.
- **Week 6: Advanced Trees and Heaps**
  - Topics: Study balanced trees (AVL or Red-Black basics) and heaps (priority queues). Understand their real-world applications.
  - Tasks: Practice using priority queues for problems involving finding the Kth smallest/largest element.
- **Week 7: Graph Basics**
  - Topics: Introduction to graph data structures (directed/undirected graphs, weighted/unweighted graphs). Learn representations like adjacency lists and matrices.
  - Tasks: Implement BFS and DFS for graph traversal.
- **Week 8: Graph Algorithms**
  - Topics: Explore essential graph algorithms, including Dijkstra's algorithm for shortest paths and algorithms for finding Minimum Spanning Trees (Prim's or Kruskal's).
  - Tasks: Solve simple shortest path problems on platforms like LeetCode or GeeksforGeeks.

Month 3: Advanced Topics and Interview Preparation

The final month covers advanced topics and shifts focus towards practical application and interview readiness.

- **Week 9: Dynamic Programming (DP) and Greedy Algorithms**
  - Topics: Introduction to dynamic programming (memoization and tabulation) and greedy algorithms. Understand when to apply which technique.
  - Tasks: Tackle classical DP problems such as the Fibonacci sequence, Coin Change, or Knapsack problem.
- **Week 10: Backtracking and Tries**
  - Topics: Learn backtracking techniques for combinatorial problems and understand Tries (prefix trees) for string-related problems.

- Tasks: Practice N-Queens, Sudoku solver (backtracking), and problems involving efficient prefix searching using Tries.

- **Week 11: Advanced Problem-Solving Techniques**
  - Topics: Cover advanced techniques like two-pointers, sliding window, and basic system design concepts.
  - Tasks: Apply these patterns to problems on practice platforms.

- **Week 12: Revision and Mock Interviews**
  - Topics: Revise all major DSA concepts, weak areas, and common interview questions.
  - Tasks: Dedicate time to solve a mix of problems and schedule mock interviews to simulate real interview scenarios.

Key to Success

- **Consistency:** Dedicate consistent daily hours to study and practice.
- **Practice Platforms:** Utilize platforms such as LeetCode, GeeksforGeeks or CodeChef for hands-on problem solving.
- **Analyze Mistakes:** Focus on understanding the optimal solutions and analyze your mistakes to improve problem-solving skills.
