Master DSA for Product-Based Interviews

One Guide to Crack Amazon, Google, Flipkart & Microsoft

(0–3 Years)

15-25 lpa

---

1. Two Sum

Problem: Given an array nums and a target integer target, return indices of two numbers such that they add up to target.

Approach (Hash Map):

- Iterate over the array.
- At each element nums[i], check if target - nums[i] is in a hash map.
- If yes, return the pair of indices.
- Otherwise, store nums[i] in the map.

Python Code:

```
def twoSum(nums, target):
    num_map = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_map:
            return [num_map[complement], i]
        num_map[num] = i
    return []
```

---

2. Longest Substring Without Repeating Characters

Problem:

Find the length of the longest substring without repeating characters.

Approach (Sliding Window):

- Use two pointers, left and right, to represent a window.
- Use a set to track characters in the current window.
- Move right forward and update the longest length.
- If a duplicate is found, move left forward until the duplicate is removed.

Python Code:

```
def lengthOfLongestSubstring(s):
    char_set = set()
    left = 0
    max_len = 0
    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_len = max(max_len, right - left + 1)
    return max_len
```

---

3. Number of Islands

Problem:

Given a 2D grid map of '1's (land) and '0's (water), count the number of islands.

Approach (DFS):

- Iterate over every cell.
- For every '1' found, do a DFS to mark connected land cells as visited ('0').
- Increment island count.

Python Code:

```
def numIslands(grid):
    if not grid:
        return 0
    rows, cols = len(grid), len(grid[0])
    count = 0
    def dfs(r, c):
        if r < 0 or c < 0 or r >= rows or c >= cols or grid[r][c] == '0':
            return
        grid[r][c] = '0'  # Mark visited
        dfs(r + 1, c)
        dfs(r - 1, c)
        dfs(r, c + 1)
        dfs(r, c - 1)
    for r in range(rows):
        for c in range(cols):
            if grid[r][c] == '1':
                dfs(r, c)
                count += 1
    return count
```

---

4. Trapping Rain Water

Problem:

Given an elevation map, compute how much water can be trapped.

Approach (Two Pointers):

- Keep two pointers left and right.
- Track max_left and max_right.
- Water trapped is min(max_left, max_right) - height[i].

Python Code:

```
def trap(height):
    if not height:
        return 0
    left, right = 0, len(height) - 1
    max_left, max_right = height[left], height[right]
    water_trapped = 0
    while left < right:
        if max_left < max_right:
            left += 1
            max_left = max(max_left, height[left])
            water_trapped += max_left - height[left]
        else:
            right -= 1
            max_right = max(max_right, height[right])
            water_trapped += max_right - height[right]
    return water_trapped
```

---

5. Lowest Common Ancestor of Binary Tree

Problem:

Given a binary tree and two nodes p and q, find their lowest common ancestor.

Approach (Recursion):

- Recurse down the tree.
- If current node matches p or q, return it.
- The first node where both left and right are not None is the LCA.

Python Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def lowestCommonAncestor(root, p, q):
    if not root or root == p or root == q:
        return root
    left = lowestCommonAncestor(root.left, p, q)
    right = lowestCommonAncestor(root.right, p, q)
    if left and right:
        return root
    return left if left else right
```

---

6. Valid Parentheses

Problem:

Given a string with only ‘(’, ‘)’, ‘{’, ‘}’, ‘[’, and ‘]’, determine if it is valid (balanced).

Approach (Stack):

- Use a stack to track open brackets.
- Push opening brackets to the stack.
- For closing brackets, check if the top of the stack matches.

Python Code:

```
def isValid(s):
    stack = []
    mapping = {')': '(', '}': '{', ']': '['}
    for char in s:
        if char in mapping:
            top_element = stack.pop() if stack else '#'
            if mapping[char] != top_element:
                return False
        else:
            stack.append(char)
    return not stack
```

---

7. Merge Intervals

Problem:

Given a list of intervals, merge overlapping intervals.

Approach (Sorting):

- Sort intervals by start time.
- Merge overlapping intervals iteratively.

Python Code:

```
def merge(intervals):
    if not intervals:
        return []
    intervals.sort(key=lambda x: x[0])
    merged = [intervals[0]]
    for current in intervals[1:]:
        last = merged[-1]
        if current[0] <= last[1]:
            last[1] = max(last[1], current[1])
        else:
            merged.append(current)
    return merged
```

8. Kth Largest Element in an Array

Problem:

Find the kth largest element in an unsorted array.

Approach (Heap):

- Use a min-heap of size k.

Python Code:

```
import heapq

def findKthLargest(nums, k):
    return heapq.nlargest(k, nums)[-1]
```

---

9. Top K Frequent Elements

Problem:

Return the k most frequent elements in an array.

Approach (HashMap + Heap):

- Count frequency using a map.
- Use a heap to get top k elements.

Python Code:

```
from collections import Counter
import heapq
def topKFrequent(nums, k):
    count = Counter(nums)
    return heapq.nlargest(k, count.keys(), key=count.get)
```

---

10. Binary Tree Inorder Traversal

Problem:

Return inorder traversal of a binary tree.

Approach (Recursion / Stack):

- Either use recursion or iterative stack.

Python Code (Iterative):

```
def inorderTraversal(root):
    res, stack = [], []
    current = root
    while current or stack:
        while current:
            stack.append(current)
            current = current.left
        current = stack.pop()
        res.append(current.val)
        current = current.right
    return res
```

---

11. Coin Change (Minimum Coins)

Problem:

Given coins and an amount, return minimum coins to make the amount.

Approach (DP):

- Bottom-up DP approach.

Python Code:

```
def coinChange(coins, amount):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0
    for coin in coins:
        for x in range(coin, amount + 1):
            dp[x] = min(dp[x], dp[x - coin] + 1)
    return dp[amount] if dp[amount] != float('inf') else -1
```

---

12. Longest Increasing Subsequence

Problem:

Find the length of the longest increasing subsequence.

Approach (DP):

- Use DP or Binary Search for optimization.

Python Code (DP $O(n^2)$):

```
def lengthOfLIS(nums):
    if not nums:
        return 0
    dp = [1] * len(nums)
    for i in range(len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)
    return max(dp)
```

---

13. Word Ladder (Shortest Transformation Sequence)

Problem:

Transform beginWord to endWord using dictionary words, changing one letter at a time.

Approach (BFS):

- Use BFS for shortest path.

Python Code:

```
from collections import deque
def ladderLength(beginWord, endWord, wordList):
    word_set = set(wordList)
    queue = deque([(beginWord, 1)])
    while queue:
        word, length = queue.popleft()
        if word == endWord:
            return length
        for i in range(len(word)):
            for c in 'abcdefghijklmnopqrstuvwxyz':
                next_word = word[:i] + c + word[i+1:]
                if next_word in word_set:
                    word_set.remove(next_word)
                    queue.append((next_word, length + 1))
    return 0
```

---

14. Rotate Image (Matrix)

Problem:

Rotate n x n matrix by 90 degrees in-place.

Approach (Transpose + Reverse Rows):

- Transpose the matrix.
- Reverse each row.

Python Code:

```
def rotate(matrix):
    n = len(matrix)
    # Transpose
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
    # Reverse rows
    for row in matrix:
        row.reverse()
```

---

15. Serialize and Deserialize Binary Tree

Problem:

Serialize a tree to a string and deserialize back.

Approach (DFS):

- Preorder traversal for serialization.
- Rebuild tree during deserialization.

Python Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def serialize(root):
    res = []
    def dfs(node):
        if not node:
            res.append('#')
            return
        res.append(str(node.val))
        dfs(node.left)
        dfs(node.right)
    dfs(root)
    return ','.join(res)
def deserialize(data):
    vals = iter(data.split(','))
    def dfs():
        val = next(vals)
        if val == '#':
            return None
        node = TreeNode(int(val))
```

```
        node.left = dfs()
        node.right = dfs()
        return node
    return dfs()
```

---

16. Find First and Last Position of Element in Sorted Array

Problem:

Given a sorted array nums and target target, find the starting and ending position of the target.

Approach (Binary Search):

- Use binary search twice:
  1. To find the first occurrence.
  2. To find the last occurrence.

Python Code:

```
def searchRange(nums, target):
    def findBound(isFirst):
        left, right = 0, len(nums) - 1
        bound = -1
        while left <= right:
            mid = (left + right) // 2
            if nums[mid] == target:
                bound = mid
                if isFirst:
                    right = mid - 1
                else:
                    left = mid + 1
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
        return bound
    first = findBound(True)
    last = findBound(False)
    return [first, last]
```

---

17. Product of Array Except Self

Problem:

Return an array where each element is the product of all elements except itself, without using division.

Approach (Two-pass):

- Calculate prefix products, then postfix products.

Python Code:

```
def productExceptSelf(nums):
    n = len(nums)
    result = [1] * n
    prefix = 1
    for i in range(n):
        result[i] = prefix
        prefix *= nums[i]
    postfix = 1
    for i in range(n - 1, -1, -1):
        result[i] *= postfix
        postfix *= nums[i]
    return result
```

---

18. Subarray Sum Equals K

Problem:

Find the total number of continuous subarrays whose sum equals k.

Approach (Prefix Sum + HashMap):

- Store prefix sums and use a hashmap to find count of subarrays summing to k.

Python Code:

```
def subarraySum(nums, k):
    count = 0
    sum_map = {0: 1}
    curr_sum = 0
    for num in nums:
        curr_sum += num
        if (curr_sum - k) in sum_map:
            count += sum_map[curr_sum - k]
        sum_map[curr_sum] = sum_map.get(curr_sum, 0) + 1
    return count
```

---

19. Binary Search (Standard Problem)

Problem:

Given a sorted array and a target, return its index, or -1 if not found.

Approach (Binary Search):

- Standard iterative binary search.

Python Code:

```
def binarySearch(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

---

20. Climbing Stairs

Problem:

You can climb 1 or 2 steps at a time. How many distinct ways can you climb to the top of n steps?

Approach (DP – Fibonacci Relation):

- DP where dp[i] = dp[i - 1] + dp[i - 2].

Python Code:

```
def climbStairs(n):
    if n <= 2:
        return n
    first, second = 1, 2
    for _ in range(3, n + 1):
        first, second = second, first + second
    return second
```

Master DSA for Product-Based Interviews – 20 Problems, One Guide to Crack Amazon, Google, Flipkart & Microsoft.

Author: Devkant Bhagat
Source: Curated DSA Case Studies & Interview Experiences
Telegram: Techverse_hub
LinkedIn: Devkant Bhagat