ChatGPT

Essential DSA Patterns for Tech Interviews

Below is a comprehensive list of **Data Structures and Algorithms (DSA) patterns** that an experienced full-stack developer (11+ years) should master to crack coding interviews at MAANG/Microsoft and other top product-based companies [1]. The patterns are organized by topic (Arrays, Trees, Graphs, etc.), with common sub-patterns or problem types. Under each pattern, we list representative LeetCode and GeeksforGeeks problems, grouped by difficulty (Easy, Medium, Hard), and relevant to languages like Python, JavaScript, or C#.

Arrays & Strings

- **Two Pointers Pattern:** Common for sorted arrays or partitioning problems (using two indices moving towards center).
- *Easy:* Two Sum (LeetCode) – Two Sum (GfG) – *Find two numbers in an array that add up to a target* [2].
- *Medium:* 3Sum (LeetCode) – Triplet Sum to Zero (GfG) – *Find all unique triplets that sum to zero* [3].
- *Hard:* Trapping Rain Water (LeetCode) – Trapping Rain Water (GfG) – *Compute water trapped between bars using two-pointer scan* [4] [5].
- **Sliding Window Pattern:** Handling subarrays or substrings by maintaining a window of elements.
- *Easy:* Maximum Average Subarray I (LeetCode) – Max Avg Subarray of Size K (GfG) – *Find contiguous subarray of length* k *with maximum average (use window sum)* [6].
- *Medium:* Longest Substring Without Repeating Characters (LeetCode) – Longest Distinct Characters in Substring (GfG) – *Use sliding window to find longest unique-char substring*.
- *Hard:* Sliding Window Maximum (LeetCode) – Max of Subarrays of Size K (GfG) – *Use deque/queue to get max in every window of size* k**.
- **Kadane's Algorithm (Max Subarray):** Dynamic programming approach for subarray problems.
- *Easy:* Maximum Subarray (LeetCode) – Kadane's Algorithm (GfG) – *Find the largest sum contiguous subarray (O(n) DP solution)*.
- *Medium:* Maximum Product Subarray (LeetCode) – Max Product Subarray (GfG) – *Compute largest product of a contiguous subarray (handling negatives)*.
- *Hard:* Subarray Sum Equals K (LeetCode) – Subarray with Given Sum (GfG) – *Find number of subarrays that sum to* k *(uses prefix sum + hashing)*.

(Other Array/String patterns to master: Cyclic Sort for finding missing/duplicate numbers, Merge Intervals for interval merging, Two-pointer variants for Dutch National Flag (sort colors) etc.)

Linked Lists

- **Fast & Slow Pointers (Cycle Detection):** Using two pointers moving at different speeds to find cycles or middle.
- *Easy:* Middle of Linked List (LeetCode) – Middle of Linked List (GfG) – *Use slow/fast pointer to find midpoint of list*.
- *Medium:* Linked List Cycle (LeetCode) – Detect Loop in Linked List (GfG) – *Detect if a cycle exists in a linked list (Floyd's cycle algorithm)*.

- *Medium:* Linked List Cycle II (LeetCode) – Find Loop Starting Node (GfG) – *Find the starting node of the cycle in a linked list*.

- **In-place Reversal Pattern:** Reversing links in various scenarios.

- *Easy:* Reverse Linked List (LeetCode) – Reverse a Linked List (GfG) – *Iteratively or recursively reverse a singly linked list*.
- *Medium:* Reverse Linked List II (LeetCode) – Reverse a Linked List in Groups (GfG) – *Reverse nodes between positions* m *and* n*, or in k-sized groups*.
- *Hard:* Reverse Nodes in k-Group (LeetCode) – *Reverse the list in groups of k (advanced pointer manipulation)*.
- *Hard:* LRU Cache (LeetCode) – LRU Cache Implementation (GfG) – *Design a Least-Recently-Used cache using doubly-linked list + hashing*.

(Also practice merging and rearrangement: Merge Two Sorted Lists (LC 21 – easy), Reorder List (LC 143 – medium), etc.)

Stacks & Queues

- **Parentheses & Stack Usage:** Problems involving matching or removing brackets using a stack.
- *Easy:* Valid Parentheses (LeetCode) – Balanced Parentheses (GfG) – *Check if brackets are correctly balanced using a stack*.
- *Medium:* Min Remove to Make Valid String (LeetCode) – Remove Invalid Parentheses (GfG) – *Remove minimum brackets to balance the string*.

- *Hard:* Longest Valid Parentheses (LeetCode) – Longest Valid Parentheses (GfG) – *Find length of longest valid (well-formed) parentheses substring*.

- **Monotonic Stack Pattern:** Using a stack to maintain increasing/decreasing sequences (for span or next greater/smaller problems).

- *Easy:* Next Greater Element I (LeetCode) – Next Greater Element (GfG) – *Find the next greater element for each element in an array*.
- *Medium:* Next Greater Element II (LeetCode) – *Circular array variant of next greater (wrap-around handling)*.
- *Hard:* Largest Rectangle in Histogram (LeetCode) – Largest Rectangle in Histogram (GfG) – *Use a stack to compute max area in a histogram*.

- *Hard:* Maximal Rectangle (LeetCode) – *Extend histogram logic to 2D matrix of 0/1 (largest rectangle of 1s).*

- **Queue & Deque Applications:**

- *Medium:* Implement Queue using Stacks (LeetCode) – *Use two stacks to simulate a queue.*
- *Medium:* Sliding Window Maximum (LeetCode) – Deque for Sliding Window (GfG) – *Use deque to maintain max of current window* (monotonic queue technique).
- *Medium:* Open the Lock (LeetCode) – *Use BFS with a queue for shortest steps in state-space (lock combination puzzle).*
- *Medium:* Time Needed to Inform All Employees (LeetCode) – *Use queue for level-order propagation (multi-source BFS).*

Trees & Binary Trees

- **Depth-First Search (DFS) – Recursion on Trees:** Traversals and recursive tree problems.
- *Easy:* Maximum Depth of Binary Tree (LeetCode) – Height of Binary Tree (GfG) – *Compute the max depth of a binary tree (DFS recursion).*
- *Medium:* Lowest Common Ancestor of Binary Tree (LeetCode) – Lowest Common Ancestor in BT (GfG) – *Find LCA of two nodes in a binary tree (DFS search).*
- *Hard:* Binary Tree Maximum Path Sum (LeetCode) – Max Path Sum in BT (GfG) – *DFS to find the maximum sum of any path in the tree.*

- *Hard:* Serialize and Deserialize Binary Tree (LeetCode) – Serialize/Deserialize Tree (GfG) – *Use DFS or BFS to convert a tree to/from string representation.*

- **Breadth-First Search (BFS) – Level Order Traversal:** Iterative level-by-level traversal with a queue.

- *Easy:* Level Order Traversal (LeetCode) – Level Order Traversal (GfG) – *Traverse the binary tree level by level using a queue.*
- *Medium:* Binary Tree Zigzag Level Order (LeetCode) – Zigzag Level Order (GfG) – *Alternating direction at each level using BFS.*
- *Medium:* Binary Tree Right Side View (LeetCode) – *Use BFS (or DFS) to capture the rightmost node at each level.*

- *Hard:* Populating Next Right Pointers (LeetCode) – Connect Nodes at Same Level (GfG) – *Use BFS to connect each node with its right neighbor in the same level.*

- **Binary Search Tree (BST) Patterns:** Exploiting BST properties (inorder sorted, left<root<right).

- *Easy:* Validate Binary Search Tree (LeetCode) – Check BST (GfG) – *Use DFS inorder traversal to validate BST property.*
- *Medium:* Lowest Common Ancestor of BST (LeetCode) – LCA in BST (GfG) – *Find LCA efficiently by traversing down BST (using BST ordering).*
- *Medium:* Insert into BST (LeetCode) – *Insert a value into a BST (simple recursive or iterative insertion).*
- *Hard:* Recover Binary Search Tree (LeetCode) – Fix Swapped Nodes in BST (GfG) – *Inorder traversal to recover BST where two nodes are swapped (restore order).*

(Also practice Tree Traversals (preorder, inorder, postorder iterative) and Balanced BST operations as needed.)

Heaps & Priority Queues

- **Top-K Elements Pattern:** Using a min-heap or max-heap for finding Kth largest/smallest or top K frequent elements.
- *Easy:* Kth Smallest Element in Array (LeetCode) – Kth Smallest Element (GfG) – *Find Kth smallest (or largest) element in an unsorted array* (use heap or QuickSelect) [7] [8].
- *Medium:* Top K Frequent Elements (LeetCode) – Top K Frequent Numbers (GfG) – *Use hash map + max-heap (or bucket sort) to get k most frequent elements*.
- *Medium:* Task Scheduler (LeetCode) – Rearrange Task Scheduling (GfG) – *Greedy + max-heap to schedule tasks with cooldown intervals*.
- *Hard:* Merge k Sorted Lists (LeetCode) – Merge K Sorted Linked Lists (GfG) – *Use a min-heap to efficiently merge k sorted lists*.
- **Two-Heaps Pattern:** Maintaining two heaps (min-heap and max-heap) for median or balanced partition.
- *Hard:* Find Median from Data Stream (LeetCode) – Median in a Stream (GfG) – *Use two heaps (max-heap for lower half, min-heap for upper half) to get median in O(log n) per entry*.
- *Hard:* Sliding Window Median (LeetCode) – *Find median of each window (balance two heaps as window moves)*.

(Also be familiar with Heap Sort, PriorityQueue APIs in your language, and using heaps in graph algorithms like Dijkstra.)

Hashing

- **Hash Maps/Sets for Lookup:** Using hashing to achieve O(1) average lookups for various problems.
- *Easy:* Contains Duplicate (LeetCode) – *Use a hash set to check if any number appears more than once*.
- *Easy:* Valid Anagram (LeetCode) – Check Anagrams (GfG) – *Use a hash map or array count to verify if two strings are anagrams*.
- *Medium:* Two Sum (LeetCode) – *Use hash map to find complement in O(1) time* (alternate to two-pointer) [9].
- *Medium:* Subarray Sum Equals K (LeetCode) – Subarray with 0 Sum (GfG) – *Use hash map of prefix sums to count subarrays summing to k*.
- *Hard:* Longest Consecutive Sequence (LeetCode) – Longest Consecutive Subsequence (GfG) – *Use a hash set to find the length of the longest run of consecutive integers*.
- *Hard:* Substring with Concatenation of All Words (LeetCode) – *Use hash maps for word counts and sliding window to find substring containing all words*.
- **Hashing for** Big-O **Optimization:**
- *Medium:* LRU Cache (LeetCode) – *Use hash map + linked list for O(1) cache operations*.

- *Medium:* Group Anagrams (LeetCode) – Group Anagrams (GfG) – *Use hashing of sorted string or char counts to group anagrams*.
- *Hard:* Design HashMap (LeetCode) – *Implement a hash map using chaining or open addressing (low-level understanding of hashing)*.

(Master common uses of hash sets/maps for caching results, deduplicating data, frequency counting, etc.)

Recursion & Backtracking

- **Subsets & Permutations (Backtracking):** Generating combinations, subsets, and permutations via recursion.
- *Medium:* Subsets (LeetCode) – Power Set (GfG) – *Use backtracking to generate all subsets of a set (all combinations of elements)*.
- *Medium:* Permutations (LeetCode) – Permutations of String (GfG) – *Generate all permutations of a list or string via DFS/backtracking*.
- *Medium:* Combination Sum (LeetCode) – Combination Sum (GfG) – *Backtrack to find all combinations of candidates summing to target (choose/un-choose pattern)*.
- *Hard:* Permutation Sequence (LeetCode) – Nth Lexicographic Permutation (GfG) – *Find the k-th permutation of n numbers (factorial number system / backtracking)*.
- **Constraint Satisfaction & Puzzles:** Using backtracking to build solutions under constraints.
- *Hard:* N-Queens (LeetCode) – N-Queens Problem (GfG) – *Place N queens on an N×N board so that no two attack each other (backtracking with pruning)*.
- *Hard:* Sudoku Solver (LeetCode) – Sudoku Solver (GfG) – *Fill a 9×9 Sudoku grid by backtracking, respecting constraints*.
- *Hard:* Word Search II (LeetCode) – Word Boggle (GfG) – *Backtrack through a board to find words (often optimized with a Trie for pruning)*.
- *Hard:* Regex Matching (LeetCode) – *Backtracking and DP to match regex patterns (complex backtracking state)*.

(Tip: Understand recursion depth and state restoration for backtracking. Practice generating combinatorial solutions and solving small-scale search problems.)

Binary Search

- **Classic Binary Search:** Dividing search space by half each time (works on sorted arrays).
- *Easy:* Binary Search (LeetCode) – Binary Search (GfG) – *Standard binary search for a target in a sorted array*.
- *Medium:* First and Last Position in Sorted Array (LeetCode) – First/Last Occurrence (GfG) – *Use binary search variants to find boundaries (lower and upper bound)*.
- *Medium:* Search in Rotated Sorted Array (LeetCode) – Search in Rotated Array (GfG) – *Binary search on a rotated sorted array (find pivot then binary search)*.
- *Medium:* Peak Element (LeetCode) – Peak Element (GfG) – *Find any peak (element greater than neighbors) via binary search on unimodal array*.

- *Hard:* Median of Two Sorted Arrays (LeetCode) – *Binary search partitioning approach to find median of two sorted arrays.*
- *Hard:* Split Array Largest Sum (LeetCode) – Allocate Minimum Pages (GfG) – *Binary search on answer (find minimum possible "max subarray sum" when splitting array into m parts)* [1].
- *Hard:* Aggressive Cows / Place Cameras (Misc) – *Binary search on the minimum distance feasible for placing items given constraints.*

(Key is to recognize when a problem's search space can be binary searched even if not directly searching in a sorted array – e.g., searching for a minimum feasible value.)

Dynamic Programming

- **Fibonacci & 1-D DP:** Simple DP relations that build on previous states (typically O(n) or O(n^2)).
- *Easy:* Climbing Stairs (LeetCode) – Nth Fibonacci (GfG) – *Classic DP: f(n) = f(n-1) + f(n-2) for counting ways (stairs, Fibonacci).*
- *Medium:* House Robber (LeetCode) – Stickler Thief (GfG) – *1-D DP: maximize sum with no adjacent elements (choose or skip pattern).*
- *Medium:* Coin Change (LeetCode) – Coin Change (GfG) – *Unbounded knapsack: minimum coins to make a given amount (DP over amount).*
- *Hard:* Jump Game II (LeetCode) – *Min steps to reach end of array (can be solved greedily or via DP).*

- *Hard:* Russian Doll Envelopes (LeetCode) – *Variations of Longest Increasing Subsequence (n log n solution with DP & binary search).*

- **0/1 Knapsack & Subset DP:** DP for selection problems (choose or not choose each item).

- *Medium:* Partition Equal Subset Sum (LeetCode) – Subset Sum Problem (GfG) – *Determine if subset sums to half of total (DP boolean table).*
- *Medium:* Target Sum (LeetCode) – *Count ways to assign +/- to make sum S (can reduce to subset sum count DP).*
- *Hard:* 0/1 Knapsack (Classic) – *Maximize value within weight capacity (DP table of items vs weight).*

- *Hard:* Balanced Partition / Min Subset Sum Diff (GfG) – *DP to find subset minimizing difference to total/2.*

- **Two-Sequences DP:** DP on strings/sequences (typically 2D DP table).

- *Medium:* Longest Common Subsequence (LeetCode) – LCS Problem (GfG) – *Classic DP: find length of longest subsequence present in two sequences.*
- *Medium:* Longest Palindromic Subsequence (LeetCode) – Longest Palindromic Subsequence (GfG) – *LCS variation (string vs its reverse).*
- *Hard:* Edit Distance (LeetCode) – Edit Distance (GfG) – *Minimum operations to convert one string to another (DP matrix of sizes m×n).*

- *Hard:* Wildcard Matching (LeetCode) – Wildcard Pattern Matching (GfG) – *DP for matching string with pattern (handles* `?` *and* `*` *wildcards).*

- **Matrix/Grid DP:** DP on a 2D grid for path counting or cost optimization.

- *Easy:* Min Path Sum (LeetCode) – Min Cost Path (GfG) – *DP to find minimum sum path from top-left to bottom-right of grid*.
- *Medium:* Unique Paths (LeetCode) – Number of Paths in Grid (GfG) – *Count ways to reach bottom-right from top-left in a grid (combinatorial DP)*.
- *Medium:* Coin Change 2 (LeetCode) – *Count ways to make amount with coins (2D DP: coins vs amount)*.
- *Hard:* Dungeon Game (LeetCode) – *DP from bottom-right to top-left to ensure minimum health needed to survive a grid of dangers*.
- *Hard:* Matrix Chain Multiplication (GfG) – *DP for optimal parenthesization of matrix multiplication (minimize cost)*.

(Also explore Bitmask DP for advanced problems like Traveling Salesman (TSP), though these are less common in interviews.)

Graphs

- **Graph Traversal (BFS & DFS):** Explore all nodes in graph; used for connectivity, island counting, etc.
- *Easy:* Flood Fill (LeetCode) – Flood Fill (GfG) – *DFS/BFS to recolor connected region in a grid*.
- *Medium:* Number of Islands (LeetCode) – Number of Islands (GfG) – *Count connected components in a 2D grid using DFS/BFS*.
- *Medium:* Clone Graph (LeetCode) – Clone Graph (GfG) – *Use DFS/BFS with a hash map to clone a graph*.
- *Medium:* Pacific Atlantic Water Flow (LeetCode) – *Multi-source DFS/BFS to find cells reachable to both Pacific and Atlantic edges*.

- *Hard:* Word Ladder (LeetCode) – Word Ladder (GfG) – *BFS on word transformations to find shortest transformation sequence length*.

- **Graph Shortest Paths:** Finding shortest path distances in weighted or unweighted graphs.

- *Medium:* Shortest Path in Binary Matrix (LeetCode) – *Use BFS in a grid (unweighted) to find the shortest path from top-left to bottom-right*.
- *Medium:* Network Delay Time (LeetCode) – Dijkstra's Shortest Path (GfG) – *Use Dijkstra's algorithm (with a min-heap) to find how long for all nodes to receive a signal*.
- *Hard:* Cheapest Flights Within K Stops (LeetCode) – *Use BFS (level by level for <=K stops) or Dijkstra for finding cheapest price within K hops*.

- *Hard:* Graph Coloring / M Coloring Problem (GfG) – *Assign colors to graph nodes so that no adjacent have same color (backtracking on graph)*.

- **Topological Sort (DAG order):** Linear ordering of vertices such that all directed edges go from earlier to later in the order.

- *Medium:* Course Schedule (LeetCode) – Detect Cycle in Directed Graph (GfG) – *Determine if you can complete all courses (detect cycle in prerequisite graph using DFS/BFS)*.
- *Medium:* Course Schedule II (LeetCode) – Topo Sort (GfG) – *Produce a possible order to take courses (topologically sort the DAG of prerequisites)*.

- *Hard:* Alien Dictionary (LeetCode) – Alien Dictionary (GfG) – *Derive alphabet order from sorted alien words (build graph of letter precedence, then topologically sort)*.

- **Union-Find (Disjoint Set Union):** Union-Find data structure to manage connected components in graphs [10] [11].

- *Medium:* Number of Provinces (LeetCode) – Friends Circle (GfG) – *Use Union-Find or DFS to count connected components in an undirected graph (e.g., connectivity in an adjacency matrix of cities).*
- *Medium:* Graph Valid Tree (LeetCode) – *Use Union-Find to check if a graph is a single connected component with no cycles.*
- *Medium:* Redundant Connection (LeetCode) – *Use Union-Find to find an extra edge creating a cycle in an undirected graph.*
- *Hard:* Accounts Merge (LeetCode) – *Use Union-Find to group emails/accounts belonging to the same user.*
- *Hard:* Optimize Water Distribution (GfG) – *Use Union-Find (with Kruskal's MST) to minimize cost to supply water in a network.*

(Also practice Minimum Spanning Tree algorithms (Kruskal using Union-Find, Prim's) and Bipartite Check (using DFS/BFS or Union-Find).)

Tries (Prefix Trees)

- **Prefix Tree Construction:** Storing strings in a trie for fast prefix-based operations.
- *Medium:* Implement Trie (LeetCode) – Trie Insert and Search (GfG) – *Design a Trie with insert and search (prefix tree of characters).*
- *Medium:* Add and Search Word (LeetCode) – Word Dictionary (GfG) – *Modify Trie to handle* `.` *wildcard in searches.*
- *Medium:* Word Search II (LeetCode) – Word Boggle (GfG) – *Use a Trie to optimize backtracking searches for words in a grid.*
- *Hard:* Design Search Autocomplete (LeetCode) – *Use Trie with additional data (frequency of sentences) to auto-complete queries.*
- *Hard:* Word Squares (LeetCode) – *Use backtracking with Trie pruning to build word squares (each prefix must be valid).*

(Also know that tries can be used for bitwise problems (bitwise trie for max XOR pair), and for efficient prefix/suffix queries.)

Greedy Algorithms

- **Interval Scheduling & Selection:** Greedily selecting intervals or tasks based on some optimal choice (usually after sorting).
- *Easy:* Activity Selection (GfG) – *Select maximum non-overlapping intervals (greedy by finishing times).*
- *Medium:* Non-overlapping Intervals (LeetCode) – *Erase minimum intervals to avoid overlaps (equivalently select max non-overlapping set).*
- *Medium:* Meeting Rooms II (LeetCode) – Minimum Platforms (GfG) – *Find minimum number of conference rooms or platforms needed (greedy + sort by time, or use min-heap).*
- *Hard:* Insert Interval (LeetCode) – *Insert a new interval and merge if necessary (greedy merging scan).*

- *Hard:* Interval Scheduling with Profits (Weighted) / Job Scheduling (LeetCode) – *Select non-overlapping jobs with max profit (sort + binary search + DP/greedy hybrid).*

- **Greedy for Arrays/Strings:** Locally optimal choices for partitioning or modifying arrays/strings.

- *Medium:* Jump Game (LeetCode) – Jump Game (GfG) – *Greedily determine if you can reach the last index (track furthest reachable index)*.
- *Medium:* Gas Station (LeetCode) – Circular Tour (GfG) – *Greedy approach to find a start position in circular gas stations to complete the circuit*.
- *Hard:* Candy Distribution (LeetCode) – Candy Distribution (GfG) – *Greedy left-to-right and right-to-left passes to distribute candies fairly to children based on ratings*.
- *Hard:* Rearrange String k Distance Apart (LeetCode) – *Greedy + heap to rearrange string so that same chars are at least k apart*.

- *Hard:* Increasing Triplet Subsequence (LeetCode) – *Greedy tracking of two smallest values to find an increasing triplet (O(n) solution)*.

- **Greedy Algorithms in Graphs:**

- *Medium:* Minimum Spanning Tree (Kruskal/Prim) (GfG)[★] – *Use greedy approach to connect all nodes with minimum total edge weight (Kruskal's uses union-find, Prim's uses greedy edge selection)*.
- *Medium:* Dijkstra's Algorithm (Single Source Shortest Path) (GfG)[★] – *Greedily pick the nearest unvisited node to finalize shortest path distances (uses a priority queue)*.
- *Hard:* Huffman Coding (GfG) – *Greedy algorithm to build an optimal prefix-free encoding (uses min-heap to merge lowest frequency nodes)*.

(Greedy approaches often require a careful proof of correctness. Always consider counterexamples and why the greedy choice works.)

Union-Find / Disjoint Sets

(Covered under Graphs above – see Union-Find pattern in Graphs section for connectivity and cycle detection.)

(Also useful in problems like networking, Kruskal's MST, checking connectivity dynamically, and DSU-based solutions for puzzles like coupling or grouping problems.)

Bit Manipulation

- **Basic Bit Tricks:** Leveraging bit operations (AND, OR, XOR, shifts) for efficiency.
- *Easy:* Power of Two (LeetCode) – Power of 2 Check (GfG) – *Check if n is a power of 2 (n & (n-1) == 0)*.
- *Easy:* Hamming Weight / Count Set Bits (LeetCode) – Count Set Bits (GfG) – *Use bit tricks (n &= n-1 loop) to count 1s in binary representation*.
- *Easy:* Single Number (LeetCode) – Element Appearing Once (GfG) – *Use XOR to find unique element in array where others appear twice*.
- *Medium:* Single Number II (LeetCode) – *Bit manipulation to find unique number when others appear thrice (use bit count mod 3)*.
- *Medium:* Bitwise AND of Range (LeetCode) – Bitwise AND Range (GfG) – *Find common bit prefix of all numbers in range [m, n] by bit shifting*.

- *Hard:* Maximum XOR of Two Numbers (LeetCode) – Max XOR of Pair (GfG) – *Use a bitwise Trie to find two numbers with maximum XOR (greedy bit-by-bit).*
- *Hard:* Subset XOR Sum (GfG) – *Use recursion or DP to find XOR of all subset XOR totals (illustrates bit DP concepts).*

(Also understand bit masking for subsets representation (useful in DP or brute force optimizations) and bit operations for toggling and extracting specific bits.)

Math & Number Theory

- **Prime Numbers & Factors:**
- *Easy:* Count Primes (LeetCode) – Sieve of Eratosthenes (GfG) – *Count number of primes less than n (using the Sieve for efficiency).*
- *Easy:* Greatest Common Divisor (GCD) (LeetCode) – Euclidean GCD (GfG) – *Use Euclidean algorithm to find GCD (and LCM via GCD).*
- *Medium:* Trailing Zeroes in Factorial (LeetCode) – Count Trailing Zeros (GfG) – *Count number of 5 factors in n! (integer division method).*
- *Medium:* Excel Sheet Column Title (LeetCode) – Excel Column Title (GfG) – *Convert a number to base-26 alphabet system (repeated division technique).*
- *Medium:* Pow(x, n) (LeetCode) – Modular Exponentiation (GfG) – *Fast exponentiation (binary exponentiation) to compute power efficiently.*
- *Hard:* Permutation Sequence (LeetCode) – *Math solution using factorial number system to get the k-th permutation of [1..n].*
- *Hard:* Convex Hull (GfG) – *Graham scan or Jarvis march algorithm to find convex hull of points (computational geometry).*
- *Hard:* Random Pick with Weight (LeetCode) – *Use prefix sum and binary search to pick index with probability proportional to weight (math & binary search).*

(Mathematical reasoning is key for combinatorics (nCr calculations), probability, and understanding algorithm complexity. Be comfortable with modular arithmetic and basic combinatorial formulas.)

Practice and Mastery: Mastering these patterns and problem types will enable you to tackle most interview coding challenges [1]. Focus on recognizing patterns in problems – for example, sliding window for subarray problems, two pointers for sorted pairs/triplets, DFS/BFS for traversal, or DP for optimization problems. Use LeetCode and GeeksforGeeks links to practice each pattern. By covering easy through hard problems in each category, you'll build confidence and the ability to apply the right pattern for any interview problem. Good luck with your interview preparation!

---

[1] #dsapatterns #dsa #collab #preplacedcollab #tech #interview | Ayushi Sharma | 95 comments
https://www.linkedin.com/posts/ayushi-sharma-8a285a185_dsapatterns-dsa-collab-activity-7278732294561918976-b5Dp

[2] [9] Two Sum - Pair with given Sum - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/check-if-pair-with-given-sum-exists-in-array/

[3] 3 Sum - Find All Triplets with Zero Sum - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/find-triplets-array-whose-sum-equal-zero/

4 5 Trapping Rain Water Problem - Tutorial with Illustrations - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/trapping-rain-water/

6 Find maximum average subarray of k length - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/find-maximum-average-subarray-of-k-length/

7 8 K'th Smallest Element in Unsorted Array - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/kth-smallest-largest-element-in-unsorted-array/

10 11 Introduction to Disjoint Set (Union-Find Algorithm) - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/introduction-to-disjoint-set-data-structure-or-union-find-algorithm/