Cracking the Big Tech Interview

The Comprehensive Handbook of LeetCode Patterns

Synthesized High-Yield Strategies for FAANG/MAMAA

January 18, 2026

Contents

Introduction: The Philosophy of Patterns

Why Patterns?

Top-tier tech companies (Google, Meta, Amazon) rarely ask "textbook" questions anymore. Instead, they present novel problems that are actually **disguised versions** of standard algorithmic patterns.
Mastery is not solving 1000 problems; it is solving 150 problems that represent the 15 core patterns. This document catalogues those patterns, the "clues" to spot them, and the specific problems to practice.

I. Arrays, Strings & Pointers

1. Sliding Window

The Concept: Instead of iterating over all subarrays (which is $O(N^2)$), we define a window (start and end pointers) that slides over the data to satisfy a condition. The window size can be **fixed** (e.g., sum of subarray size 3) or **dynamic** (e.g., smallest subarray with sum $>$ S).

Visual Clues (When to use):

- The problem involves a linear data structure (Array, String, Linked List).
- You are asked to find the *longest/shortest* substring, subarray, or specific value.
- Keywords: "Contiguous", "Subarray", "Substring".

The Approach:

1. Initialize 'windowStart = 0' and 'currentSum' (or generic counter).
2. Iterate 'windowEnd' from 0 to 'len(arr)'.
3. Add 'arr[windowEnd]' to 'currentSum'.
4. **While** the condition is broken (e.g., window sum is too large), shrink the window from the left:
   - Remove 'arr[windowStart]' from 'currentSum'.
   - Increment 'windowStart'.
5. Update the global maximum/minimum result.

Practice Problems:

- **[Easy]** Maximum Sum Subarray of Size K
- **[Med]** Longest Substring Without Repeating Characters (LeetCode #3)
- **[Med]** Longest Repeating Character Replacement (LeetCode #424)
- **[Hard]** Minimum Window Substring (LeetCode #76)

---

2. Two Pointers

The Concept: Using two pointers to traverse an array, usually from distinct ends, to find a set of elements that fulfill a constraint. This reduces time complexity from $O(N^2)$ to $O(N)$.

Visual Clues:

- The input array is **sorted** (critical hint).
- You need to find a pair, triplet, or triplet sum equals a target.
- You need to compare elements from the ends of the array.

The Approach:

1. Place ‘left‘ pointer at index 0, ‘right‘ at index ‘n-1‘.
2. Loop while ‘left ¡ right‘:
   - Calculate current metric (e.g., ‘sum = arr[left] + arr[right]‘).
   - If ‘sum == target‘, return result.
   - If ‘sum ¡ target‘, increment ‘left‘ (need larger sum).
   - If ‘sum ¿ target‘, decrement ‘right‘ (need smaller sum).

Practice Problems:

- **[Easy]** Two Sum II - Input Array Sorted (LeetCode #167)
- **[Easy]** Squares of a Sorted Array (LeetCode #977)
- **[Med]** 3Sum (LeetCode #15)
- **[Med]** Container With Most Water (LeetCode #11)
- **[Hard]** Trapping Rain Water (LeetCode #42)

---

3. Fast & Slow Pointers (Tortoise Hare)

The Concept: A specific two-pointer approach where one moves 1 step and the other moves 2 steps. If there is a cycle, they are guaranteed to meet.

Visual Clues:

- Linked Lists or Arrays with cyclic dependencies.
- Need to find the ”middle” of a Linked List in one pass.
- Determine if a cycle exists.

Practice Problems:

- **[Easy]** Linked List Cycle (LeetCode #141)
- **[Easy]** Middle of the Linked List (LeetCode #876)
- **[Med]** Find the Duplicate Number (LeetCode #287) - *Pigeonhole Principle variant*
- **[Med]** Happy Number (LeetCode #202)

II. Linked List Patterns

4. In-place Reversal of a Linked List

The Concept: Reversing nodes without extra memory by manipulating ‘next‘ pointers.

The Approach:

1. Maintain three pointers: ‘prev‘, ‘curr‘, ‘next‘.
2. Loop ‘while curr is not null‘:
   - ‘next = curr.next‘ (save the next node)
   - ‘curr.next = prev‘ (reverse the arrow)
   - ‘prev = curr‘ (move forward)
   - ‘curr = next‘ (move forward)

Practice Problems:

- **[Easy]** Reverse Linked List (LeetCode #206)
- **[Med]** Reverse Linked List II (Range Reversal) (LeetCode #92)
- **[Hard]** Reverse Nodes in k-Group (LeetCode #25)

III. Tree & Graph Traversal

5. Tree BFS (Level Order Traversal)

The Concept: Process a tree level-by-level using a **Queue**. Essential for finding the shortest path in unweighted graphs.

The Approach:

1. Push ‘root‘ to Queue.
2. While Queue is not empty:
   - Get ‘levelSize = queue.length‘.
   - Loop ‘levelSize‘ times (process current level):
     - Pop node, process it.
     - Push left child, Push right child.

Practice Problems:

- **[Med]** Binary Tree Level Order Traversal (LeetCode #102)
- **[Med]** Binary Tree Zigzag Level Order Traversal (LeetCode #103)
- **[Easy]** Minimum Depth of Binary Tree (LeetCode #111)

6. Tree DFS (Depth First Search)

The Concept: Explore a branch as deep as possible before backtracking. Implemented via **Recursion** or a **Stack**.

Visual Clues:

- You need to traverse Pre-order, In-order, or Post-order.
- Problems asking for "Path Sum" or valid sequences from root to leaf.

Practice Problems:

- [**Easy**] Path Sum (LeetCode #112)
- [**Med**] Path Sum II (All paths) (LeetCode #113)
- [**Hard**] Binary Tree Maximum Path Sum (LeetCode #124)

7. Topological Sort (Graph)

The Concept: Used for Directed Acyclic Graphs (DAGs). It produces a linear ordering of vertices such that for every edge $U \to V$, U comes before V.

Visual Clues:

- Prerequisites, Task Scheduling, Dependency Resolution.
- "Can you finish all courses?"

The Approach (Kahn's Algorithm):

1. Calculate in-degrees (number of incoming edges) for all vertices.
2. Push all vertices with 'in-degree == 0' to a Queue.
3. While Queue is not empty:
   - Pop vertex 'U', add to result list.
   - For each neighbor 'V' of 'U':
     - Decrement in-degree of 'V'.
     - If in-degree becomes 0, push 'V' to Queue.

Practice Problems:

- [**Med**] Course Schedule (LeetCode #207)
- [**Med**] Course Schedule II (LeetCode #210)
- [**Hard**] Alien Dictionary (LeetCode #269)

IV. Heaps, Intervals Optimization

8. Two Heaps

The Concept: Using a Min-Heap and a Max-Heap to divide a dataset into two halves. This allows finding the **Median** of a stream in $O(1)$ time.

Visual Clues:

- Scheduling, Prioritization.

- Situations where you need quick access to the median, max, or min dynamically.

Practice Problems:

- **[Hard]** Find Median from Data Stream (LeetCode #295)
- **[Hard]** Sliding Window Median (LeetCode #480)

9. Merge Intervals

The Concept: Dealing with overlapping time intervals. **Always sort the intervals by start time first.**

Visual Clues:

- "Overlapping", "Merge", "Meeting Times".

Practice Problems:

- **[Med]** Merge Intervals (LeetCode #56)
- **[Med]** Insert Interval (LeetCode #57)
- **[Med]** Meeting Rooms II (LeetCode #253)

10. Top 'K' Elements

The Concept: Finding the K largest, smallest, or most frequent elements. The naive sort is $O(N \log N)$. Using a Heap reduces this to $O(N \log K)$.

Practice Problems:

- **[Med]** Top K Frequent Elements (LeetCode #347)
- **[Med]** Kth Largest Element in an Array (LeetCode #215)
- **[Med]** K Closest Points to Origin (LeetCode #973)

V. Advanced Algorithmic Patterns

11. Modified Binary Search

The Concept: Binary search on data that isn't perfectly sorted (e.g., rotated) or has unknown length. **Practice Problems:**

- **[Med]** Search in Rotated Sorted Array (LeetCode #33)
- **[Med]** Find First and Last Position of Element (LeetCode #34)

12. 0/1 Knapsack (Dynamic Programming)

The Concept: Given a set of items with weights and values, determine the items to include to maximize value within a capacity. **Practice Problems:**

- **[Med]** Partition Equal Subset Sum (LeetCode #416)
- **[Med]** Target Sum (LeetCode #494)

13. Unbounded Knapsack

The Concept: Similar to 0/1, but items can be used unlimited times. **Practice Problems:**

- **[Med]** Coin Change (LeetCode #322)
- **[Med]** Rod Cutting (GeeksForGeeks)

14. Monotonic Stack

The Concept: Maintaining a stack where elements are always sorted (increasing or decreasing). Used to find "Next Greater Element" or "Previous Smaller Element" in $O(N)$. **Practice Problems:**

- **[Med]** Daily Temperatures (LeetCode #739)
- **[Hard]** Largest Rectangle in Histogram (LeetCode #84)

VI. The Final Checklist

Before your interview, ensure you can:

1. Identify the pattern from the problem statement (the "Visual Clues").
2. Write the "Template Code" for BFS, DFS, Sliding Window, and Binary Search from scratch without errors.
3. Explain the Time and Space complexity of your solution confidently ($O(N)$ vs $O(N \log N)$).