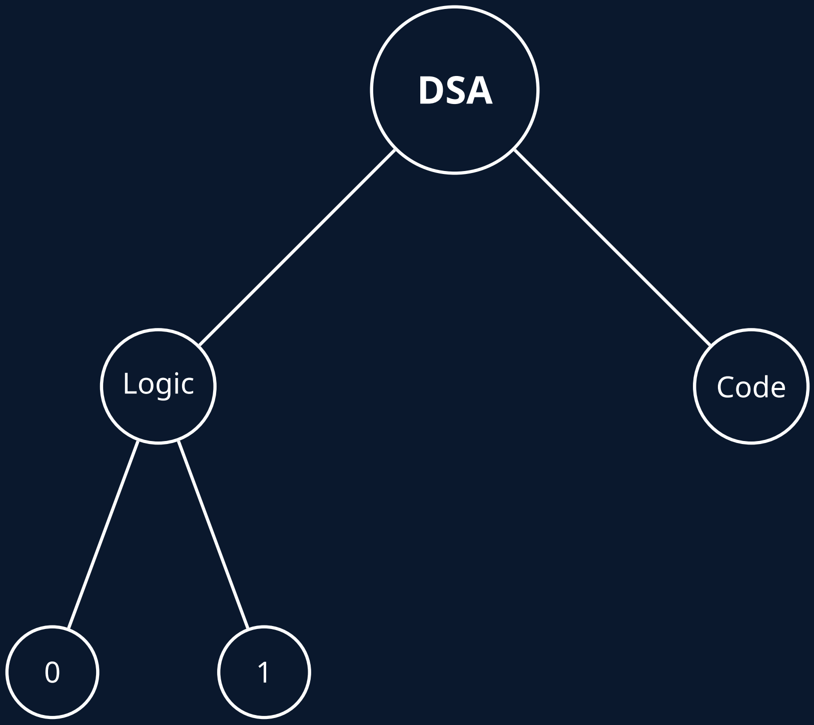
THE DSA BLUEPRINT

Part 1: The Strategy & Roadmap

(Comprehensive Learning Notes Coming Soon)



"Code is just syntax.
Logic is the real language."

Author: Shaikh Mahad (codewithmahad)

Version: 1.0 (Public Launch)

Prepared for the Computer Science Community

Contents

Introduction: Why This Exists

Welcome to the start of a long but rewarding path.

I wrote this document because I noticed a pattern. Many of us enter Computer Science with high energy. We attend classes, we pass exams, and we get good grades. But when we look outside our bubble — when we look at students in India, the US, or Europe — we see a gap. We see students who **aren't just passing subjects**; they are solving hard problems on LeetCode, building complex systems, and cracking interviews at top tech giants before they even graduate.

The world of software engineering does not care where you come from or which university name is on your degree. It only cares about one thing: **Can you solve the problem?**

This guide is not here to replace your university education. Your degree gives you the qualification and the broad theoretical base. This guide is here to add the **practical edge**. It is here to bridge the gap between "knowing the definition of a Linked List" and "using a Linked List to solve a real-world optimization problem."

The Global Standard

If you look at the curriculum followed by top students worldwide, they treat Data Structures and Algorithms (DSA) as a sport. They practice it daily. They don't just learn a topic to pass a quiz; they learn it to break down complex logic.

My goal with **The DSA Blueprint** is to bring that same intensity and structure to you. This is not a list of random tutorials. It is a filtered, organized path. It cuts through the noise. There is so much content on YouTube and Google that it is easy to get lost or study things in the wrong order. You might try to learn Graphs before you understand Recursion, get frustrated, and quit. This guide stops that from happening.

Who Is This For?

- **The Beginner:** Who knows `if-else` and `loops` but gets scared when they hear words like "Time Complexity."
- **The Student:** Who wants to stay ahead of their class and secure a high GPA, but also wants to be ready for technical interviews.
- **The Dreamer:** Who aims to work remotely for international startups or land roles in big tech companies.

How to Use This Guide

Treat this document like a map. Do not skip steps. Do not jump to the "cool" topics like Artificial Intelligence until you can handle the basics of data storage. I have arranged the topics in a way that builds your muscle memory.

We start with the building blocks. We move to linear data structures. Then we tackle the complex non-linear structures. Finally, we look at the patterns that solve 90% of interview questions.

Let's build logic that lasts a career, not just a semester.

— **Shaikh Mahad** (codewithmahad)

The Resource Vault: Before You Begin

Don't waste time searching for "best tutorials." Here are the industry-standard resources.

1. Setup Your Platforms

Before writing code, create accounts here. This is your gym.

- **HackerRank:** `https://www.hackerrank.com/` (Start here for basics)
- **LeetCode:** `https://leetcode.com/` (The main battleground)

2. Video Resources by Language

For C++ Students:

- **Striver (TakeUForward):** A-Z DSA Course. The absolute gold standard.
- **Love Babbar:** DSA Series.

For Java Students:

- **Apna College (Shradha Khapra):** Alpha Placement Course.
- **Kunal Kushwaha:** DSA Bootcamp.

For Python Students:

- **NeetCode:** NeetCode.io. Best for Python solutions.

Visual Tip

Why Resources Matter: Imagine trying to build a house with bad tools. You might finish, but it will be crooked. Use the best tools (resources) to build a strong foundation.

The Strategy Manual: Read Before Starting

Most students fail not because they are not smart, but because they have no system.

1. How to Measure Progress (The Reality Check)

Phase	Milestone (You are safe if...)	Feeling
Phase 0	You can write nested loops for star patterns without Google.	Confident
Phase 1	You understand why nested loops are $O(n^2)$.	Curious
Phase 2	You stop seeing ”Array questions” and start seeing ”Sliding Window questions.”	Powerful
Phase 3	You can reverse a Linked List on paper.	Annoyed
Phase 4	You trust Recursion (The Leap of Faith).	Scared
Phase 5	You can explain your logic before writing code.	Ready

2. The Mistakes Hall of Fame

I made these mistakes so you don't have to.

- **Mistake #1:** I watched solution videos immediately. **Result:** I felt smart, but couldn't solve anything in the exam.
- **Mistake #2:** I avoided Recursion for months. **Result:** I got stuck on Trees and Graphs later.
- **Mistake #3:** I solved random easy problems to feel good. **Result:** My rating never improved.
- **Mistake #4:** I did not make handwritten notes. **Result:** I forgot concepts after 2 weeks and had to re-watch hours of video tutorials to recall them. **Make notes!**

3. How to Revise (Spaced Repetition)

DSA is volatile. You will forget it.

- **The Weekend Rule:** Every Sunday, do NOT learn anything new. Only re-solve 2 problems you got wrong during the week.
- **The Mistake Notebook:** Maintain a Google Doc/Notion with specific notes like: *”Forgot to handle the null case in Linked List.”*

The Pro Mindset: How to Practice

The Daily Routine Template

To succeed, you need a system. Use this routine:

1. **Concept (30 Mins):** Watch a video or read an article on the topic.
2. **Paper Solving (15 Mins):** Do not touch the keyboard. Take a pen and paper. Dry run the logic.
3. **Coding (60 Mins):** Try to code it on LeetCode. Debug errors yourself.
4. **Notes (15 Mins):** Write down *what you learned*.

How to Solve ANY Problem

1. **Read Twice:** Understand constraints. Is $N < 100$ or $N < 10^5$?
2. **Brute Force:** First, think of the stupidest solution. $O(n^2)$ is fine to start.
3. **Optimize:** Can we use a Map? Two Pointers? Binary Search?
4. **Dry Run:** Test your logic on paper with an example input.
5. **Code:** Finally, write the syntax.

1 Phase 0: The Prerequisites

Timeline: 1–2 Weeks

This phase establishes the non-negotiable coding fluency required before tackling actual algorithms.

“Before we talk about algorithms, we have to talk about fluency. You cannot write a novel if you are still struggling with spelling. Similarly, you cannot master Data Structures if you are still struggling with syntax.”

This phase is the gatekeeper. If you find yourself constantly Googling ”how to write a for loop” or ”how to return a value from a function,” do not move to Phase 1. Stay here. Practice more. The code must flow from your hands naturally.

Mahad’s Rule

If you can’t write loops without thinking, DSA will destroy you later. Do not skip this phase.

1.1 The Core Syntax Checklist

Regardless of your language, you must be comfortable with these four pillars:

- **Loops:** Nested loops ($O(n^2)$ logic). Can you print a star pattern?
- **Functions:** Passing parameters by Value vs. Reference.
- **Memory:** Stack vs. Heap.

1.2 Object-Oriented Programming (OOP) - The 3rd Semester Essential

Since you are in Semester 3, DSA and OOP often go hand-in-hand. You must know:

- **Classes & Objects:** How to create a ‘Node‘ class.
- **Constructors:** Initializing your data structures.
- **Pointers/References:** Linking objects together (Crucial for Linked Lists/Trees).

1.3 Language Specific Alerts

C++ Students

Watch Out: Be careful with Pass-by-Value on Vectors. Always pass vectors by reference (&) in recursion, or you will get Time Limit Exceeded (TLE) because C++ copies the whole array every time.

>_ Java Students

Watch Out: Java handles memory automatically, but be careful with String concatenation in loops. Use `StringBuilder`, otherwise, your complexity becomes $O(n^2)$.

>_ Python Students

Watch Out: Python recursion limit is small (1000). For deep recursion (Graphs/Trees), you might need `sys.setrecursionlimit`. Also, Python lists are dynamic arrays, not linked lists.

Important Check

Gatekeeper Problems:

1. **Pattern Printing:** Write a program to print a half-pyramid of stars using nested loops.
2. **Basic Array Logic:** Given an array of integers, write a function that returns the sum of all even numbers.
3. **Modular Arithmetic:** Write a function that checks if a number is Prime.

Key Takeaway

If you can write the ”Gatekeeper Problems” on paper without syntax errors, your foundation is solid enough to start engineering real solutions.

2 Phase 1: The Foundation & Complexity

Timeline: 2 Weeks

This phase bridges the gap between basic coding and engineering solutions by introducing efficiency metrics.

"We are done with "learning to code." Now we start "learning to engineer." In the 3rd Semester, the difference between a good student and a great engineer is not whether they can solve the problem—it is whether they can solve it ***efficiently****."*

Mahad's Rule

Always ask yourself: "Can this be faster?" If you wrote $O(n^2)$, ask "Can I do it in $O(n)$?"

2.1 Time & Space Complexity (The Yardstick)

Concept in a Nutshell

Big O is just "Worst Case Scenario." It measures how the number of operations grows as the input size (N) grows. We don't care about exact seconds; we care about the rate of growth.

How to Read Constraints

- $N \leq 10$: $O(N!)$ or $O(2^N)$ (Recursion/Backtracking)
- $N \leq 100$: $O(N^3)$ or $O(N^4)$ (Floyd Warshall / DP)
- $N \leq 10^4$: $O(N^2)$ (Nested Loops / Bubble Sort)
- $N \leq 10^6$: $O(N \log N)$ or $O(N)$ (Sorting / Hash Map / Two Pointers)
- $N \leq 10^9$: $O(\log N)$ or $O(1)$ (Binary Search / Math Formula)

Recommended Read (Deep Dive)

Read Article: Time Complexity Analysis (GeeksforGeeks)
Visualize how your code's performance changes as data grows.

2.2 Arrays & Strings

Concept in a Nutshell

Arrays are contiguous memory blocks. Access is fast ($O(1)$), but insertion/deletion is slow ($O(n)$) because elements must shift.

Easy / Warmup

- Reverse an Array (In-place). [LeetCode 344]
- Check if Array is Sorted and Rotated. [LeetCode 1752]

- Remove Duplicates from Sorted Array. [LeetCode 26]
- Running Sum of 1d Array. [LeetCode 1480]

Medium / Standard

- Move Zeroes. [LeetCode 283]
- Rotate Array by K steps. [LeetCode 189]
- Maximum Subarray (Kadane's Algorithm). [LeetCode 53]
- Sort Colors (Dutch National Flag). [LeetCode 75]
- Rearrange Array Elements by Sign. [LeetCode 2149]

Hard / Master

- Product of Array Except Self. [LeetCode 238]
- Missing Number (XOR Method). [LeetCode 268]
- First Missing Positive. [LeetCode 41]

2.3 Basic Number Theory

Concept in a Nutshell

Math logic often replaces loops. Instead of checking every number, use properties like Modulo or Sieve.

- **Easy:** Palindrome Number. [LeetCode 9]
- **Medium:** Count Primes (Sieve of Eratosthenes). [LeetCode 204]
- **Medium:** GCD of two numbers.

2.4 Sorting Algorithms (The Order of Chaos)

Concept in a Nutshell

Sorting is fundamental. While you will often use built-in functions like `.sort()`, understanding $O(N \log N)$ (Merge/Quick Sort) vs $O(N^2)$ (Bubble/Insertion) is critical for interviews.

Pattern Trigger

"Output must be sorted", "Merge Intervals", "Find Kth Largest", "Group Anagrams" $\rightarrow$ **Sorting**

Medium / Standard

- Sort Colors (Dutch National Flag). [LeetCode 75]
- Merge Intervals. [LeetCode 56]
- Non-overlapping Intervals. [LeetCode 435]
- Kth Largest Element in an Array (QuickSelect or Sort). [LeetCode 215]

Common Student Mistake

Mistake: Implementing Bubble Sort in an interview when asked for an optimal solution.
Fix: Always default to Merge Sort or Quick Sort logic if asked to implement from scratch. If using a library, know that Python/Java use Timsort ($O(N \log N)$).

Recommended Read (Deep Dive)

Read Article: Sorting Algorithms (GeeksforGeeks)
See visualizations of Merge Sort and Quick Sort to understand how they divide and conquer.

Pro Tip

Don't reinvent the wheel in Online Assessments (OA). Use `Arrays.sort()` (Java) or `sorted()` (Python) to save time, unless the question specifically forbids it.

Key Takeaway

The Constraint (N) is the most important hint in the question. It tells you the maximum allowable time complexity before you write a single line of code.

3 Phase 2: The Logic Patterns

Timeline: 3 Weeks

This phase moves you from brute-force guessing to pattern recognition, the hallmark of a skilled engineer.

"Okay, listen closely. This is the most important part of the entire roadmap. This is where most students feel lost. Suddenly, problems stop looking like arrays and start looking abstract. This confusion is expected. Do not jump topics. Patterns take time to form."

Mahad's Rule

If you stare at a problem for 15 minutes and can't find a pattern, stop. Go back and review the patterns below.

3.1 Hashing (The Superpower)

Concept in a Nutshell

Hashing allows us to look up data in $O(1)$ time. Whenever you see a problem asking for "frequency," "duplicates," or "pairs," think Hash Map immediately.

Pattern Trigger

If you hear "Frequency", "Duplicates", or "Find Pair" $\rightarrow$ **Hash Map / Set**

Easy / Warmup

- Two Sum. [LeetCode 1]
- Contains Duplicate. [LeetCode 217]
- Valid Anagram. [LeetCode 242]

Medium / Standard

- Group Anagrams. [LeetCode 49]
- Top K Frequent Elements. [LeetCode 347]
- Longest Consecutive Sequence. [LeetCode 128]

Hard / Master

- Subarray Sum Equals K. [LeetCode 560]
- Insert Delete GetRandom O(1). [LeetCode 380]

3.2 Two Pointers

Concept in a Nutshell

If the array is sorted, we can use two pointers moving from opposite ends (or same direction) to solve problems in $O(n)$ instead of $O(n^2)$.

Pattern Trigger

Sorted Array + ”Find Pair” or ”Target Sum” $\rightarrow$ **Two Pointers**

Easy / Warmup

- Valid Palindrome. [LeetCode 125]
- Merge Sorted Array. [LeetCode 88]

Medium / Standard

- Two Sum II (Sorted). [LeetCode 167]
- Container With Most Water. [LeetCode 11]
- 3Sum. [LeetCode 15]

Hard / Master

- Trapping Rain Water. [LeetCode 42]
- 4Sum. [LeetCode 18]

3.3 Sliding Window (The Subarray Killer)

Concept in a Nutshell

If you see ”Subarray of size K” or ”Longest Substring,” use a window that slides. Add the new element, remove the old one.

Pattern Trigger

”Longest Substring”, ”Shortest Subarray”, or ”Window of size K” $\rightarrow$ **Sliding Window**

Easy / Warmup

- Maximum Average Subarray I. [LeetCode 643]
- Substrings of Size Three with Distinct Characters. [LeetCode 1876]

Medium / Standard

- Longest Substring Without Repeating Characters. [LeetCode 3]
- Max Consecutive Ones III. [LeetCode 1004]
- Permutation in String. [LeetCode 567]

Hard / Master

- Minimum Window Substring. [LeetCode 76]
- Sliding Window Maximum. [LeetCode 239]

Pro Tip

Sliding Window is tricky. You will get ”Off-by-one” errors (e.g., does the window end at i or $i + 1$?). Debug by printing the window start/end indices at every step.

3.4 Binary Search (The Logarithmic King)

Concept in a Nutshell

Never search linearly ($O(n)$) in a sorted space. Divide and conquer ($O(\log n)$). This also works for ”Search Space” problems where the answer lies in a range.

Pattern Trigger

”Sorted” or ”Monotonic Function” $\rightarrow$ **Binary Search**

Easy / Warmup

- Binary Search. [LeetCode 704]
- Search Insert Position. [LeetCode 35]

Medium / Standard

- Find First and Last Position of Element. [LeetCode 34]
- Search in Rotated Sorted Array. [LeetCode 33]
- Find Peak Element. [LeetCode 162]
- Koko Eating Bananas (BS on Answer). [LeetCode 875]

Hard / Master

- Median of Two Sorted Arrays. [LeetCode 4]

- Aggressive Cows (SPOJ/GFG).

Mahad's Rule

The Overflow Bug: Never calculate mid as `(low + high) / 2`. If `low` and `high` are large integers, their sum will overflow. Always use `low + (high - low) / 2`.

Key Takeaway

Don't solve the problem. Identify the pattern. If you know it's a "Sliding Window" problem, the solution is already 50% written.

4 Phase 3: The Linear Structures

Timeline: 3 Weeks

This phase deals with how data is linked and accessed in memory, introducing pointers and dynamic structures.

“Alright, we are done with Arrays. Up until now, everything you stored was sitting nicely next to each other in memory. But the real world isn’t always comfortable. Pointers are annoying. You will get Segmentation Faults. This is the rite of passage.”

Mahad’s Rule

Draw it. If you try to write Linked List code without drawing boxes and arrows on paper, you will get a Segmentation Fault.

4.1 Linked Lists

Concept in a Nutshell

Nodes linked by pointers. Great for insertion/deletion, bad for random access.

Easy / Warmup

- Reverse Linked List. [LeetCode 206]
- Middle of the Linked List. [LeetCode 876]
- Delete Node in a Linked List. [LeetCode 237]

Medium / Standard

- Linked List Cycle II. [LeetCode 142]
- Palindrome Linked List. [LeetCode 234]
- Remove Nth Node From End. [LeetCode 19]
- Intersection of Two Linked Lists. [LeetCode 160]
- Add Two Numbers. [LeetCode 2]

Hard / Master

- Merge k Sorted Lists. [LeetCode 23]
- Reverse Nodes in k-Group. [LeetCode 25]
- LRU Cache. [LeetCode 146]

Common Student Mistake

Mistake: Accessing `head.next` without checking if `head` is `null`.
Result: Null Pointer Exception / Segmentation Fault. Always guard your pointers!

Recommended Read (Deep Dive)

Read Article: Linked List Introduction (GeeksforGeeks)
Visualize how pointers connect nodes in memory.

University Exam Lab Tasks

These questions appear frequently in University Exams & Technical Interviews:

1. **Sorted Insert:** Write a function to insert a value into a linked list such that the list remains sorted.
2. **Move Min/Max:** Find the Minimum value node and move it to the Head. Find Max value node and move it to the Tail.
3. **Copy Reverse:** Create a deep copy of a linked list, but in reverse order (without modifying the original).
4. **Doubly Linked List:** Implement Insert, Delete, and Print for a DLL.

4.2 Stacks & Queues

Concept in a Nutshell

Stack is LIFO (Last In First Out). Queue is FIFO (First In First Out).

Easy / Warmup

- Valid Parentheses. [LeetCode 20]
- Implement Queue using Stacks. [LeetCode 232]

Medium / Standard

- Min Stack. [LeetCode 155]
- Daily Temperatures (Monotonic Stack). [LeetCode 739]
- Next Greater Element I. [LeetCode 496]

Hard / Master

- Largest Rectangle in Histogram. [LeetCode 84]
- Trapping Rain Water (Stack Solution). [LeetCode 42]

Common Student Mistake

Mistake: Forgetting edge cases (Empty list, Single node).
Fix: Always test your code against an empty input.

Key Takeaway

Pointers are just addresses. Don't fear them. If you lose a reference to a node (by overwriting a pointer), it is gone forever. Order of operations matters!

5 Phase 4: The Non-Linear World

Timeline: 4 Weeks

This phase introduces hierarchical data structures, where relationships are parent-child rather than sequential.

"Welcome to the jungle. Your brain will fight Recursion. It wants to trace every step. Do not let it. This is about trust."

Mahad's Rule

My Personal Struggle: I got stuck on Recursion for weeks. I felt stupid. But one day, I stopped trying to trace every step and just "trusted" the function. That's when Trees clicked.

5.1 Recursion & Backtracking

Concept in a Nutshell

A function calling itself. Trust the base case. Backtracking is just recursion with an "undo" button.

Easy / Warmup

- Fibonacci Number. [LeetCode 509]
- Power of Three. [LeetCode 326]

Medium / Standard

- Subsets. [LeetCode 78]
- Permutations. [LeetCode 46]
- Combination Sum. [LeetCode 39]
- Generate Parentheses. [LeetCode 22]

Hard / Master

- N-Queens. [LeetCode 51]
- Sudoku Solver. [LeetCode 37]

Pro Tip

Remember that Recursion uses the Stack Memory. If you go too deep ($> 10^4$), you will get a Stack Overflow Error. This is essentially the Space Complexity of your recursive solution.

5.2 Trees & BST

Concept in a Nutshell

Hierarchical data. Most problems are solved using DFS (Pre/In/Post) or BFS (Level Order).

Easy / Warmup

- Maximum Depth of Binary Tree. [LeetCode 104]
- Invert Binary Tree. [LeetCode 226]
- Same Tree. [LeetCode 100]

Medium / Standard

- Binary Tree Level Order Traversal. [LeetCode 102]
- Validate Binary Search Tree. [LeetCode 98]
- Lowest Common Ancestor. [LeetCode 236]
- Diameter of Binary Tree. [LeetCode 543]

Hard / Master

- Binary Tree Maximum Path Sum. [LeetCode 124]
- Serialize and Deserialize Binary Tree. [LeetCode 297]

Recommended Read (Deep Dive)

Read Article: Tree Traversals (GeeksforGeeks)
See the difference between Inorder, Preorder, and Postorder visually.

5.3 Heaps (Priority Queue)

Concept in a Nutshell

A complete binary tree used to efficiently find the Max or Min element ($O(1)$).

Pattern Trigger

”Top K Elements”, ”Kth Largest/Smallest”, or ”Median of Stream” $\rightarrow$ **Heap (Priority Queue)**

Medium / Standard

- Kth Largest Element in an Array. [LeetCode 215]

- Top K Frequent Elements. [LeetCode 347]
- K Closest Points to Origin. [LeetCode 973]

Key Takeaway

The ”Leap of Faith” is real. Assume your recursive function works for the child node, and just write the logic for the current node.

6 Phase 5: The Endgame

Timeline: 3 Weeks

This phase covers complex relationships (Graphs) and optimization (DP), separating the good engineers from the great ones.

"This is it. The final boss. If Trees were a hierarchy, Graphs are a network. Think of Facebook friends or Google Maps. This looks scary, but Graphs are just Trees with loops. DP is just Recursion with a notebook."

6.1 Graphs

Concept in a Nutshell

Nodes and Edges. BFS finds shortest path in unweighted graphs. DFS explores deep.

Medium / Standard

- Number of Islands. [LeetCode 200]
- Rotting Oranges. [LeetCode 994]
- Course Schedule. [LeetCode 207]
- Clone Graph. [LeetCode 133]

Hard / Master

- Word Ladder. [LeetCode 127]
- Alien Dictionary (Topological Sort). [LeetCode 269]

Common Student Mistake

Mistake: Forgetting the `visited` array in Graphs.
Result: Infinite loops and Stack Overflow. Unlike Trees, Graphs have cycles!

Recommended Read (Deep Dive)

Read Article: BFS vs DFS (GeeksforGeeks)
Comparison table on when to use which traversal.

6.2 Greedy Algorithms

Concept in a Nutshell

Making the locally optimal choice at each stage with the hope of finding a global optimum.

Medium / Standard

- Jump Game. [LeetCode 55]
- Maximum Subarray. [LeetCode 53]
- Gas Station. [LeetCode 134]

6.3 Dynamic Programming

Concept in a Nutshell

Recursion + Caching (Memoization) or Iterative Filling (Tabulation).

Easy / Warmup

- Climbing Stairs. [LeetCode 70]
- Min Cost Climbing Stairs. [LeetCode 746]

Medium / Standard

- House Robber. [LeetCode 198]
- Coin Change. [LeetCode 322]
- Longest Common Subsequence. [LeetCode 1143]
- Maximum Product Subarray. [LeetCode 152]

Hard / Master

- Edit Distance. [LeetCode 72]
- Longest Increasing Subsequence. [LeetCode 300]

Pro Tip

For DP, always solve it using Recursion first. Then add Memoization. Only then try Tabulation. Don't jump to the table directly.

Key Takeaway

Graph Theory and DP are not about "coding" complexity. They are about "state" management. Can you define the state of your problem at any given moment? If yes, you can solve it.

Life After DSA: What Comes Next?

DSA is a tool, not the destination. Once you are comfortable, you must move on to building real things.

- **Build Real Projects:** Apply your logic to Web Development or App Development. A balanced engineer has both DSA skills and Dev skills.
- **Start Mock Interviews:** Coding on a whiteboard (or a Google Doc) is very different from coding in an IDE with auto-complete. Practice explaining your thought process out loud.
- **System Design:** Learn how to scale applications. Read about High-Level Design (HLD) and Low-Level Design (LLD).
- **Revisit:** DSA is volatile. You will forget it. Solve 1-2 problems on weekends to keep the muscle memory alive.

Appendix A: The Curated Problem Vault

Note: These sheets are optional resources. Do not try to do everything at once. One sheet done properly is better than three sheets done halfway.

1. Structured Learning (Beginner to Pro)

- **Striver's A2Z DSA Sheet:** Link to Sheet. The most comprehensive roadmap available. Best for step-by-step mastery.
- **NeetCode 150:** Link to Sheet. Excellent for visual learners and interview-focused revision.

2. The Interview Cram (Time Crunch)

- **Blind 75:** Link to List. The 75 most asked questions in FAANG history. No fluff. If you have 1 month left, do this.

3. Competitive Programming (Optional Track)

- **CSES Problem Set:** Link to Problems. The holy grail for mathematical DSA and advanced logic.

- **Codeforces Div 2 (A/B):** Link to Site. Best for building raw speed and logical thinking under pressure.

Appendix B: The 16-Week Printable Tracker

Instructions: Print this page. Stick it on your wall. Tick the boxes as you complete each week. Consistency is your only currency.

Week	Phase	Focus & Goals	Done
1	Phase 0	Syntax Fluency, Nested Loops, Pattern Printing	■
2	Phase 0	OOP Basics, Memory (Stack/Heap), Pass by Ref	■
3	Phase 1	Time Complexity ($O(N)$ vs $O(N^2)$), Basic Arrays	■
4	Phase 1	String Manipulation, Basic Number Theory, Sorting	■
5	Phase 2	Hashing (Sets/Maps), Frequency Counting	■
6	Phase 2	Two Pointers (Sorted Arrays), Sliding Window	■
7	Phase 2	Binary Search (Search Space & Rotated Arrays)	■
8	Phase 3	Linked Lists (Reverse, Cycle Detection)	■
9	Phase 3	Linked Lists (Merge, Remove Nth Node)	■
10	Phase 3	Stacks (Valid Parentheses) & Queues	■
11	Phase 4	Recursion Basics (Factorial, Fibonacci)	■
12	Phase 4	Backtracking (Subsets, Permutations)	■
13	Phase 4	Trees (DFS Traversals, Max Depth)	■
14	Phase 4	BST Validation, Heaps (Priority Queue)	■
15	Phase 5	Graphs (BFS/DFS, Number of Islands)	■
16	Phase 5	Dynamic Programming (Climbing Stairs, Robber)	■

✓ Accountability Check

I, ________________________, commit to solving at least 1 problem every single day for the next 16 weeks.

Mahad's 7 Golden Rules of DSA

If you forget everything else in this book, remember these 7 rules.

1. **Consistency beats Intelligence.** A genius who codes once a month will lose to an average student who codes every day.
2. **One solved problem > Ten watched videos.** Tutorials give you a false sense of competence. Code is the only truth.
3. **Confusion is a signal of growth.** If you are confused, it means your brain is trying to build a new connection. Do not quit.
4. **If DP feels impossible, you are learning it right.** It is supposed to be hard. Give it time.
5. **Constraints are your best friend.** Always look at N. It tells you the time complexity before you even write code.
6. **Dry Run before you Run.** If you can't solve it on paper, you can't solve it on the computer.
7. **You are not late.** You are exactly where you need to be. Start today.

Share & Connect

Knowledge compounds when shared.
If this blueprint helped you, consider passing it forward to someone who might need it. Let's grow together.

Follow me for the Solution Repository:

LinkedIn: @codewithmahad
GitHub: @mahad2006

About the Author

Hi, I'm Shaikh Mahad. I am currently a 3rd Semester BSSE student (Entry Batch 2025). Like many of you, I initially found Data Structures and Algorithms overwhelming. I realized that while there are thousands of tutorials online, there was no clear structure for a university student starting from scratch.
I wrote **The DSA Blueprint** to be the guide I wish I had when I started. It is my attempt to organize the noise into a clear path. Whether you are aiming for a high GPA or a tech career, I hope this document saves you time. If this document saved you time or confusion, it has done its job.

"The algorithms you struggle with today will be the warm-ups you breeze through tomorrow."

This path is difficult. You will get stuck. You will doubt yourself.
But remember: **The difficulty is the point.** If it were easy, everyone would do it.

Let's Crack the Code.

— Shaikh Mahad

Coming Soon

Part 2: The Learning Notes

You now have the Roadmap (This Document). Next, you need the Knowledge.

I am currently writing **Part 2: The "Zero-to-Hero" DSA Notes**. Unlike standard textbooks, these notes are being hand-written specifically for university students, featuring:

- ✓ **Visual Diagrams** for every algorithm.
- ✓ **Real-world Analogies** (No boring definitions).
- ✓ **Exam-Focused Solutions** for Semester Lab Tasks.

Want Early Access?

Star the repository on GitHub and follow me on LinkedIn. I will be releasing early chapters to my community first.

Star on GitHub **Follow on GitHub**

Stay Tuned.