100 Data Structures & Algorithms Practice Questions

Arrays & Strings (Questions 1-15)

1. Two Sum

Find two numbers in an array that add up to a target sum.

```python
def two_sum(nums, target):
    """
    Given an array of integers and a target, return indices of two numbers that add up to target.
    """
    pass
```

Test Cases:

- Input: `two_sum([2, 7, 11, 15], 9)` → Output: `[0, 1]`
- Input: `two_sum([3, 2, 4], 6)` → Output: `[1, 2]`
- Input: `two_sum([3, 3], 6)` → Output: `[0, 1]`

Complexity: O(n) time, O(n) space

2. Best Time to Buy and Sell Stock

Find maximum profit from one buy and one sell.

```python
def max_profit(prices):
    """
    Given array of stock prices, find max profit from one transaction.
    """
    pass
```

Test Cases:

- Input: `max_profit([7, 1, 5, 3, 6, 4])` → Output: `5` (buy at 1, sell at 6)
- Input: `max_profit([7, 6, 4, 3, 1])` → Output: `0` (no profit)
- Input: `max_profit([1, 2, 3, 4, 5])` → Output: `4` (buy at 1, sell at 5)

Complexity: O(n) time, O(1) space

3. Contains Duplicate

Check if array contains any duplicates.

```python
def contains_duplicate(nums):
    """
    Return True if any value appears at least twice.
    """
    pass
```

Test Cases:

- Input: `contains_duplicate([1, 2, 3, 1])` → Output: `True`
- Input: `contains_duplicate([1, 2, 3, 4])` → Output: `False`
- Input: `contains_duplicate([1, 1, 1, 3, 3, 4, 3, 2, 4, 2])` → Output: `True`

Complexity: O(n) time, O(n) space

4. Product of Array Except Self

Return array where each element is product of all other elements.

```python
def product_except_self(nums):
    """
    Without using division, return array where output[i] = product of all elements except nums[i].
    """
    pass
```

Test Cases:

- Input: `product_except_self([1, 2, 3, 4])` → Output: `[24, 12, 8, 6]`
- Input: `product_except_self([−1, 1, 0, −3, 3])` → Output: `[0, 0, 9, 0, 0]`

Complexity: O(n) time, O(1) space (output array doesn't count)

5. Maximum Subarray (Kadane's Algorithm)

Find contiguous subarray with largest sum.

```python
```

```python
def max_subarray(nums):
    """
    Find the contiguous subarray with the largest sum.
    """
    pass
```

Test Cases:

- Input: `max_subarray([−2, 1, −3, 4, −1, 2, 1, −5, 4])` → Output: `6` ([4,−1,2,1])
- Input: `max_subarray([1])` → Output: `1`
- Input: `max_subarray([5, 4, −1, 7, 8])` → Output: `23`

Complexity: O(n) time, O(1) space

6. Merge Sorted Arrays

Merge two sorted arrays into one sorted array.

```python
def merge_sorted_arrays(nums1, nums2):
    """
    Merge two sorted arrays.
    """
    pass
```

Test Cases:

- Input: `merge_sorted_arrays([1, 3, 5], [2, 4, 6])` → Output: `[1, 2, 3, 4, 5, 6]`
- Input: `merge_sorted_arrays([1, 2, 3], [])` → Output: `[1, 2, 3]`

Complexity: O(n + m) time, O(n + m) space

7. Rotate Array

Rotate array to the right by k steps.

```python
def rotate_array(nums, k):
    """
    Rotate array in-place.
    """
    pass
```

Test Cases:

- Input: `nums = [1, 2, 3, 4, 5, 6, 7], k = 3` → Output: `[5, 6, 7, 1, 2, 3, 4]`
- Input: `nums = [−1, −100, 3, 99], k = 2` → Output: `[3, 99, −1, −100]`

Complexity: O(n) time, O(1) space

8. Find Missing Number

Find the missing number in array containing n distinct numbers from 0 to n.

```python
def missing_number(nums):
    """
    Array contains n numbers from range [0, n]. Find the missing one.
    """
    pass
```

Test Cases:

- Input: `missing_number([3, 0, 1])` → Output: `2`
- Input: `missing_number([0, 1])` → Output: `2`
- Input: `missing_number([9, 6, 4, 2, 3, 5, 7, 0, 1])` → Output: `8`

Complexity: O(n) time, O(1) space

9. Valid Anagram

Check if two strings are anagrams.

```python
def is_anagram(s, t):
    """
    Return True if t is an anagram of s.
    """
    pass
```

Test Cases:

- Input: `is_anagram("anagram", "nagaram")` → Output: `True`
- Input: `is_anagram("rat", "car")` → Output: `False`

Complexity: O(n) time, O(1) space

10. Longest Substring Without Repeating Characters

Find length of longest substring without repeating characters.

```python
def length_of_longest_substring(s):
    """
    Find length of longest substring without repeating characters.
    """
    pass
```

Test Cases:

- Input: `length_of_longest_substring("abcabcbb")` → Output: `3` ("abc")
- Input: `length_of_longest_substring("bbbbb")` → Output: `1` ("b")
- Input: `length_of_longest_substring("pwwkew")` → Output: `3` ("wke")

Complexity: O(n) time, O(min(m, n)) space

11. 3Sum

Find all unique triplets that sum to zero.

```python
def three_sum(nums):
    """
    Find all unique triplets [nums[i], nums[j], nums[k]] such that i != j != k and nums[i] + nums[j] + nums[k] == 0.
    """
    pass
```

Test Cases:

- Input: `three_sum([−1, 0, 1, 2, −1, −4])` → Output: `[[-1, -1, 2], [-1, 0, 1]]`
- Input: `three_sum([0, 1, 1])` → Output: `[]`
- Input: `three_sum([0, 0, 0])` → Output: `[[0, 0, 0]]`

Complexity: $O(n^2)$ time, O(1) space

12. Container With Most Water

Find two lines that together with x-axis form a container with most water.

```python
def max_area(height):
    """
    Given n non-negative integers representing heights, find max water container can store.
    """
    pass
```

Test Cases:

- Input: `max_area([1, 8, 6, 2, 5, 4, 8, 3, 7])` → Output: `49`
- Input: `max_area([1, 1])` → Output: `1`

Complexity: O(n) time, O(1) space

13. Trapping Rain Water

Calculate how much rainwater can be trapped.

```python
def trap_rain_water(height):
    """
    Given n non-negative integers representing elevation map, compute how much water it can trap.
    """
    pass
```

Test Cases:

- Input: `trap_rain_water([0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1])` → Output: `6`
- Input: `trap_rain_water([4, 2, 0, 3, 2, 5])` → Output: `9`

Complexity: O(n) time, O(1) space

14. Longest Palindromic Substring

Find the longest palindromic substring.

```python
def longest_palindrome(s):
    """
    Return the longest palindromic substring in s.
    """
    pass
```

Test Cases:

- Input: `longest_palindrome("babad")` → Output: `"bab"` or `"aba"`
- Input: `longest_palindrome("cbbd")` → Output: `"bb"`

Complexity: O(n²) time, O(1) space

15. Move Zeroes

Move all zeroes to the end while maintaining relative order.

```python
def move_zeroes(nums):
    """
    Move all 0's to the end in-place.
    """
    pass
```

Test Cases:

- Input: `nums = [0, 1, 0, 3, 12]` → Output: `[1, 3, 12, 0, 0]`
- Input: `nums = [0, 0, 1]` → Output: `[1, 0, 0]`

Complexity: O(n) time, O(1) space

Linked Lists (Questions 16-25)

16. Reverse Linked List

Reverse a singly linked list.

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverse_list(head):
    """
    Reverse a singly linked list.
    """
    pass
```

Test Cases:

- Input: `1 → 2 → 3 → 4 → 5` → Output: `5 → 4 → 3 → 2 → 1`
- Input: `1 → 2` → Output: `2 → 1`
- Input: `[]` → Output: `[]`

Complexity: O(n) time, O(1) space

17. Merge Two Sorted Lists

Merge two sorted linked lists.

```python
def merge_two_lists(l1, l2):
    """
    Merge two sorted linked lists and return as a new sorted list.
    """
    pass
```

Test Cases:

- Input: `l1 = [1, 2, 4], l2 = [1, 3, 4]` → Output: `[1, 1, 2, 3, 4, 4]`
- Input: `l1 = [], l2 = []` → Output: `[]`

Complexity: O(n + m) time, O(1) space

18. Linked List Cycle

Detect if linked list has a cycle.

```python
def has_cycle(head):
    """
    Return True if there is a cycle in the linked list.
    """
    pass
```

Test Cases:

- Input: `3 → 2 → 0 → −4 → (back to 2)` → Output: `True`
- Input: `1 → 2` → Output: `False`

Complexity: O(n) time, O(1) space (Floyd's cycle detection)

19. Remove Nth Node From End

Remove the nth node from the end of list.

```python
def remove_nth_from_end(head, n):
    """
    Remove the nth node from the end of the list.
    """
    pass
```

Test Cases:

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

Complexity: O(n) time, O(1) space

20. Middle of Linked List

Find the middle node of linked list.

```python
def middle_node(head):
    """
    Return the middle node. If two middle nodes, return the second.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3, 4, 5]` → Output: `Node with value 3`
- Input: `[1, 2, 3, 4, 5, 6]` → Output: `Node with value 4`

Complexity: O(n) time, O(1) space

21. Palindrome Linked List

Check if linked list is a palindrome.

```python
```

```
def is_palindrome_list(head):
    """
    Return True if linked list is a palindrome.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 2, 1]` → Output: `True`
- Input: `[1, 2]` → Output: `False`

Complexity: O(n) time, O(1) space

22. Intersection of Two Linked Lists

Find the node where two lists intersect.

```
python

def get_intersection_node(headA, headB):
    """
    Find the node at which two singly linked lists intersect.
    """
    pass
```

Test Cases:

- Lists intersect at node with value 8
- Lists don't intersect → Output: `None`

Complexity: O(m + n) time, O(1) space

23. Add Two Numbers (Linked Lists)

Add two numbers represented by linked lists.

```
python

def add_two_numbers(l1, l2):
    """
    Numbers stored in reverse order. Each node contains a single digit.
    """
    pass
```

Test Cases:

- Input: `l1 = [2, 4, 3], l2 = [5, 6, 4]` → Output: `[7, 0, 8]` (342 + 465 = 807)

Complexity: O(max(m, n)) time, O(max(m, n)) space

24. Remove Duplicates from Sorted List

Remove duplicates from a sorted linked list.

```python
def delete_duplicates(head):
    """
    Delete all duplicates such that each element appears only once.
    """
    pass
```

Test Cases:

- Input: `[1, 1, 2]` → Output: `[1, 2]`
- Input: `[1, 1, 2, 3, 3]` → Output: `[1, 2, 3]`

Complexity: O(n) time, O(1) space

25. Reorder List

Reorder list: L0 → Ln → L1 → Ln-1 → L2 → Ln-2 → ...

```python
def reorder_list(head):
    """
    Reorder the list in-place.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3, 4]` → Output: `[1, 4, 2, 3]`
- Input: `[1, 2, 3, 4, 5]` → Output: `[1, 5, 2, 4, 3]`

Complexity: O(n) time, O(1) space

Stacks & Queues (Questions 26-32)

26. Valid Parentheses

Check if string of brackets is valid.

```python
def is_valid_parentheses(s):
    """
    Check if brackets are properly closed and nested.
    """
    pass
```

Test Cases:

- Input: `"()"` → Output: `True`
- Input: `"()[]{}"` → Output: `True`
- Input: `"(]"` → Output: `False`
- Input: `"([)]"` → Output: `False`

Complexity: O(n) time, O(n) space

27. Min Stack

Design a stack that supports push, pop, top, and retrieving minimum in O(1).

```python
class MinStack:
    def __init__(self):
        pass

    def push(self, val):
        pass

    def pop(self):
        pass

    def top(self):
        pass

    def get_min(self):
        pass
```

Test Cases:

- Operations: push(-2), push(0), push(-3), getMin() → -3, pop(), top() → 0, getMin() → -2

Complexity: O(1) for all operations

28. Evaluate Reverse Polish Notation

Evaluate arithmetic expression in RPN.

```python
def eval_rpn(tokens):
    """
    Evaluate the value of arithmetic expression in Reverse Polish Notation.
    """
    pass
```

Test Cases:

- Input: `["2", "1", "+", "3", "*"]` → Output: `9` ((2 + 1) * 3)
- Input: `["4", "13", "5", "/", "+"]` → Output: `6` (4 + (13 / 5))

Complexity: O(n) time, O(n) space

29. Daily Temperatures

Find how many days until warmer temperature.

```python
def daily_temperatures(temperatures):
    """
    Return array where answer[i] is number of days until warmer temperature.
    """
    pass
```

Test Cases:

- Input: `[73, 74, 75, 71, 69, 72, 76, 73]` → Output: `[1, 1, 4, 2, 1, 1, 0, 0]`

Complexity: O(n) time, O(n) space

30. Implement Queue Using Stacks

Implement FIFO queue using only stacks.

```python
class MyQueue:
    def __init__(self):
        pass

    def push(self, x):
        pass

    def pop(self):
        pass

    def peek(self):
        pass

    def empty(self):
        pass
```

Complexity: Amortized O(1) for all operations

31. Implement Stack Using Queues

Implement LIFO stack using only queues.

```python
class MyStack:
    def __init__(self):
        pass

    def push(self, x):
        pass

    def pop(self):
        pass

    def top(self):
        pass

    def empty(self):
        pass
```

32. Largest Rectangle in Histogram

Find largest rectangle in histogram.

```python
def largest_rectangle_area(heights):
    """
    Find the area of largest rectangle in the histogram.
    """
    pass
```

Test Cases:

- Input: `[2, 1, 5, 6, 2, 3]` → Output: `10`
- Input: `[2, 4]` → Output: `4`

Complexity: O(n) time, O(n) space

Trees (Questions 33-47)

33. Maximum Depth of Binary Tree

Find the maximum depth of a binary tree.

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def max_depth(root):
    """
    Return the maximum depth of the tree.
    """
    pass
```

Test Cases:

- Input: `[3, 9, 20, null, null, 15, 7]` → Output: `3`

Complexity: O(n) time, O(h) space

34. Invert Binary Tree

Invert a binary tree.

```python
def invert_tree(root):
    """
    Invert the binary tree (swap left and right children).
    """
    pass
```

Test Cases:

- Input: `[4, 2, 7, 1, 3, 6, 9]` → Output: `[4, 7, 2, 9, 6, 3, 1]`

Complexity: O(n) time, O(h) space

35. Same Tree

Check if two trees are identical.

```python
def is_same_tree(p, q):
    """
    Return True if two trees are structurally identical with same values.
    """
    pass
```

Test Cases:

- Input: `p = [1, 2, 3], q = [1, 2, 3]` → Output: `True`
- Input: `p = [1, 2], q = [1, null, 2]` → Output: `False`

Complexity: O(n) time, O(h) space

36. Symmetric Tree

Check if tree is symmetric around its center.

```python
def is_symmetric(root):
    """
    Check if tree is a mirror of itself.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 2, 3, 4, 4, 3]` → Output: `True`
- Input: `[1, 2, 2, null, 3, null, 3]` → Output: `False`

Complexity: O(n) time, O(h) space

37. Path Sum

Check if tree has root-to-leaf path with given sum.

```python
def has_path_sum(root, target_sum):
    """
    Return True if tree has a root-to-leaf path such that sum equals targetSum.
    """
    pass
```

Test Cases:

- Input: `root = [5, 4, 8, 11, null, 13, 4, 7, 2, null, null, null, 1], targetSum = 22` → Output: `True`

Complexity: O(n) time, O(h) space

38. Binary Tree Level Order Traversal

Return level order traversal of tree nodes.

```python
def level_order(root):
    """
    Return list of lists, where each inner list contains values at that level.
    """
    pass
```

Test Cases:

- Input: `[3, 9, 20, null, null, 15, 7]` → Output: `[[3], [9, 20], [15, 7]]`

Complexity: O(n) time, O(n) space

39. Binary Tree Zigzag Level Order Traversal

Return zigzag level order traversal.

```python
def zigzag_level_order(root):
    """
    Return zigzag level order: left to right, then right to left alternating.
    """
    pass
```

Test Cases:

- Input: `[3, 9, 20, null, null, 15, 7]` → Output: `[[3], [20, 9], [15, 7]]`

Complexity: O(n) time, O(n) space

40. Validate Binary Search Tree

Check if tree is a valid BST.

```python
def is_valid_bst(root):
    """
    Return True if tree is a valid binary search tree.
    """
    pass
```

Test Cases:

- Input: `[2, 1, 3]` → Output: `True`
- Input: `[5, 1, 4, null, null, 3, 6]` → Output: `False`

Complexity: O(n) time, O(h) space

41. Lowest Common Ancestor of BST

Find LCA of two nodes in BST.

```python
def lowest_common_ancestor(root, p, q):
    """
    Find the lowest common ancestor of two nodes in BST.
    """
    pass
```

Test Cases:

- Input: `root = [6, 2, 8, 0, 4, 7, 9, null, null, 3, 5], p = 2, q = 8` → Output: `6`
- Input: `root = [6, 2, 8, 0, 4, 7, 9, null, null, 3, 5], p = 2, q = 4` → Output: `2`

Complexity: O(h) time, O(1) space

42. Binary Tree Right Side View

Return values of nodes visible from right side.

```python
def right_side_view(root):
    """
    Return the values of nodes you can see from right side, top to bottom.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3, null, 5, null, 4]` → Output: `[1, 3, 4]`

Complexity: O(n) time, O(h) space

43. Count Complete Tree Nodes

Count nodes in a complete binary tree.

```python
def count_nodes(root):
    """
    Count the number of nodes in a complete binary tree.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3, 4, 5, 6]` → Output: `6`

Complexity: $O(\log^2 n)$ time

44. Kth Smallest Element in BST

Find kth smallest element in BST.

```python
def kth_smallest(root, k):
    """
    Find the kth smallest element in BST (1-indexed).
    """
    pass
```

Test Cases:

- Input: `root = [3, 1, 4, null, 2], k = 1` → Output: `1`
- Input: `root = [5, 3, 6, 2, 4, null, null, 1], k = 3` → Output: `3`

Complexity: O(n) time, O(h) space

45. Serialize and Deserialize Binary Tree

Design algorithm to serialize and deserialize a binary tree.

```python
class Codec:
    def serialize(self, root):
        """Encodes a tree to a single string."""
        pass

    def deserialize(self, data):
        """Decodes your encoded data to tree."""
        pass
```

Complexity: O(n) time, O(n) space

46. Binary Tree Maximum Path Sum

Find maximum path sum in binary tree.

```python
def max_path_sum(root):
    """
    A path is any sequence of nodes connected by edges. Find max sum.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3]` → Output: `6`
- Input: `[-10, 9, 20, null, null, 15, 7]` → Output: `42`

Complexity: O(n) time, O(h) space

47. Construct Binary Tree from Preorder and Inorder

Build tree from preorder and inorder traversals.

```python
def build_tree(preorder, inorder):
    """
    Construct binary tree from preorder and inorder traversal.
    """
    pass
```

Test Cases:

- Input: `preorder = [3, 9, 20, 15, 7], inorder = [9, 3, 15, 20, 7]` → Output: `[3, 9, 20, null, null, 15, 7]`

Complexity: O(n) time, O(n) space

Graphs (Questions 48-57)

48. Number of Islands

Count number of islands in 2D grid.

```python
def num_islands(grid):
    """
    Count the number of islands ('1's surrounded by '0's).
    """
    pass
```

Test Cases:

- Input:

```
[["1","1","0","0","0"],
 ["1","1","0","0","0"],
 ["0","0","1","0","0"],
 ["0","0","0","1","1"]]
```

→ Output: `3`

Complexity: O(m × n) time, O(m × n) space

49. Clone Graph

Deep copy of an undirected graph.

```python
class Node:
    def __init__(self, val=0, neighbors=None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []

def clone_graph(node):
    """
    Return a deep copy of the graph.
    """
    pass
```

Complexity: O(n + e) time, O(n) space

50. Course Schedule (Cycle Detection)

Determine if you can finish all courses.

```python
def can_finish(num_courses, prerequisites):
    """
    Return True if you can finish all courses.
    prerequisites[i] = [ai, bi] means you must take bi before ai.
    """
    pass
```

Test Cases:

- Input: `numCourses = 2, prerequisites = [[1, 0]]` → Output: `True`
- Input: `numCourses = 2, prerequisites = [[1, 0], [0, 1]]` → Output: `False`

Complexity: O(V + E) time, O(V + E) space

51. Pacific Atlantic Water Flow

Find cells where water can flow to both oceans.

```python
def pacific_atlantic(heights):
    """
    Find all cells where water can flow to both Pacific and Atlantic oceans.
    """
    pass
```

Test Cases:

- Water flows from higher to equal or lower cells
- Return list of coordinates

Complexity: O(m × n) time, O(m × n) space

52. Word Ladder

Find shortest transformation sequence from beginWord to endWord.

```python
def ladder_length(begin_word, end_word, word_list):
    """
    Find length of shortest transformation sequence.
    Each transformed word must exist in wordList.
    """
    pass
```

Test Cases:

- Input: `beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]` → Output: `5` (hit → hot → dot → dog → cog)

Complexity: O(M^2 × N) where M is word length, N is wordList size

53. Graph Valid Tree

Check if graph is a valid tree.

```python
def valid_tree(n, edges):
    """
    Return True if edges form a valid tree with n nodes.
    """
    pass
```

Test Cases:

- Input: `n = 5, edges = [[0, 1], [0, 2], [0, 3], [1, 4]]` → Output: `True`
- Input: `n = 5, edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]]` → Output: `False`

Complexity: O(V + E) time, O(V + E) space

54. Number of Connected Components

Count connected components in undirected graph.

```python
def count_components(n, edges):
    """
    Count the number of connected components.
    """
    pass
```

Test Cases:

- Input: `n = 5, edges = [[0, 1], [1, 2], [3, 4]]` → Output: `2`

Complexity: O(V + E) time, O(V) space

55. Dijkstra's Shortest Path

Find shortest path using Dijkstra's algorithm.

```python
```

```python
import heapq

def dijkstra(graph, start):
    """
    Find shortest distances from start to all other nodes.
    """
    pass
```

Test Cases:

- Return dict of shortest distances from start node

Complexity: O((V + E) log V) with min-heap

56. Topological Sort

Perform topological sort on directed acyclic graph.

```python
def topological_sort(num_nodes, edges):
    """
    Return topological ordering of nodes.
    """
    pass
```

Test Cases:

- Input: `num_nodes = 4, edges = [[1, 0], [2, 0], [3, 1], [3, 2]]` → Output: `[3, 2, 1, 0]` or any valid ordering

Complexity: O(V + E) time, O(V) space

57. Minimum Spanning Tree (Kruskal's)

Find minimum spanning tree using Kruskal's algorithm.

```python
def kruskal_mst(n, edges):
    """
    edges = [[u, v, weight], ...]
    Return minimum cost to connect all nodes.
    """
    pass
```

Complexity: O(E log E) time

Sorting & Searching (Questions 58-65)

58. Binary Search

Implement binary search.

```python
def binary_search(nums, target):
    """
    Search for target in sorted array. Return index or -1.
    """
    pass
```

Test Cases:

- Input: `binary_search([-1, 0, 3, 5, 9, 12], 9)` → Output: `4`
- Input: `binary_search([-1, 0, 3, 5, 9, 12], 2)` → Output: `-1`

Complexity: O(log n) time, O(1) space

59. Search in Rotated Sorted Array

Search in rotated sorted array.

```python
def search_rotated(nums, target):
    """
    Array was sorted then rotated. Find target's index.
    """
    pass
```

Test Cases:

- Input: `nums = [4, 5, 6, 7, 0, 1, 2], target = 0` → Output: `4`
- Input: `nums = [4, 5, 6, 7, 0, 1, 2], target = 3` → Output: `-1`

Complexity: O(log n) time, O(1) space

60. Find Minimum in Rotated Sorted Array

Find minimum element in rotated sorted array.

```python
def find_min(nums):
    """
    Find the minimum element in rotated sorted array.
    """
    pass
```

Test Cases:

- Input: `[3, 4, 5, 1, 2]` → Output: `1`
- Input: `[4, 5, 6, 7, 0, 1, 2]` → Output: `0`

Complexity: O(log n) time, O(1) space

61. Merge Sort

Implement merge sort.

```python
def merge_sort(arr):
    """
    Sort array using merge sort.
    """
    pass
```

Test Cases:

- Input: `[38, 27, 43, 3, 9, 82, 10]` → Output: `[3, 9, 10, 27, 38, 43, 82]`

Complexity: O(n log n) time, O(n) space

62. Quick Sort

Implement quick sort.

```python
def quick_sort(arr):
    """
    Sort array using quick sort.
    """
    pass
```

Test Cases:

- Input: `[10, 7, 8, 9, 1, 5]` → Output: `[1, 5, 7, 8, 9, 10]`

Complexity: O(n log n) average, O(n^2) worst; O(log n) space

63. Kth Largest Element

Find kth largest element in unsorted array.

```python
def find_kth_largest(nums, k):
    """
    Find the kth largest element.
    """
    pass
```

Test Cases:

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

Complexity: O(n) average using quickselect

64. Top K Frequent Elements

Find k most frequent elements.

```python
def top_k_frequent(nums, k):
    """
    Return k most frequent elements.
    """
    pass
```

Test Cases:

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

Complexity: O(n log k) with heap or O(n) with bucket sort

65. Search 2D Matrix

Search in row and column sorted matrix.

```python
def search_matrix(matrix, target):
    """
    Search for target in matrix where each row is sorted.
    """
    pass
```

Test Cases:

- Input: `matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 3` → Output: `True`

Complexity: O(log(m × n)) time

Dynamic Programming (Questions 66-80)

66. Climbing Stairs

Count ways to climb n stairs (1 or 2 steps at a time).

```python
def climb_stairs(n):
    """
    Return number of distinct ways to climb to the top.
    """
    pass
```

Test Cases:

- Input: `climb_stairs(2)` → Output: `2` (1+1, 2)
- Input: `climb_stairs(3)` → Output: `3` (1+1+1, 1+2, 2+1)

Complexity: O(n) time, O(1) space

67. House Robber

Maximize money robbed from houses (can't rob adjacent).

```python
```

```
def rob(nums):
    """
    Return maximum amount you can rob without robbing adjacent houses.
    """
    pass
```

Test Cases:

- Input: `rob([1, 2, 3, 1])` → Output: `4` (rob house 1 and 3)
- Input: `rob([2, 7, 9, 3, 1])` → Output: `12` (rob house 1, 3, and 5)

Complexity: O(n) time, O(1) space

68. Coin Change

Find minimum coins needed for amount.

```
python

def coin_change(coins, amount):
    """
    Return fewest number of coins needed to make up amount. Return -1 if impossible.
    """
    pass
```

Test Cases:

- Input: `coins = [1, 2, 5], amount = 11` → Output: `3` (5+5+1)
- Input: `coins = [2], amount = 3` → Output: `-1`

Complexity: O(amount × n) time, O(amount) space

69. Longest Increasing Subsequence

Find length of longest increasing subsequence.

```
python

def length_of_lis(nums):
    """
    Return length of longest strictly increasing subsequence.
    """
    pass
```

Test Cases:

- Input: `[10, 9, 2, 5, 3, 7, 101, 18]` → Output: `4` ([2, 3, 7, 101])
- Input: `[0, 1, 0, 3, 2, 3]` → Output: `4`

Complexity: O(n log n) time with binary search

70. Longest Common Subsequence

Find length of LCS of two strings.

```python
def longest_common_subsequence(text1, text2):
    """
    Return length of longest common subsequence.
    """
    pass
```

Test Cases:

- Input: `text1 = "abcde", text2 = "ace"` → Output: `3` ("ace")
- Input: `text1 = "abc", text2 = "def"` → Output: `0`

Complexity: O(m × n) time, O(m × n) space

71. Word Break

Check if string can be segmented into dictionary words.

```python
def word_break(s, word_dict):
    """
    Return True if s can be segmented into space-separated dictionary words.
    """
    pass
```

Test Cases:

- Input: `s = "leetcode", wordDict = ["leet", "code"]` → Output: `True`
- Input: `s = "applepenapple", wordDict = ["apple", "pen"]` → Output: `True`

Complexity: O(n^2 × m) time where m is max word length

72. Combination Sum

Find all combinations that sum to target.

```python
def combination_sum(candidates, target):
    """
    Return all unique combinations where chosen numbers sum to target.
    Can reuse same number unlimited times.
    """
    pass
```

Test Cases:

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

Complexity: O(N^(T/M)) where T is target, M is min value

73. Decode Ways

Count ways to decode a digit string.

```python
def num_decodings(s):
    """
    'A' -> "1", 'B' -> "2", ..., 'Z' -> "26"
    Return number of ways to decode s.
    """
    pass
```

Test Cases:

- Input: `"12"` → Output: `2` ("AB" or "L")
- Input: `"226"` → Output: `3` ("BZ", "VF", "BBF")

Complexity: O(n) time, O(1) space

74. Unique Paths

Count paths in grid from top-left to bottom-right.

```python
```

```
def unique_paths(m, n):
    """
    Count unique paths in m x n grid (can only move right or down).
    """
    pass
```

Test Cases:

- Input: `m = 3, n = 7` → Output: `28`
- Input: `m = 3, n = 2` → Output: `3`

Complexity: O(m × n) time, O(n) space

75. Jump Game

Check if you can reach the last index.

```
python

def can_jump(nums):
    """
    nums[i] is max jump length from that position.
    Return True if you can reach last index.
    """
    pass
```

Test Cases:

- Input: `[2, 3, 1, 1, 4]` → Output: `True`
- Input: `[3, 2, 1, 0, 4]` → Output: `False`

Complexity: O(n) time, O(1) space

76. Partition Equal Subset Sum

Check if array can be partitioned into two equal sum subsets.

```
python

def can_partition(nums):
    """
    Return True if array can be partitioned into two subsets with equal sum.
    """
    pass
```

Test Cases:

- Input: `[1, 5, 11, 5]` → Output: `True` ([1, 5, 5] and [11])
- Input: `[1, 2, 3, 5]` → Output: `False`

Complexity: O(n × sum) time

77. Edit Distance

Find minimum operations to convert word1 to word2.

```python
def min_distance(word1, word2):
    """
    Operations: insert, delete, replace a character.
    Return minimum number of operations.
    """
    pass
```

Test Cases:

- Input: `word1 = "horse", word2 = "ros"` → Output: `3`
- Input: `word1 = "intention", word2 = "execution"` → Output: `5`

Complexity: O(m × n) time, O(m × n) space

78. Maximum Product Subarray

Find contiguous subarray with largest product.

```python
def max_product(nums):
    """
    Find the contiguous subarray with the largest product.
    """
    pass
```

Test Cases:

- Input: `[2, 3, −2, 4]` → Output: `6` ([2, 3])
- Input: `[−2, 0, −1]` → Output: `0`

Complexity: O(n) time, O(1) space

79. Regular Expression Matching

Implement regex matching with '.' and '*'.

```python
def is_match(s, p):
    """
    '.' matches any single character
    '*' matches zero or more of the preceding element
    """
    pass
```

Test Cases:

- Input: `s = "aa", p = "a"` → Output: `False`
- Input: `s = "aa", p = "a*"` → Output: `True`

Complexity: O(m × n) time

80. Minimum Path Sum

Find minimum path sum from top-left to bottom-right in grid.

```python
def min_path_sum(grid):
    """
    Find a path from top left to bottom right which minimizes sum.
    Can only move down or right.
    """
    pass
```

Test Cases:

- Input: `[[1, 3, 1], [1, 5, 1], [4, 2, 1]]` → Output: `7` (1→3→1→1→1)

Complexity: O(m × n) time, O(1) space

Advanced Topics (Questions 81-100)

81. Sliding Window Maximum

Find maximum in each sliding window.

```python
```

```
def max_sliding_window(nums, k):
    """
    Return max for each window of size k.
    """
    pass
```

Test Cases:

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

Complexity: O(n) time using deque

82. Median of Two Sorted Arrays

Find median of two sorted arrays.

```
python

def find_median_sorted_arrays(nums1, nums2):
    """
    Find the median of two sorted arrays.
    """
    pass
```

Test Cases:

- Input: `nums1 = [1, 3], nums2 = [2]` → Output: `2.0`
- Input: `nums1 = [1, 2], nums2 = [3, 4]` → Output: `2.5`

Complexity: O(log(min(m, n))) time

83. LRU Cache

Implement Least Recently Used cache.

```
python
```

```python
class LRUCache:
    def __init__(self, capacity):
        pass

    def get(self, key):
        pass

    def put(self, key, value):
        pass
```

Test Cases:

- Operations should work in O(1) time

Complexity: O(1) for get and put

84. Trie (Prefix Tree)

Implement a trie.

```python
class TrieNode:
    def __init__(self):
        pass

class Trie:
    def __init__(self):
        pass

    def insert(self, word):
        pass

    def search(self, word):
        pass

    def starts_with(self, prefix):
        pass
```

Complexity: O(m) for all operations where m is word length

85. Word Search

Find if word exists in 2D board.

```python
def exist(board, word):
    """
    Return True if word exists in the grid (can move up/down/left/right).
    """
    pass
```

Test Cases:

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

Complexity: O(m × n × 4^L) where L is word length

86. Meeting Rooms II

Find minimum conference rooms required.

```python
def min_meeting_rooms(intervals):
    """
    intervals[i] = [starti, endi]
    Return minimum number of conference rooms required.
    """
    pass
```

Test Cases:

- Input: `[[0, 30], [5, 10], [15, 20]]` → Output: `2`

Complexity: O(n log n) time

87. Merge Intervals

Merge overlapping intervals.

```python
def merge_intervals(intervals):
    """
    Merge all overlapping intervals.
    """
    pass
```

Test Cases:

- Input: `[[1, 3], [2, 6], [8, 10], [15, 18]]` → Output: `[[1, 6], [8, 10], [15, 18]]`

Complexity: O(n log n) time

88. Insert Interval

Insert new interval and merge if necessary.

```python
def insert_interval(intervals, new_interval):
    """
    Insert newInterval into intervals (sorted and non-overlapping).
    """
    pass
```

Test Cases:

- Input: `intervals = [[1, 3], [6, 9]], newInterval = [2, 5]` → Output: `[[1, 5], [6, 9]]`

Complexity: O(n) time

89. Subsets

Generate all subsets of a set.

```python
def subsets(nums):
    """
    Return all possible subsets (power set).
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3]` → Output: `[[], [1], [2], [1,2], [3], [1,3], [2,3], [1,2,3]]`

Complexity: O(2^n) time

90. Permutations

Generate all permutations.

```python
```

```
def permute(nums):
    """
    Return all possible permutations.
    """
    pass
```

Test Cases:

- Input: `[1, 2, 3]` → Output: `[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

Complexity: O(n!) time

91. N-Queens

Solve the N-Queens problem.

```
python

def solve_n_queens(n):
    """
    Place n queens on n×n chessboard so no two attack each other.
    """
    pass
```

Test Cases:

- Input: `n = 4` → Output: 2 distinct solutions

Complexity: O(n!) time

92. Sudoku Solver

Solve a Sudoku puzzle.

```
python

def solve_sudoku(board):
    """
    Fill the empty cells (marked '.') to solve the puzzle.
    Modify board in-place.
    """
    pass
```

Complexity: O(9^m) where m is number of empty cells

93. Palindrome Partitioning

Partition string into all palindromic substrings.

```python
def partition(s):
    """
    Return all possible palindrome partitioning of s.
    """
    pass
```

Test Cases:

- Input: `"aab"` → Output: `[["a","a","b"], ["aa","b"]]`

Complexity: O(n × 2^n) time

94. Generate Parentheses

Generate all combinations of well-formed parentheses.

```python
def generate_parenthesis(n):
    """
    Generate all combinations of n pairs of parentheses.
    """
    pass
```

Test Cases:

- Input: `n = 3` → Output: `["((()))","(()())","(())()","()(())","()()()"]`

Complexity: O(4^n / √n) time (Catalan number)

95. Largest BST Subtree

Find size of largest BST subtree.

```python
```

```
def largest_bst_subtree(root):
    """
    Find the largest subtree which is a Binary Search Tree.
    """
    pass
```

Complexity: O(n) time

96. Recover Binary Search Tree

Two nodes of BST are swapped, recover the tree.

```
python

def recover_tree(root):
    """
    Two elements were swapped by mistake. Recover without changing structure.
    """
    pass
```

Complexity: O(n) time, O(1) space (Morris traversal)

97. Count of Smaller Numbers After Self

Count smaller elements to the right.

```
python

def count_smaller(nums):
    """
    For each nums[i], count how many numbers to the right are smaller.
    """
    pass
```

Test Cases:

- Input: `[5, 2, 6, 1]` → Output: `[2, 1, 1, 0]`

Complexity: O(n log n) time using merge sort

98. Maximum XOR of Two Numbers

Find maximum XOR of two numbers in array.

```
python
```

```python
def find_maximum_xor(nums):
    """
    Find the maximum result of nums[i] XOR nums[j].
    """
    pass
```

Test Cases:

- Input: `[3, 10, 5, 25, 2, 8]` → Output: `28` (5 XOR 25)

Complexity: O(n) time using trie

99. Alien Dictionary

Derive order of characters in alien language.

```python
def alien_order(words):
    """
    Return a string of unique letters sorted in alien dictionary order.
    """
    pass
```

Test Cases:

- Input: `["wrt","wrf","er","ett","rftt"]` → Output: `"wertf"`

Complexity: O(C) where C is total length of all words

100. Find Median from Data Stream

Design data structure for median from stream.

```python
class MedianFinder:
    def __init__(self):
        pass

    def add_num(self, num):
        pass

    def find_median(self):
        pass
```

Test Cases:

- addNum(1), addNum(2), findMedian() → 1.5

Complexity: O(log n) for addNum, O(1) for findMedian

Study Guide

Topics by Difficulty:

Easy (Good Starting Point): 1-9, 16-17, 26, 33-36, 58, 66-67

Medium (Core Interview Questions): 10-15, 18-25, 27-32, 37-47, 48-57, 59-65, 68-80, 86-90

Hard (Advanced Concepts): 81-85, 91-100

Recommended Study Order:

1. **Arrays & Two Pointers** (1-15)
2. **Linked Lists** (16-25)
3. **Stacks & Queues** (26-32)
4. **Trees - BFS/DFS** (33-47)
5. **Graphs** (48-57)
6. **Binary Search** (58-60)
7. **Sorting** (61-65)
8. **Dynamic Programming** (66-80)
9. **Advanced Topics** (81-100)

Key Patterns to Master:

- Two Pointers
- Sliding Window
- Fast & Slow Pointers
- Binary Search
- BFS/DFS
- Backtracking
- Dynamic Programming (Memoization & Tabulation)
- Greedy Algorithms

- Union Find
- Topological Sort

Time Complexity Cheat Sheet:

- O(1) - Constant
- O(log n) - Logarithmic (Binary Search, Balanced Trees)
- O(n) - Linear (Single Pass)
- O(n log n) - Linearithmic (Efficient Sorting)
- $O(n^2)$ - Quadratic (Nested Loops)
- O(2^n) - Exponential (Subsets, Recursion)
- O(n!) - Factorial (Permutations)

Good luck with your DSA preparation! 🚀