**Phase 1: Basics of Programming**

1. **Learn a Programming Language**
   * Choose one: **C++**, **Java**, **Python** (C++ is most common for DSA)
   * Focus on syntax, loops, conditional statements, and functions.
2. **Understand Time Complexity & Space Complexity**
   * Big-O Notation: O(1), O(log n), O(n), O(n^2), O(2^n)
   * Best, worst, and average-case analysis.
   * Analyze time complexity of your own code.

**Phase 2: Basic Data Structures**

1. **Arrays**
   * Insertion, deletion, searching.
   * Key algorithms: Kadane's Algorithm, Binary Search, Two-pointer technique.
2. **Strings**
   * Operations like reversing, palindrome checking, string comparison.
   * Important algorithms: KMP Algorithm, Rabin-Karp Algorithm, Longest Palindromic Substring, and more.
3. **Linked Lists**
   * Singly, Doubly, and Circular linked lists.
   * Operations: Insertion, Deletion, Traversal, Reversal.
   * Key problems: Detecting loops, reversing linked lists.
4. **Stacks and Queues**
   * Stack: LIFO (Last In First Out), operations using arrays and linked lists.
   * Queue: FIFO (First In First Out), Circular queue, Priority Queue.
   * Important algorithms: Stack-based expressions evaluation, Stock span problem.

**Phase 3: Recursion and Backtracking**

1. **Understand Recursion**
   * Base case, recursive calls.
   * Practice problems like factorial, Fibonacci, and permutations.
2. **Backtracking**
   * Explore problems like N-Queens, Sudoku Solver, Rat in a Maze, Subsets.

**Phase 4: Advanced Data Structures**

1. **Trees**
   * Binary Trees, Binary Search Trees (BST), AVL Trees, Segment Trees.
   * Key concepts: Preorder, Inorder, Postorder traversals.
   * Problems: Lowest Common Ancestor (LCA), Diameter of a Tree, Balanced BST.
2. **Heaps**
   * Min Heap, Max Heap, Heapify, Priority Queue.
   * Heap sort algorithm.
   * Key problems: Kth largest element, Merge k sorted arrays.
3. **Graphs**
   * Graph representation: Adjacency List/Matrix.
   * Traversal Algorithms: Depth First Search (DFS), Breadth First Search (BFS).
   * Shortest Path Algorithms: Dijkstra, Bellman-Ford.
   * Minimum Spanning Tree (MST): Prim's, Kruskal’s algorithm.
   * Important problems: Detect cycles, Topological Sorting.
4. **Hashing**
   * Hash tables, Hash maps, Collision handling techniques.
   * Problems: Counting frequency of elements, Longest subarray with sum.

**Phase 5: Dynamic Programming**

1. **Understand the Idea of Overlapping Subproblems and Optimal Substructure**
   * Tabulation vs. Memoization (Bottom-up vs. Top-down approaches).
2. **Famous DP Problems**
   * 0/1 Knapsack Problem
   * Longest Increasing Subsequence
   * Longest Common Subsequence
   * Matrix Chain Multiplication
   * Edit Distance
   * Coin Change Problem

**Phase 6: Greedy Algorithms**

1. **Understand the Greedy Choice Property and Optimal Substructure**
   * Problems: Activity Selection, Fractional Knapsack, Job Sequencing, Huffman Encoding.

**Phase 7: Advanced Algorithms**

1. **Sorting Algorithms**
   * Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, and Heap Sort.
   * Stability of Sorting algorithms, Time complexity comparisons.
2. **Searching Algorithms**
   * Linear Search, Binary Search (Iterative and Recursive).
   * Binary Search on Answer (problems like allocating pages to books, aggressive cows problem).

**Phase 8: Divide and Conquer**

1. **Important Concepts**
   * Understanding the divide-and-conquer approach.
   * Examples: Merge Sort, Quick Sort, Binary Search, Closest Pair of Points.

**Phase 9: Advanced Topics**

1. **Trie**
   * Insert, Search, Delete in a Trie.
   * Applications: Auto-complete, Spell-checker.
2. **Segment Tree**
   * Build, Update, Query operations.
   * Range queries (sum, minimum, maximum).
3. **Fenwick Tree (Binary Indexed Tree)**
   * Sum range queries, point updates.
4. **Disjoint Set Union (Union-Find)**
   * Union by rank, Path compression.
   * Applications: Detecting cycle in a graph, Kruskal’s algorithm.

**Phase 10: Competitive Programming Practice**

1. **Practice Platforms**
   * **LeetCode**: Practice DSA problems and contest-style problems.
   * **HackerRank**: Detailed tutorials for DSA and algorithms.
   * **Codeforces**: Competitive programming problems.
   * **GeeksforGeeks**: Extensive problem bank with solutions.
2. **Practice Problems by Topic**
   * Solve problems from easy to hard for each data structure and algorithm.
   * Focus on time complexity, edge cases, and optimal solutions.

**Phase 11: Mock Interviews & Problem-Solving Strategy**

1. **Participate in Mock Interviews**
   * Simulate real coding interview conditions.
2. **Optimize Your Problem-Solving Strategy**
   * Use the **READ-THINK-CODE-TEST** approach:
     + **READ** the problem carefully.
     + **THINK** of various edge cases and an optimal solution.
     + **CODE** the solution with clarity.
     + **TEST** on various cases.

**Phase 12: Master System Design (Optional but Helpful)**

1. **Learn System Design Concepts**
   * Microservices, Load balancing, Caching, Database partitioning, Scalability.
2. **Design Key Systems**
   * Design a Rate Limiter, Message Queue, Distributed File System, etc.

**Tools and Resources**

1. **Books**
   * **Introduction to Algorithms** by CLRS.
   * **Data Structures and Algorithm Analysis** by Mark Allen Weiss.
2. **YouTube Channels**
   * **Abdul Bari**: For basic DSA explanations.
   * **Take U Forward**: Detailed DSA tutorials and explanations.
3. **Courses**
   * **Coursera**: Algorithm Specialization by Stanford University.
   * **Udemy**: Mastering DSA & algorithms from scratch.

**Conclusion**

Mastering DSA requires time and patience. Start with basic concepts and gradually move to advanced topics. Regular practice, consistency, and mock interviews will build your problem-solving skills and prepare you for competitive programming and coding interviews.

Would you like suggestions for specific problem sets based on this roadmap?

If you dedicate 5 hours daily to learning Data Structures and Algorithms (DSA), the time it will take to achieve mastery depends on your background, problem-solving skills, and learning pace. However, here's a rough estimate:

Phase-wise Time Estimation:

Phase 1: Basics of Programming

(1-2 weeks)

If you're already familiar with a programming language, you can quickly go through this phase in about 1 week.

Phase 2: Basic Data Structures

(3-4 weeks)

Learning arrays, strings, linked lists, stacks, and queues with hands-on practice.

Phase 3: Recursion and Backtracking

(2-3 weeks)

Mastering recursion and backtracking will require both understanding and practicing related problems.

Phase 4: Advanced Data Structures

(4-5 weeks)

Trees, heaps, graphs, and hashing will take time due to their complexity and the number of algorithms you need to practice.

Phase 5: Dynamic Programming

(3-4 weeks)

DP can be challenging. Dedicate time to learning, understanding patterns, and solving common problems.

Phase 6: Greedy Algorithms

(2 weeks)

Greedy algorithms are relatively easier but require some thought and problem-solving practice.

Phase 7: Advanced Algorithms

(2-3 weeks)

Sorting, searching, and divide-and-conquer techniques.

Phase 8: Competitive Programming Practice

(3-4 months)

This phase involves consistent problem-solving on platforms like LeetCode, Codeforces, etc. Depending on your goals (competitive programming or interviews), the time may vary.

Total Time Estimation

Without prior knowledge: 6-8 months.

With some prior experience in programming: 4-6 months.

This estimate assumes steady learning, with deep dives into concepts and sufficient practice. If you're solving more complex problems or participating in competitions, it may take longer.

Let me know if you'd like any adjustments to your plan!