# 90 day roadmap

## Phase 1: Days 1–15 (Foundations & Basic Math)

### Day 1–5: Mathematical & Analytical Foundations

1. **Order of Growth (Time Complexity & Big-O Notation)**
    - Big O, Big Theta, Big Omega
    - Common complexities: O(1), O(log n), O(n), O(n log n), O($n^2$), etc.
2. **Basic Math Refresher**
    - Factorials (n!)
    - GCD/HCF, LCM
    - Prime numbers, Fibonacci series (basic generation)
3. **Implementation Practice**
    - Write small programs to calculate factorial, HCF/LCM, Fibonacci series.
    - Analyze each program's time complexity.

**Practice Suggestions:**

- Implement factorial using both iterative & recursive approaches.
- Try different ways to find GCD (Euclid's algorithm vs. brute force) and note performance differences.

---

### Day 6–10: Intro to Programming Concepts & Language Basics

1. **Data Types, Variables, and Operators**
    - Primitive data types (int, float, bool, char, etc.)

2. **Control Structures**
    - Conditional statements (if-else, switch)
    - Loops (for, while, do-while)
3. **Functions & Modularity**
    - Function declaration and definition
    - Parameter passing (by value, by reference – depending on language)

**Practice Suggestions:**

- Simple programs like checking palindromes, reversing numbers, sum of digits, etc.
- Focus on writing clean, modular code with functions.

---

## Day 11–15: Algorithmic Basics & Getting Comfortable

1. **Introduction to Basic Data Structures**
    - Arrays (1D & 2D) conceptually (just a quick overview, deep dive will come later)
    - Strings (basic manipulation)
2. **Mathematical Problems in More Detail**
    - Practice more problems on prime numbers, factorial, permutations/combinations logic.
    - Get comfortable with implementation details.
3. **Complexity Practice**
    - For each program, estimate complexity.
    - Understand how array-based solutions differ in complexity from naive solutions.

● Start exploring a coding platform (e.g., **LeetCode**, **CodeChef**, **HackerRank**, or **GeeksforGeeks**) and solve easy-level problems.

● Maintain a notebook/log of solutions and complexities.

---

# Phase 2: Days 16–30 (Recursion & Arrays in Depth)

## Day 16–20: Recursion Fundamentals

1. **Concept of Recursion**
    - ○ Base case, recursive case, stack memory usage
    - ○ Tail recursion vs. non-tail recursion
2. **Classic Recursive Problems**
    - ○ **Tower of Hanoi**
    - ○ **Josephus Problem**
    - ○ Generating subsets/permutations recursively

**Practice Suggestions:**

● Understand how recursion unfolds via recursion tree diagrams.

● Dry-run your code for small inputs and see how the call stack changes.

---

## Day 21–25: Arrays (Basics to Advanced)

1. **Array Operations**
    - ○ Insertion, Deletion, Traversal, Searching, Sorting
    - ○ Common Sorting Algorithms: Bubble, Insertion, Selection (understand complexities, implement at least one or two)

- Kadane’s Algorithm (Maximum Subarray Sum)
- Finding subarray with given sum

3. **Intermediate Problems**
   - **Stock Buy and Sell** (maximize profit)
   - **Rainwater Trapping** (classic two-pointer approach)

**Practice Suggestions:**

- Solve array-based questions from a platform like GeeksforGeeks or LeetCode.
- Focus on two-pointer approach & prefix-sums for array problems.

---

## Day 26–30: Arrays Deep Dive + Two-Dimensional Arrays

1. **2D Array Problems**
   - Matrix traversal (spiral order, diagonal traversal)
   - Searching a sorted 2D matrix
2. **Rotation & Rearrangement**
   - Rotate matrix by 90 degrees
   - Reverse rows/columns
3. **Edge Cases & Complexity**
   - Learn how large 2D array solutions might become expensive in time and memory.

**Practice Suggestions:**

- Practice about 5–10 matrix-related problems that cover different traversal styles.

# Phase 3: Days 31–45 (Stacks, Queues & Intro to Trees)

## Day 31–35: Stacks & Queues

1. **Stack Basics**
    - Implementation (array-based or linked list-based)
    - Common operations (push, pop, peek, isEmpty)
2. **Stack Problems**
    - **Balanced Parentheses**
    - **Stock Span**
    - Next Greater Element (NGE)
3. **Queue Basics**
    - Implementation (circular array, linked list)
    - Operations (enqueue, dequeue, front, rear)
    - **Queue Applications** (e.g., generating binary numbers)

**Practice Suggestions:**

- Understand the difference between stack & queue usage in real-world scenarios.
- Solve 5–7 problems each on stack and queue from an online judge.

---

## Day 36–40: Tree Foundations

1. **Introduction to Trees**

- Types: Binary Tree, Binary Search Tree (BST), AVL, Red-Black (just an overview)

2. **Basic Tree Traversals**
   - **Pre-order**, **In-order**, **Post-order** (recursive & iterative)
   - Level-order traversal (using queue)
3. **Implementation**
   - Building a binary tree from array/linked representation
   - Insertion in BST

**Practice Suggestions:**

- Write functions for all traversal methods.
- Practice building a BST from a list of numbers, then implement search and insertion.

---

## Day 41–45: Tree Problems

1. **Common Binary Tree Problems**
   - Height/Depth of a tree
   - Count leaves, internal nodes
   - Mirror/Invert a binary tree
   - Check if two trees are identical
2. **Binary Search Tree Specific**
   - Validate BST
   - Deletion in BST (understand the 3 cases)

○ Zigzag (spiral) level-order traversal

○ Tree diameter

**Practice Suggestions:**

● Solve problems on **LeetCode** (e.g., “Binary Tree Level Order Traversal”, “Symmetric Tree”, etc.).

● Solidify your understanding of tree recursion.

---

# Phase 4: Days 46–60 (Graphs & Dynamic Programming)

## Day 46–50: Graphs

1. **Graph Theory Basics**

   ○ Terminology: vertices, edges, weighted/unweighted, directed/undirected

   ○ Adjacency Matrix vs. Adjacency List

   ○ Representation in code

2. **Graph Traversal Algorithms**

   ○ **BFS (Breadth-First Search)**

   ○ **DFS (Depth-First Search)**

   ○ Applications (shortest path in an unweighted graph using BFS, connected components using DFS)

3. **Implementation Practice**

   ○ Implement BFS & DFS both iteratively and recursively (for DFS).

   ○ Practice on small graph examples and visualize them.

**Practice Suggestions:**

- Try BFS & DFS on various data sets (undirected, directed graphs).

---

## Day 51–55: Graph Algorithms & Introduction to DP

1. **Graph Algorithms**
    - **Dijkstra's Algorithm** (shortest path in weighted graph with non-negative edges)
    - Basic idea of Bellman-Ford or Floyd-Warshall (if time permits)
2. **Dynamic Programming Foundations**
    - DP definition: overlapping subproblems & optimal substructure
    - Memoization vs. Tabulation approaches
    - **Fibonacci** with DP as a starter example

**Practice Suggestions:**

- Write code for Dijkstra's from scratch (using priority queue/min-heap).
- Practice simple DP problems (Fibonacci, climb stairs, ways to reach a cell in a grid).

---

## Day 56–60: Core Dynamic Programming Problems

1. **Key DP Problems**
    - **Longest Common Subsequence (LCS)**
    - **Coin Change (min coins / ways to make change)**
    - **Egg Dropping** (classic DP for optimization)
2. **Problem-Solving Approach**

- Draw DP tables or recursion trees to visualize solutions.

3. **Optimization & Complexity**
   - Time complexity analysis of DP solutions.
   - Memory optimization (space optimization from $O(n^2)$ to $O(n)$ where possible).

**Practice Suggestions:**

- Focus on a diverse set of DP problems (knapsack variations, subsets, partition problems).
- Revise BFS/DFS and see if any DP approach can solve those graph-based problems better.

---

# Phase 5: Days 61–75 (Advanced Data Structures & More)

## Day 61–65: Tries (Prefix Trees) & Segment Trees

1. **Tries**
   - **Concept**: storing strings by character paths
   - Operations: insert, search, prefix search, auto-complete
   - Use cases: dictionary, spell-check
2. **Segment Trees (Range Queries)**
   - Building a segment tree for sum or min/max in a range
   - Update queries
3. **Fenwick Tree (Binary Indexed Tree)** (Optional if time allows)
   - Alternative to segment tree for simpler updates and prefix sums

**Practice Suggestions:**

- For segment trees, do range sum queries, range min/max queries, and updates.

---

## Day 66–70: Disjoint Set (Union-Find) & Further Advanced DS

1. **Disjoint Set / Union-Find**
    - **Make-Set, Union, Find**
    - Path compression, union by rank/size
    - Applications: detecting cycles in an undirected graph, Kruskal's MST
2. **Heap (Priority Queue)**
    - Min-heap vs. max-heap
    - Common operations (insert, extract-min/max)
    - Applications: scheduling, Dijkstra's algorithm
3. **Advanced Tree Structures (Overview)**
    - AVL/Red-Black (just conceptual if short on time)
    - B-Trees/B+ Trees (used in databases, only conceptual if short on time)

**Practice Suggestions:**

- Use Union-Find for solving "connected components" or "cycle detection" in graphs.
- Practice priority queue-based problems (merge k-sorted lists, etc.).

---

## Day 71–75: Complex Problem-Solving Sessions

1. **Integrate Concepts**
    - Combine DS + DP + Graph knowledge for complex problem statements.

2. **Mock Interviews & Timed Practice**

    ○ Attempt timed mock tests (1–2 hours) simulating interview environment.

    ○ Solve multiple medium/hard-level problems under time constraints.

3. **Weakness Review**

    ○ Identify areas where you struggle (maybe graph-based DP, or advanced data structures).

    ○ Revise & re-practice these specifically.

**Practice Suggestions:**

● Platforms like **LeetCode** (medium/hard), **Codeforces**, and **InterviewBit** will give you a good range of tougher problems.

● Focus on explaining your solution approach clearly (as if in an interview).

---

# Phase 6: Days 76–90 (Revision, Projects & Final Touches)

## Day 76–80: Revision & Practice

1. **Structured Revision**

    ○ Revisit each major topic (Arrays, Stacks, Queues, Trees, Graphs, DP, Advanced DS) in a structured manner.

    ○ Make summary notes or mind maps to recall key operations & complexities.

2. **Daily Problem-Solving**

    ○ Practice at least **2–3** problems daily from random topics to keep everything fresh.

    ○ Focus on variety: BFS, DP, recursion, stack-based, etc.

3. **Interview Prep**

    ○ Common interview questions (like Two-Sum, Reverse Linked List, etc.) for warm-up.

# 90 day roadmap

## Day 81–85: Mini-Projects (Hands-On Implementation)

1. **Sudoku Solver**
    - Backtracking approach (DFS + recursion).
    - Great for testing recursion + backtracking knowledge.
2. **Shortest Path Finder**
    - Use BFS for unweighted, or Dijkstra's for weighted.
    - Possibly build a small GUI or console-based program to visualize the graph & path.
3. **N-Queen Visualizer**
    - Classic backtracking problem.
    - Building a visual representation to see queen placements.

**Why Projects?**

- Projects **solidify** concepts.
- They **boost** your résumé (showing hands-on DSA experience).

---

## Day 86–90: Final Review & Preparation

1. **Project Completion & Polishing**
    - Finish any pending tasks in your mini-projects.
    - Add them to a GitHub repository or portfolio.
2. **Final Assessments**
    - Re-attempt previously solved problems to see if you can solve faster.

3. **Resume & Profile Upgrade**

    ○ Update your résumé with your newly acquired DSA projects.

    ○ If you're taking a course like **GFG's 3–90 Challenge**, finalize any course requirements for the refund or certificate.

**Tip:**

● Make sure you can explain each project's approach, complexities, and possible improvements.

---

# Additional Tips & Resources

1. **Time Management**

    ○ Spend at least 2–3 hours daily on weekdays, and 4–5 hours on weekends if possible.

    ○ Consistency is key. If you skip days, ensure you catch up on critical topics.

2. **Coding Platforms**

    ○ **GeeksforGeeks**: Great for theoretical + basic to advanced DSA practice.

    ○ **LeetCode**: Focused on interview-style problems.

    ○ **HackerRank** / **CodeChef**: Good for competitive programming & timed practice.

3. **Focus on Problem-Solving Approach**

    ○ Before coding, always think about constraints, possible edge cases, and optimal approach.

    ○ Write pseudocode or outline your approach to avoid confusion.

4. **Peer Learning / Community**

    ○ Join study groups or forums to discuss solutions, get hints, and stay motivated.

# 90 day roadmap

5. **Mentor or Guide**
    - If you can, find a mentor who can do code reviews for you.
    - Regular feedback accelerates learning.