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!)