Top 50 Data Structures and Algorithms Problems for Senior C++ Interviews

October 24, 2025

Introduction

This document compiles 50 frequently asked Data Structures and Algorithms (DSA) problems for senior-level C++ interviews at companies like Amazon, Google, and Microsoft. Problems are organized by topic, focusing on medium to hard difficulties, with sample inputs and outputs for clarity. Each entry includes the LeetCode ID (LC#), difficulty, description, why its frequent, and example input/output. For C++ interviews, prioritize efficient implementations using STL (e.g., `std::unordered_map`, `std::priority_queue`). Revise by solving 23 problems daily, focusing on patterns like sliding windows or DFS.

1 Arrays & Strings

1.1 Two Sum (LC1, Easy)

Description: Given an array of integers and a target, return indices of two numbers that add up to the target.
Why Frequent: Tests hashing for O(1) lookups; common warmup for sum variations.
Sample Input/Output:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: nums[0] + nums[1] = 2 + 7 = 9
```

1.2 3Sum (LC15, Medium)

Description: Find all unique triplets in an array that sum to zero.
Why Frequent: Uses sorting and two pointers; common for multi-sum problems.
Sample Input/Output:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation: Triplets summing to 0, no duplicates.
```

1.3 Container With Most Water (LC11, Medium)

Description: Given heights of lines, find two lines forming a container with max area.
Why Frequent: Classic two-pointer optimization; tests greedy choices.
Sample Input/Output:

```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
```

```
Explanation: Max area between height[1] and height[8].
```

1.4 Longest Substring Without Repeating Characters (LC3, Medium)

Description: Find the length of the longest substring without repeating characters.
Why Frequent: Sliding window with hash set; common for string problems.
Sample Input/Output:

```
Input: s = "abcabcbb"
Output: 3
Explanation: Longest substring is "abc".
```

1.5 Group Anagrams (LC49, Medium)

Description: Group a list of strings by their anagrams.
Why Frequent: Hashing with sorted keys; tests map usage in C++.
Sample Input/Output:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

2 Linked Lists

2.1 Reverse Linked List (LC206, Easy)

Description: Reverse a singly linked list.
Why Frequent: Basic pointer manipulation; extensible to groups.
Sample Input/Output:

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

2.2 Linked List Cycle (LC141, Easy)

Description: Detect if a cycle exists in a linked list.
Why Frequent: Floyds tortoise-hare algorithm; tests cycle detection.
Sample Input/Output:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: Cycle at node 2 (index 1).
```

2.3 Merge Two Sorted Lists (LC21, Easy)

Description: Merge two sorted linked lists into one.
Why Frequent: Recursive or iterative merging; common for k-merge.
Sample Input/Output:

```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

2.4 Remove Nth Node From End of List (LC19, Medium)

Description: Remove the nth node from the end of a linked list.
Why Frequent: Two-pass or one-pass with dummies; tests edge cases.
Sample Input/Output:

```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

2.5 Reorder List (LC143, Medium)

Description: Reorder list to L0 → Ln → L1 → Ln-1 → ...
Why Frequent: Combines reverse and merge; complex manipulations.
Sample Input/Output:

```
Input: head = [1,2,3,4]
Output: [1,4,2,3]
```

3 Stacks & Queues

3.1 Valid Parentheses (LC20, Easy)

Description: Check if a string of parentheses is valid.
Why Frequent: Basic stack usage; often a starter.
Sample Input/Output:

```
Input: s = "()"
Output: true
```

3.2 Next Greater Element I (LC496, Easy)

Description: Find the next greater element for each in an array.
Why Frequent: Monotonic stack; extends to daily temperatures.
Sample Input/Output:

```
Input: nums1 = [4,1,2], nums2 = [1,3,4,2]
Output: [-1,3,-1]
```

3.3 Sliding Window Maximum (LC239, Hard)

Description: Find max in each k-sized window of an array.
Why Frequent: Deque for O(n); tests optimizations.
Sample Input/Output:

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
```

3.4 Evaluate Reverse Polish Notation (LC150, Medium)

Description: Evaluate a postfix expression.
Why Frequent: Stack for operators; infix/postfix conversions.
Sample Input/Output:

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: (2 + 1) * 3 = 9
```

3.5 Implement Queue using Stacks (LC232, Easy)

Description: Implement a queue using two stacks.
Why Frequent: Tests stack/queue differences; design variant.
Sample Input/Output:

```
Input: push(1), push(2), peek(), pop(), empty()
Output: 1, 1, false
```

4 Hashing / Maps / Sets

4.1 Contains Duplicate (LC217, Easy)

Description: Check if an array has duplicates.
Why Frequent: Simple set usage; warmup for unique elements.
Sample Input/Output:

```
Input: nums = [1,2,3,1]
Output: true
```

4.2 Subarray Sum Equals K (LC560, Medium)

Description: Find number of subarrays summing to k.
Why Frequent: Prefix sum + hashmap; cumulative tricks.
Sample Input/Output:

```
Input: nums = [1,1,1], k = 2
Output: 2
Explanation: Subarrays [1,1] at indices 0-1, 1-2.
```

4.3 Longest Consecutive Sequence (LC128, Medium)

Description: Find longest sequence of consecutive numbers in unsorted array.
Why Frequent: Set for O(n) lookup; boundary checks.
Sample Input/Output:

```
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: Sequence [1,2,3,4].
```

4.4 Top K Frequent Elements (LC347, Medium)

Description: Find top k frequent elements in an array.
Why Frequent: Map + priority queue; multi-data structure.
Sample Input/Output:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

4.5 LRU Cache (LC146, Medium)

Description: Design LRU cache with get/put in O(1).
Why Frequent: List + map; system design for seniors.
Sample Input/Output:

```
Input: ["LRUCache","put","put","get","put","get"]
       [[2],[1,1],[2,2],[1],[3,3],[2]]
Output: [null,null,null,1,null,-1]
```

5 Trees & Binary Search Trees

5.1 Invert Binary Tree (LC226, Easy)

Description: Flip a binary tree.
Why Frequent: Recursive traversal; simple but extensible.
Sample Input/Output:

```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

5.2 Validate Binary Search Tree (LC98, Medium)

Description: Check if a binary tree is a valid BST.
Why Frequent: Inorder traversal; BST properties.
Sample Input/Output:

```
Input: root = [2,1,3]
Output: true
```

5.3 Binary Tree Maximum Path Sum (LC124, Hard)

Description: Find max path sum in a binary tree.
Why Frequent: Recursion with global max; hard for seniors.
Sample Input/Output:

```
Input: root = [1,2,3]
Output: 6
Explanation: Path 2->1->3.
```

5.4 Lowest Common Ancestor of a Binary Tree (LC236, Medium)

Description: Find LCA of two nodes.
Why Frequent: Recursive search; common in Microsoft.
Sample Input/Output:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
```

5.5 Kth Smallest Element in a BST (LC230, Medium)

Description: Find kth smallest in BST.
Why Frequent: Inorder with count; BST properties.
Sample Input/Output:

```
Input: root = [3,1,4,null,2], k = 1
Output: 1
```

6 Graphs

6.1 Number of Islands (LC200, Medium)

Description: Count islands in a grid (1s connected).
Why Frequent: DFS/BFS flood fill; Google favorite.
Sample Input/Output:

```
Input: grid = [["1","1","0"],["1","1","0"],["0","0","1"]]
Output: 2
```

6.2 Course Schedule (LC207, Medium)

Description: Detect if courses can be finished with prerequisites.
Why Frequent: Topo sort with cycle detect; Amazon for DAGs.
Sample Input/Output:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
```

6.3 Clone Graph (LC133, Medium)

Description: Deep copy an undirected graph.
Why Frequent: BFS/DFS with map; visited handling.
Sample Input/Output:

```
Input: adjList = [[2,4],[1,3],[2,4],[1,3]]
Output: [[2,4],[1,3],[2,4],[1,3]]
```

6.4 Pacific Atlantic Water Flow (LC417, Medium)

Description: Find cells where water flows to both oceans.
Why Frequent: Multi-source BFS; graph traversals.
Sample Input/Output:

```
Input: heights = [[1,2,2],[3,8,2],[5,3,5]]
Output: [[0,2],[1,0],[1,2],[2,0],[2,2]]
```

6.5 Word Search (LC79, Medium)

Description: Find if a word exists in a grid.
Why Frequent: Backtracking on graph; common variant.
Sample Input/Output:

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],
    word = "ABCCED"
Output: true
```

7 Heaps / Priority Queue

7.1 Kth Largest Element in an Array (LC215, Medium)

Description: Find kth largest using heap.
Why Frequent: Min-heap of size k; efficient in C++.
Sample Input/Output:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

7.2 Find Median from Data Stream (LC295, Hard)

Description: Add numbers and find median dynamically.
Why Frequent: Two heaps; senior design.
Sample Input/Output:

```
Input: ["MedianFinder","addNum","addNum","findMedian"]
       [[],[1],[2],[]]
Output: [null,null,null,1.5]
```

7.3 Merge K Sorted Lists (LC23, Hard)

Description: Merge k lists using heap.
Why Frequent: Priority queue for mins; scales to arrays.
Sample Input/Output:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
```

7.4 Task Scheduler (LC621, Medium)

Description: Schedule tasks with cooldown.
Why Frequent: Heap for frequencies; practical.
Sample Input/Output:

```
Input: tasks = ["A","A","A","B","B","B"], n = 2
Output: 8
Explanation: A->B->idle->A->B->idle->A->B
```

7.5 Top K Frequent Elements (LC347, Medium)

Description: As above, using heap.
Sample Input/Output:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

8 Recursion & Backtracking

8.1 Combination Sum (LC39, Medium)

Description: Find combinations summing to target.
Why Frequent: Backtracking with duplicates.
Sample Input/Output:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
```

8.2 Subsets (LC78, Medium)

Description: Generate all subsets.
Why Frequent: Recursive inclusion; powerset problems.
Sample Input/Output:

```
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

8.3 Permutations (LC46, Medium)

Description: Generate all permutations.
Why Frequent: Backtracking with swaps.
Sample Input/Output:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

8.4 Word Break (LC139, Medium)

Description: Break string into dictionary words.
Why Frequent: Recursion with memo; leads to DP.
Sample Input/Output:

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
```

8.5 N-Queens (LC51, Hard)

Description: Place queens without attacks.
Why Frequent: Classic backtracking; optimizations.
Sample Input/Output:

```
Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
```

9 Dynamic Programming

9.1 Climbing Stairs (LC70, Easy)

Description: Ways to climb n stairs (1/2 steps).
Why Frequent: Basic DP fib-like; warmup.
Sample Input/Output:

```
Input: n = 3
Output: 3
Explanation: [1,1,1], [1,2], [2,1]
```

9.2 Coin Change (LC322, Medium)

Description: Min coins to make amount.
Why Frequent: Unbounded knapsack.
Sample Input/Output:

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 5+5+1
```

9.3 Longest Increasing Subsequence (LC300, Medium)

Description: Find LIS length.
Why Frequent: DP or binary search; senior variant.
Sample Input/Output:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: [2,3,7,101]
```

9.4 Unique Paths (LC62, Medium)

Description: Paths in grid from top-left to bottom-right.
Why Frequent: Grid DP; obstacles variant.
Sample Input/Output:

```
Input: m = 3, n = 2
Output: 3
Explanation: 3 paths to bottom-right.
```

9.5 House Robber (LC198, Medium)

Description: Max rob without adjacent houses.
Why Frequent: 1D DP; extends to tree/circle.
Sample Input/Output:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (2) + 3 (9) + 5 (1).
```

10 Searching & Sorting

10.1 Search in Rotated Sorted Array (LC33, Medium)

Description: Binary search in rotated array.
Why Frequent: Modified binary search.
Sample Input/Output:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

10.2 Find Minimum in Rotated Sorted Array (LC153, Medium)

Description: Find min in rotated sorted array.
Why Frequent: Binary search pivot.
Sample Input/Output:

```
Input: nums = [3,4,5,1,2]
Output: 1
```

10.3 Best Time to Buy and Sell Stock (LC121, Easy)

Description: Max profit from one buy/sell.
Why Frequent: One-pass; variants for transactions.
Sample Input/Output:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy at 1, sell at 6.
```

10.4 Merge Sort (Implement)

Description: Implement merge sort.
Why Frequent: Stable sort; asked to code in C++.
Sample Input/Output:

```
Input: nums = [5,2,3,1]
Output: [1,2,3,5]
```

10.5 Quickselect for Kth Element (LC215 variant)

Description: Find kth smallest using partition.
Why Frequent: Average O(n); sorting alternative.
Sample Input/Output:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 2
```

11 Bonus Senior-Level Topics

11.1 LFU Cache (LC460, Hard)

Description: Design LFU cache with frequency + recency.
Why Frequent: Complex data structure; system design.
Sample Input/Output:

```
Input: ["LFUCache","put","put","get","put","get"]
       [[2],[1,1],[2,2],[1],[3,3],[2]]
Output: [null,null,null,1,null,-1]
```

11.2 Minimum Window Substring (LC76, Hard)

Description: Find smallest substring containing all chars.
Why Frequent: Sliding window + hash; senior string problem.
Sample Input/Output:

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
```

11.3 Binary Tree Serialize/Deserialize (LC297, Hard)

Description: Tree to string and back.
Why Frequent: String parsing; tree handling.
Sample Input/Output:

```
Input: root = [1,2,3,null,null,4,5]
Output: [1,2,3,null,null,4,5]
```

11.4 Trapping Rain Water (LC42, Hard)

Description: Compute water trapped by bars.
Why Frequent: Two pointers or stack; optimization.
Sample Input/Output:

```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

11.5 Word Search II (LC212, Hard)

Description: Find all dictionary words in a grid.
Why Frequent: Trie + backtracking; multi-word search.
Sample Input/Output:

```
Input: board = [["o","a","a","n"],["e","t","a","e"]], words = ["oath","
   pea"]
Output: ["oath"]
```

12 Revision Strategy

- Pick one topic per day and solve 23 problems.
- Focus on patterns (e.g., sliding window, DFS) rather than individual problems.
- Maintain a cheat sheet of key algorithms (e.g., Kadanes, Floyds cycle detection).
- Practice medium to hard problems on LeetCode or GeeksforGeeks, targeting FAANG-style questions.
- Implement solutions in C++ using STL for efficiency (e.g., `std::unordered_map`, `std::priority_queue`)