# **Day 1:**

## **Easy Questions**

**Q1. Sum of Array[Iterate through the array and add each element to a cumulative sum.] def array\_sum(arr):**

return sum(arr)

print(array\_sum([1, 2, 3, 4, 5])) # Output: 15

Time Complexity: O(n)O(n)

**Q2. Check Palindrome[Compare the string with its reverse. If they are the same, it's a palindrome.] def is\_palindrome(s):**

return s == s[::-1]

print(is\_palindrome("madam")) # Output: True

Time Complexity: O(n)O(n)

**Q3. Fibonacci Numbers[Start with 0 and 1. Each next number is the sum of the previous two.] def fibonacci(n):**

fib = [0, 1]

for i in range(2, n):

fib.append(fib[i-1] + fib[i-2])

return fib[:n]

print(fibonacci(5)) # Output: [0, 1, 1, 2, 3]

Time Complexity: O(n)O(n)

**Q4. Largest Number in Array[Traverse the array and keep track of the highest value encountered.] def largest\_number(arr):**

return max(arr)

print(largest\_number([1, 3, 5, 2, 4])) # Output: 5

Time Complexity: O(n)O(n)

**Q5. Count Vowels**

def count\_vowels(s):

vowels = set("aeiouAEIOU")

return sum(1 for char in s if char in vowels)

print(count\_vowels("hello world")) # Output: 3

Time Complexity: O(n)O(n)

## **Medium Questions**

**Q6. Binary Search**

def binary\_search(arr, target):

low, high = 0, len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1

print(binary\_search([1, 2, 3, 4, 5], 3)) # Output: 2

Time Complexity: O(logn)O(\log n)

**Q7. Rotate Array Right**

def rotate\_array(arr, k):

k %= len(arr)

return arr[-k:] + arr[:-k]

print(rotate\_array([1, 2, 3, 4, 5], 2)) # Output: [4, 5, 1, 2, 3]

Time Complexity: O(n)O(n)

**Q8. Check Anagrams**

def are\_anagrams(s1, s2):

return sorted(s1) == sorted(s2)

print(are\_anagrams("listen", "silent")) # Output: True Time Complexity: O(nlogn)O(n \log n)

## **Hard Questions**

**Q9. Longest Substring Without Repeating Characters**

def longest\_unique\_substring(s):

char\_map = {}

left = max\_length = 0

for right, char in enumerate(s):

if char in char\_map and char\_map[char] >= left: left = char\_map[char] + 1

char\_map[char] = right

max\_length = max(max\_length, right - left + 1)

return max\_length

print(longest\_unique\_substring("abcabcbb")) # Output: 3

Time Complexity: O(n)O(n)

**Q10. Kth Largest Element**

import heapq

def kth\_largest(arr, k):

return heapq.nlargest(k, arr)[-1]

print(kth\_largest([3, 2, 1, 5, 6, 4], 2)) # Output: 5

Time Complexity: O(nlogk)O(n \log k)

# **Day 2**

## **Easy Questions**

**Q1. Reverse a String**

def reverse\_string(s):

return s[::-1]

print(reverse\_string("hello")) # Output: "olleh"

Time Complexity: O(n)O(n)

**Q2. Find Second Largest Number**

def second\_largest(arr):

arr = list(set(arr))

arr.sort()

return arr[-2] if len(arr) > 1 else None

print(second\_largest([1, 3, 2, 4, 5])) # Output: 4

Time Complexity: O(nlogn)O(n \log n)

**Q3. Check Prime Number**

def is\_prime(n):

if n <= 1:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

print(is\_prime(29)) # Output: True

Time Complexity: O(n)O(\sqrt{n})

**Q4. Find Factorial**

def factorial(n):

if n == 0 or n == 1:

return 1

return n \* factorial(n - 1)

print(factorial(5)) # Output: 120

Time Complexity: O(n)O(n)

**Q5. Remove Duplicates from List**

def remove\_duplicates(arr):

return list(set(arr))

print(remove\_duplicates([1, 2, 2, 3, 3, 4]))

Time Complexity: O(n)O(n)

## **Medium Questions**

**Q6. Merge Two Sorted Arrays**

def merge\_sorted\_arrays(arr1, arr2):

return sorted(arr1 + arr2)

print(merge\_sorted\_arrays([1, 3, 5], [2, 4, 6])) # Output: [1, 2, 3, 4, 5, 6]

Time Complexity: O(nlogn)O(n \log n)

**Q7. Subarray with Given Sum**

def subarray\_with\_sum(arr, target):

current\_sum = 0

prefix\_map = {}

for i, num in enumerate(arr):

current\_sum += num

if current\_sum == target:

return arr[:i + 1]

if current\_sum - target in prefix\_map:

return arr[prefix\_map[current\_sum - target] + 1:i + 1]

prefix\_map[current\_sum] = i

return []

print(subarray\_with\_sum([1, 2, 3, 7, 5], 12)) # Output: [2, 3, 7]

Time Complexity: O(n)O(n)

**Q8. Longest Increasing Subsequence**

def longest\_increasing\_subsequence(arr):

dp = [1] \* len(arr)

for i in range(len(arr)):

for j in range(i):

if arr[i] > arr[j]:

dp[i] = max(dp[i], dp[j] + 1)

return max(dp)

print(longest\_increasing\_subsequence([10, 9, 2, 5, 3, 7, 101, 18]))

Time Complexity: O(n2)O(n^2)

## **Hard Questions**

**Q9. Trap Rainwater Problem**

def trap\_rainwater(height):

left, right = 0, len(height) - 1

max\_left, max\_right = 0, 0

water = 0

while left <= right:

if height[left] <= height[right]:

if height[left] >= max\_left:

max\_left = height[left]

else:

water += max\_left - height[left]

left += 1

else:

if height[right] >= max\_right:

max\_right = height[right]

else:

water += max\_right - height[right]

right -= 1

return water

print(trap\_rainwater([0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1])) # Output: 6

Time Complexity: O(n)O(n)

**Q10. Find All Permutations**

from itertools import permutations

def find\_permutations(s):

return [''.join(p) for p in permutations(s)]

print(find\_permutations("abc")) # Output: ['abc', 'acb', 'bac', 'bca', 'cab', 'cba'] Time Complexity: O(n!)O(n!)

# **Day 3**

## **Easy Questions**

**Q1. Palindrome Check**

def is\_palindrome(s):

return s == s[::-1]

print(is\_palindrome("madam")) # Output: True

Time Complexity: O(n)O(n)

**Q2. Count Vowels**

def count\_vowels(s):

return sum(1 for char in s.lower() if char in "aeiou") print(count\_vowels("ConnectWise")) # Output: 4

Time Complexity: O(n)O(n)

**Q3. Find GCD of Two Numbers**

def gcd(a, b):

while b:

a, b = b, a % b

return a

print(gcd(36, 60)) # Output: 12

Time Complexity: O(log(min(a,b)))O(\log(\min(a, b)))

**Q4. Sum of Digits**

def sum\_of\_digits(n):

return sum(int(d) for d in str(n))

print(sum\_of\_digits(12345)) # Output: 15

Time Complexity: O(d)O(d), where dd is the number of digits.

**Q5. Fibonacci Series up to n Terms**

def fibonacci(n):

a, b = 0, 1

result = []

for \_ in range(n):

result.append(a)

a, b = b, a + b

return result

print(fibonacci(5)) # Output: [0, 1, 1, 2, 3]

Time Complexity: O(n)O(n)

## **Medium Questions**

**Q6. Find Missing Number in Array**

def find\_missing\_number(arr, n):

total = n \* (n + 1) // 2

return total - sum(arr)

print(find\_missing\_number([1, 2, 4, 5, 6], 6)) # Output: 3

Time Complexity: O(n)O(n)

**Q7. Rotate Array by k Steps**

def rotate\_array(arr, k):

k %= len(arr)

return arr[-k:] + arr[:-k]

print(rotate\_array([1, 2, 3, 4, 5], 2)) # Output: [4, 5, 1, 2, 3]

Time Complexity: O(n)O(n)

**Q8. Find Majority Element**

def majority\_element(nums):

count, candidate = 0, None

for num in nums:

if count == 0:

candidate = num

count += 1 if num == candidate else -1

return candidate

print(majority\_element([3, 3, 4, 2, 4, 4, 2, 4, 4])) # Output: 4

Time Complexity: O(n)O(n)

## **Hard Questions**

**Q9. Word Break Problem**

def word\_break(s, word\_dict):

dp = [False] \* (len(s) + 1)

dp[0] = True

for i in range(1, len(s) + 1):

for word in word\_dict:

if dp[i - len(word)] and s[i - len(word):i] == word: dp[i] = True

return dp[-1]

print(word\_break("leetcode", ["leet", "code"])) # Output: True

Time Complexity: O(n⋅m)O(n \cdot m), where nn is the string length and mm is the word dictionary size.

**Q10. Minimum Path Sum in Grid**

def min\_path\_sum(grid):

rows, cols = len(grid), len(grid[0])

for r in range(1, rows):

grid[r][0] += grid[r - 1][0]

for c in range(1, cols):

grid[0][c] += grid[0][c - 1]

for r in range(1, rows):

for c in range(1, cols):

grid[r][c] += min(grid[r - 1][c], grid[r][c - 1])

return grid[-1][-1]

print(min\_path\_sum([[1, 3, 1], [1, 5, 1], [4, 2, 1]])) # Output: 7

Time Complexity: O(m⋅n)O(m \cdot n)

# **Day 4**

## **Easy Questions**

**Q1. Reverse a String**

def reverse\_string(s):

return s[::-1]

print(reverse\_string("ConnectWise")) # Output: "esiWtcennoC"

Time Complexity: O(n)O(n)

**Q2. Check Prime Number**

def is\_prime(n):

if n <= 1:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

print(is\_prime(29)) # Output: True

Time Complexity: O(n)O(\sqrt{n})

**Q3. Count Words in a Sentence**

def count\_words(sentence):

return len(sentence.split())

print(count\_words("ConnectWise is a great company")) # Output: 5 Time Complexity: O(n)O(n)

**Q4. Maximum of Three Numbers**

def max\_of\_three(a, b, c):

return max(a, b, c)

print(max\_of\_three(10, 20, 15)) # Output: 20

Time Complexity: O(1)O(1)

Q5. Generate Multiplication Table

def multiplication\_table(n):

return [n \* i for i in range(1, 11)]

print(multiplication\_table(5)) # Output: [5, 10, 15, ..., 50] Time Complexity: O(10)O(10)

## **Medium Questions**

**Q6. Find Second Largest Element in Array**

def second\_largest(arr):

arr = list(set(arr))

arr.sort()

return arr[-2] if len(arr) >= 2 else None

print(second\_largest([1, 3, 4, 5, 0, 2])) # Output: 4

Time Complexity: O(nlogn)O(n \log n)

**Q7. Merge Two Sorted Arrays**

def merge\_sorted(arr1, arr2):

return sorted(arr1 + arr2)

print(merge\_sorted([1, 3, 5], [2, 4, 6])) # Output: [1, 2, 3, 4, 5, 6] Time Complexity: O((n+m)log(n+m))O((n+m) \log(n+m))

**Q8. Longest Common Prefix**

def longest\_common\_prefix(strs):

if not strs:

return ""

prefix = strs[0]

for s in strs[1:]:

while not s.startswith(prefix):

prefix = prefix[:-1]

return prefix

print(longest\_common\_prefix(["flower", "flow", "flight"])) # Output: "fl"

Time Complexity: O(n⋅k)O(n \cdot k), where nn is the number of strings and kk is the average string length.

## **Hard Questions**

**Q9. Longest Increasing Subsequence**

def length\_of\_lis(nums):

dp = []

for num in nums:

i = 0

while i < len(dp) and dp[i] < num:

i += 1

if i < len(dp):

dp[i] = num

else:

dp.append(num)

return len(dp)

print(length\_of\_lis([10, 9, 2, 5, 3, 7, 101, 18])) # Output: 4

Time Complexity: O(n2)O(n^2)

**Q10. Trap Rain Water**

def trap(height):

left, right = 0, len(height) - 1

left\_max, right\_max = 0, 0

water = 0

while left <= right:

if height[left] < height[right]:

left\_max = max(left\_max, height[left])

water += left\_max - height[left]

left += 1

else:

right\_max = max(right\_max, height[right])

water += right\_max - height[right]

right -= 1

return water

print(trap([0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1])) # Output: 6 Time Complexity: O(n)O(n)

# **Day 5**

## **Easy Questions**

**Q1. Find Factorial**

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

print(factorial(5)) # Output: 120

Time Complexity: O(n)O(n)

**Q2. Reverse an Integer**

def reverse\_integer(n):

sign = -1 if n < 0 else 1

n = abs(n)

reversed\_num = int(str(n)[::-1])

return sign \* reversed\_num

print(reverse\_integer(-123)) # Output: -321

Time Complexity: O(d)O(d), where dd is the number of digits.

**Q3. Find Maximum Element in Array**

def find\_max(arr):

return max(arr)

print(find\_max([1, 3, 5, 2, 4])) # Output: 5

Time Complexity: O(n)O(n)

**Q4. Sum of Elements in Array**

def sum\_array(arr):

return sum(arr)

print(sum\_array([1, 2, 3, 4, 5])) # Output: 15

Time Complexity: O(n)O(n)

**Q5. Convert Decimal to Binary**

def decimal\_to\_binary(n):

return bin(n)[2:]

print(decimal\_to\_binary(10)) # Output: "1010" Time Complexity: O(logn)O(\log n)

## **Medium Questions**

**Q6. Check for Anagram**

def is\_anagram(s1, s2):

return sorted(s1) == sorted(s2)

print(is\_anagram("listen", "silent")) # Output: True Time Complexity: O(nlogn)O(n \log n)

**Q7. Find First Non-Repeating Character**

from collections import Counter

def first\_unique\_char(s):

count = Counter(s)

for char in s:

if count[char] == 1:

return char

return None

print(first\_unique\_char("swiss")) # Output: "w" Time Complexity: O(n)O(n)

**Q8. Rotate Matrix**

def rotate\_matrix(matrix):

return [list(row[::-1]) for row in zip(\*matrix)] print(rotate\_matrix([[1, 2], [3, 4]])) # Output: [[3, 1], [4, 2]] Time Complexity: O(n2)O(n^2)

## **Hard Questions**

**Q9. Longest Palindromic Substring**

def longest\_palindrome(s):

def expand(center, radius):

while center - radius >= 0 and center + radius < len(s) and s[center - radius] == s[center + radius]: radius += 1

return center - radius + 1, center + radius - 1

start, end = 0, 0

for i in range(len(s)):

for radius in [0, 1]: # Odd and even centers

l, r = expand(i, radius)

if r - l > end - start:

start, end = l, r

return s[start:end + 1]

print(longest\_palindrome("babad")) # Output: "bab" or "aba"

Time Complexity: O(n2)O(n^2)

**Q10. N-Queens Problem**

def solve\_n\_queens(n):

def is\_safe(board, row, col):

for i in range(row):

if board[i] == col or \

abs(board[i] - col) == abs(i - row):

return False

return True

def solve(board, row):

if row == n:

result.append(board[:])

return

for col in range(n):

if is\_safe(board, row, col):

board[row] = col

solve(board, row + 1)

board[row] = -1

result = []

solve([-1] \* n, 0)

return result

print(len(solve\_n\_queens(4))) # Output: 2 (solutions)

Time Complexity: O(n!)O(n!)