Phase 1: C++ Foundation & Basic Problem Solving (Months 1-2)

Your goal here is to become fluent in C++. You must be comfortable with its syntax and the Standard Template Library (STL), as it's the foundation for everything that follows.

- **Week 1-2: C++ Basics**
    - **Topics:** Basic syntax, data types, operators, loops, functions, pointers, and references.
    - **Action:** Write simple programs for every concept. Use HackerRank's C++ track for guided practice.
    - **Resource:** "Starting Out with C++" by Tony Gaddis or any beginner-friendly C++ course on Udacity/Coursera.
- **Week 3-4: Object-Oriented Programming (OOP)**
    - **Topics:** Classes, objects, inheritance, polymorphism, encapsulation, and abstraction.
    - **Action:** Implement simple classes like Student or BankAccount to solidify your understanding.
    - **Resource:** GeeksforGeeks articles on C++ OOP.
- **Week 5-6: C++ Standard Template Library (STL)**
    - **Topics:** Master vector, string, pair, map, set, stack, queue, and priority_queue.
    - **Action:** Solve at least 20-30 easy problems on LeetCode using only STL containers. This is **critical**.
    - **Resource:** TopCoder tutorial on C++ STL.
- **Week 7-8: Time/Space Complexity & Basic Math**
    - **Topics:** Big O, Big Ω, Big Θ notation. Learn to analyze the complexity of your code. Cover basic number theory concepts like prime numbers, GCD, and modular arithmetic.
    - **Action:** For every problem you solve, manually calculate the time and space complexity.
    - **Resource:** HackerRank's Big-O notation guide.

Phase 2: Core Data Structures (Months 3-4)

Now, you'll build and understand the fundamental tools of a programmer. For each data structure, follow this pattern: **Learn theory -> Implement it from scratch -> Solve problems**.

- **Week 9-10: Arrays & Strings**
    - **Topics:** Array manipulation, sliding window, prefix sums, two-pointer technique. String searching (KMP), palindromes, and subsequences.
    - **Action:** Solve 10-15 array and string problems on LeetCode (mix of easy/medium).
    - **Resource:** LeetCode's "Top Interview Questions - Easy" collections for Arrays and Strings.
- **Week 11-12: Linked Lists**
    - **Topics:** Singly, doubly, and circular linked lists. Operations like insertion, deletion, reversal, and cycle detection (Floyd's algorithm).
    - **Action:** Implement a linked list from scratch. Solve classic problems like "Reverse a Linked List" and "Merge Two Sorted Lists."
    - **Resource:** "Data Structures and Algorithms in C++" by Goodrich.
- **Week 13-14: Stacks & Queues**
    - **Topics:** LIFO and FIFO principles. Implementations using arrays and linked lists. Applications like parenthesis matching, expression evaluation, and BFS.
    - **Action:** Solve problems that use stacks for tracking state (e.g., "Valid Parentheses") and queues for level-order traversal.
    - **Resource:** GeeksforGeeks articles on Stacks and Queues.
- **Week 15-16: Recursion & Backtracking**
    - **Topics:** Principle of recursion, base cases, recursive tree. Introduction to backtracking for solving problems like N-Queens and Sudoku.
    - **Action:** Start with simple recursive problems (e.g., factorial, Fibonacci) and gradually move to backtracking puzzles.
    - **Resource:** LeetCode's Recursion I explore card.

Phase 3: Advanced Data Structures & Algorithms (Months 5-8)

This phase is intense. You'll tackle complex structures and algorithms that are frequently asked about in top-tier company interviews.

- **Week 17-20: Trees & Heaps**
    - **Topics:** Binary Trees, Binary Search Trees (BST), tree traversals (in-order, pre-order, post-order, level-order), balancing (AVL/Red-Black basics). Heaps (Min-Heap, Max-Heap) and their use in Priority Queues.
    - **Action:** Implement a BST with insert, search, and delete functions. Solve at least 20 tree problems. Use std::priority_queue for heap-based problems.
    - **Resource:** "Introduction to Algorithms" (CLRS) for theory. LeetCode problems are essential here.
- **Week 21-24: Hash Tables & Graphs**
    - **Topics:** Hashing functions, collision resolution. Graph representations (Adjacency Matrix/List), traversals (BFS, DFS), cycle detection, and topological sort.
    - **Action:** Solve problems involving frequency counting with std::unordered_map. Implement BFS and DFS on a graph.
    - **Resource:** GeeksforGeeks Graph Data Structure and Algorithms section.
- **Week 25-28: Greedy Algorithms & Dynamic Programming (DP)**
    - **Topics:** Understanding the greedy choice property. Introduction to DP: memoization vs. tabulation, identifying DP problems (e.g., optimal substructure).
    - **Action:** Solve classic greedy problems (Activity Selection). For DP, start with 1D problems (Fibonacci, Climbing Stairs) before moving to 2D (Knapsack, Longest Common Subsequence).
    - **Resource:** "Cracking the Coding Interview" for a great introduction to DP.
- **Week 29-32: Advanced Algorithms**
    - **Topics:** Sorting (Merge, Quick, Heap), Searching (Binary Search on answers), Dijkstra's, Floyd-Warshall.
    - **Action:** Implement Merge Sort and Quick Sort. Solve problems that use binary search on a non-obvious range.
    - **Resource:** CLRS book for in-depth algorithm analysis.

Phase 4: Interview Readiness & Specialization (Months 9-10)

The final stretch is about consolidating your knowledge, practicing under pressure, and preparing for the interview environment.

- **Week 33-36: Advanced Problem Solving & Contests**
    - **Topics:** Tries, Segment Trees (basics), Bit Manipulation.
    - **Action:** Solve problems from different topics on LeetCode (medium/hard). Start participating in weekly contests on LeetCode, CodeChef (Long Challenge), and Codeforces (Div 3/4).
    - **Goal:** Focus on speed and accuracy. Aim to solve 2-3 problems in a contest.
- **Week 37-40: System Design & Mock Interviews**
    - **Topics:** Basics of system design: scalability, load balancing, caching, database choices.
    - **Action:** Read the "Grokking the System Design Interview" course summary. Practice explaining your thought process out loud. Do mock interviews with peers or on platforms like Pramp.
    - **Resource:** "Cracking the Coding Interview" for behavioral questions and final preparation tips.

Performance Tracking & Final Advice

- **GitHub Portfolio:** Create a repository for your DSA journey. Push the code for every data structure you implement from scratch and solutions to challenging problems.
- **Problem Tracker:** Use a simple spreadsheet (or LeetCode's built-in lists) to track problems. Note the problem name, difficulty, data structure/algorithm used, and time taken. Revisit problems you struggled with after a few weeks.
- **Consistency is Key:** A little bit every day is better than a lot once a week. Your schedule is tight, so every session counts.
- **Understand, Don't Memorize:** Never copy-paste a solution. If you're stuck, look at the logic, then close the tab and try to code it yourself. If you can't, you haven't understood it yet.
- **Quality over Quantity:** Solving 300 problems with deep understanding is far better than rushing through 500.