1. Question: Given a list of numbers, write a Python function to find the second highest number.

Answer: We can first convert the list into a set to remove duplicates. Then, we'll convert it back to a list and sort it. We can retrieve the second last element to get the second highest number.

```
def second_highest(numbers):
    numbers = list(set(numbers))
    numbers.sort()
    return numbers[-2]

numbers = [1, 3, 2, 4, 4, 5, 6, 6]
print(second_highest(numbers))  # Output: 5
```

---

2. Question: Write a function to compute the factorial of a number using recursion.

Answer:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

number = 5
print(factorial(number))  # Output: 120
```

---

3. Question: You are given a list of strings. Write a function to filter out all strings that are palindromes.

Answer: A palindrome is a word, phrase, number, or other sequences of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization).

```
def is_palindrome(s):
    s = ''.join(e for e in s if e.isalnum())  # Remove punctuation and spaces
    return s.lower() == s.lower()[::-1]

def filter_palindromes(strings):
    return [s for s in strings if is_palindrome(s)]

words = ["radar", "python", "level", "world"]
print(filter_palindromes(words))  # Output: ['radar', 'level']
```

---

4. Question: Given a string, write a function to check if it is an anagram of another string.

Answer: An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

```
def are_anagrams(s1, s2):
    return sorted(s1) == sorted(s2)

str1 = "listen"
str2 = "silent"
print(are_anagrams(str1, str2))  # Output: True
```

---

5. Question: Write a function to flatten a nested list.

Answer:

```
def flatten(lst):
    result = []
    for i in lst:
        if isinstance(i, list):
            result.extend(flatten(i))
        else:
            result.append(i)
    return result

nested_list = [1, [2, 3, [4, 5]], 6]
print(flatten(nested_list))  # Output: [1, 2, 3, 4, 5, 6]
```

Remember, these solutions can be optimized or presented in different ways depending on the context and requirements of the interview.

---

6. Question: Given two lists, write a function that returns the elements that are common to both lists.

Answer:

```
def common_elements(list1, list2):
    return list(set(list1) &amp;amp; set(list2))

list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
print(common_elements(list1, list2))  # Output: [4, 5]
```

---

7. Question: Write a function that returns the number of words in a string.

Answer:

```
def word_count(s):
    return len(s.split())

sentence = "The quick brown fox"
print(word_count(sentence))  # Output: 4
```

---

8. Question: Write a Python function to merge two dictionaries. If both dictionaries have the same key, prefer the second dictionary's value.

Answer:

```
def merge_dicts(dict1, dict2):
    merged = dict1.copy()
    merged.update(dict2)
    return merged

dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
print(merge_dicts(dict1, dict2))  # Output: {'a': 1, 'b': 3, 'c': 4}
```

---

9. Question: Write a function that finds the most repeated character in a string.

Answer:

```
def most_repeated(s):
    char_count = {}
    for char in s:
        if char in char_count:
```

```
            char_count[char] += 1
        else:
            char_count[char] = 1
    max_char = max(char_count, key=char_count.get)
    return max_char

string = "aabbbcdddde"
print(most_repeated(string))  # Output: 'd'
```

---

10. Question: Write a function that checks if a string contains all letters of the alphabet at least once.

Answer:

```
import string

def contains_all_alphabets(s):
    alphabet = set(string.ascii_lowercase)
    return set(s.lower()) >= alphabet

test_string = "The quick brown fox jumps over the lazy dog"
print(contains_all_alphabets(test_string))  # Output: True
```

---

11. Question: Write a function that checks if a given string is a valid IPv4 address.

Answer:

```
def is_valid_ipv4(ip):
    parts = ip.split(".")
    if len(parts) != 4:
        return False
    for item in parts:
        if not item.isdigit():
            return False
        num = int(item)
        if num < 0 or num > 255:
            return False
    return True

address = "192.168.1.1"
print(is_valid_ipv4(address))  # Output: True
```

---

12. Question: Given a list of numbers, write a function to compute the mean, median, and mode.

Answer:

```
from statistics import mean, median, mode

def compute_stats(numbers):
    return {
        "mean": mean(numbers),
        "median": median(numbers),
        "mode": mode(numbers)
    }

numbers = [1, 2, 3, 4, 4, 5, 5, 5, 6]
print(compute_stats(numbers))  # Output: {'mean': 3.89, 'median': 4, 'mode': 5}
```

---

13. Question: Write a function to compute the Fibonacci series up to n.

Answer:

```
def fibonacci(n):
    series = [0, 1]
    while len(series) < n:
        series.append(series[-1] + series[-2])
    return series

number = 10
print(fibonacci(number))  # Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

---

14. Question: Given a string, write a function that returns the first non-repeated character.

Answer:

```
def first_non_repeated(s):
    char_count = {}
    for char in s:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1
    for char in s:
```

```
        if char_count[char] == 1:
            return char
    return None

string = "swiss"
print(first_non_repeated(string))  # Output: 'w'
```

---

15. Question: Write a function to check if two strings are a rotation of each other (e.g., "abcde" and "cdeab" are rotations of each other).

Answer:

```
def are_rotations(str1, str2):
    if len(str1) != len(str2):
        return False
    return str1 in str2 + str2

s1 = "abcde"
s2 = "cdeab"
print(are_rotations(s1, s2))  # Output: True
```

---

16. Question: Write a function to determine if a string has all unique characters (i.e., no character is repeated).

Answer:

```
def has_unique_chars(s):
    return len(s) == len(set(s))

string = "abcdef"
print(has_unique_chars(string))  # Output: True
```

---

17. Question: Write a function that returns the longest consecutive subsequence in a list of numbers.

Answer:

```
def longest_consecutive_subsequence(nums):
    if not nums:
```

```
        return []
    nums = sorted(set(nums))
    longest_streak = []
    current_streak = [nums[0]]

    for i in range(1, len(nums)):
        if nums[i] - nums[i - 1] == 1:
            current_streak.append(nums[i])
        else:
            if len(current_streak) > len(longest_streak):
                longest_streak = current_streak
            current_streak = [nums[i]]

    return longest_streak if len(longest_streak) > len(current_streak) else
current_streak

numbers = [1, 2, 3, 5, 6, 7, 8, 10]
print(longest_consecutive_subsequence(numbers))  # Output: [5, 6, 7, 8]
```

---

18. Question: Write a function to compute the square root of a given non-negative integer `n` without using built-in square root functions or libraries. Return the floor value of the result.

Answer:

```
def sqrt(n):
    if n < 0:
        return None
    if n == 1:
        return 1
    start, end = 0, n
    while start <= end:
        mid = (start + end) // 2
        if mid * mid == n:
            return mid
        elif mid * mid < n:
            start = mid + 1
            ans = mid
        else:
            end = mid - 1
    return ans

number = 17
print(sqrt(number))  # Output: 4
```

---

19. Question: Given a list of integers, write a function to move all zeros to the end of the list while maintaining the order of the other elements.

Answer:

```
def move_zeros(nums):
    count = nums.count(0)
    nums = [num for num in nums if num != 0]
    nums.extend([0] * count)
    return nums

numbers = [1, 2, 0, 4, 0, 5, 6, 0]
print(move_zeros(numbers))  # Output: [1, 2, 4, 5, 6, 0, 0, 0]
```

---

20. Question: Write a function that returns the sum of two numbers represented as strings. Your function should not use built-in arithmetic operators or functions.

Answer:

```
def add_strings(num1, num2):
    res, carry, i, j = "", 0, len(num1) - 1, len(num2) - 1
    while i >= 0 or j >= 0 or carry:
        n1 = int(num1[i]) if i >= 0 else 0
        n2 = int(num2[j]) if j >= 0 else 0
        temp_sum = n1 + n2 + carry
        res = str(temp_sum % 10) + res
        carry = temp_sum // 10
        i, j = i - 1, j - 1
    return res

n1 = "123"
n2 = "789"
print(add_strings(n1, n2))  # Output: "912"
```

---

21. Question: Write a function that checks if a given binary tree is a valid binary search tree.

Answer:

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

```
        self.right = right

def is_valid_bst(root, left=None, right=None):
    if not root:
        return True
    if left and root.value <= left.value:
        return False
    if right and root.value >= right.value:
        return False
    return is_valid_bst(root.left, left, root) and is_valid_bst(root.right, root,
right)

# Example usage:
root = TreeNode(2, TreeNode(1), TreeNode(3))
print(is_valid_bst(root))  # Output: True
```

---

22. Question: Write a function to find the longest common prefix of a list of strings.

Answer:

```
def longest_common_prefix(strings):
    if not strings:
        return ""
    prefix = strings[0]
    for s in strings[1:]:
        while not s.startswith(prefix):
            prefix = prefix[:-1]
    return prefix

strings = ["flower", "flow", "flight"]
print(longest_common_prefix(strings))  # Output: "fl"
```

---

23. Question: Write a function that returns the intersection of two sorted arrays. Assume each array does not have duplicates.

Answer:

```
def intersection_of_sorted_arrays(nums1, nums2):
    i, j = 0, 0
    intersection = []
    while i < len(nums1) and j < len(nums2):
        if nums1[i] == nums2[j]:
            intersection.append(nums1[i])
            i += 1
```

```
            j += 1
        elif nums1[i] < nums2[j]:
            i += 1
        else:
            j += 1
    return intersection

arr1 = [1, 2, 4, 5, 6]
arr2 = [2, 3, 5, 7]
print(intersection_of_sorted_arrays(arr1, arr2))  # Output: [2, 5]
```

---

24. Question: Write a function to determine if two strings are one edit (or zero edits) away.

Answer:

```
def is_one_edit_away(s1, s2):
    if abs(len(s1) - len(s2)) > 1:
        return False

    if len(s1) > len(s2):
        s1, s2 = s2, s1

    i, j, found_difference = 0, 0, False
    while i < len(s1) and j < len(s2):
        if s1[i] != s2[j]:
            if found_difference:
                return False
            found_difference = True
            if len(s1) == len(s2):
                i += 1
        else:
            i += 1
        j += 1
    return True

print(is_one_edit_away("pale", "ple"))   # Output: True
print(is_one_edit_away("pales", "pale")) # Output: True
print(is_one_edit_away("pale", "bale"))  # Output: True
print(is_one_edit_away("pale", "bake"))  # Output: False
```

---

25. Question: Write a function that returns the shortest path in a maze from a start point to an end point, given that you can only move up, down, left, or right. The maze is represented as a 2D list where `0` represents an open path and `1` represents a wall.

Answer:

```
def shortest_path(maze, start, end):
    if not maze or not maze[0]:
        return None
    from collections import deque
    queue = deque([(start, 0)])
    visited = set([start])
    while queue:
        (x, y), steps = queue.popleft()
        if (x, y) == end:
            return steps
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < len(maze) and 0 <= ny < len(maze[0]) and maze[nx][ny] == 0
and (nx, ny) not in visited:
                visited.add((nx, ny))
                queue.append(((nx, ny), steps + 1))
    return -1

maze = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 1],
    [0, 0, 0, 0, 0]
]
start = (0, 0)
end = (4, 4)
print(shortest_path(maze, start, end))  # Output: 12 (or -1 if there's no path)
```

Remember to adapt and explain your code as necessary during an interview, ensuring you understand every line and are prepared to discuss alternative solutions or optimizations.

---

26. Question: Write a function that returns the nth number in the Fibonacci sequence using recursion.

Answer:

```
def fibonacci_recursive(n):
    if n <= 1:
        return n
    else:
        return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)

print(fibonacci_recursive(7))  # Output: 13
```

---

27. Question: Write a function to flatten a nested list of integers. Assume each element is either an integer or a list.

Answer:

```
def flatten(nested_list):
    flat_list = []
    for item in nested_list:
        if isinstance(item, list):
            flat_list.extend(flatten(item))
        else:
            flat_list.append(item)
    return flat_list

nested = [1, [2, 3, [4, 5], 6], 7, [8, 9]]
print(flatten(nested))  # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

---

28. Question: Write a function to check if a given string is a palindrome.

Answer:

```
def is_palindrome(s):
    return s == s[::-1]

string = "radar"
print(is_palindrome(string))  # Output: True
```

---

29. Question: Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if:

- Open brackets are closed by the same type of brackets.
- Open brackets are closed in the correct order.

Answer:

```
def is_valid_brackets(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

brackets = "{[]}"
print(is_valid_brackets(brackets))  # Output: True
```

---

30. Question: Write a function to find the two numbers in a list that sum up to a specific target.

Answer:

```
def two_sum(nums, target):
    num_dict = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_dict:
            return [num_dict[complement], i]
        num_dict[num] = i
    return None

numbers = [2, 7, 11, 15]
target_value = 9
print(two_sum(numbers, target_value))  # Output: [0, 1]
```

---

31. Question: Write a function that reverses a string, but maintains the position of all non-alphabetic characters.

Answer:

```
def reverse_alphabet_only(s):
    s = list(s)
    i, j = 0, len(s) - 1
    while i < j:
        if not s[i].isalpha():
            i += 1
        elif not s[j].isalpha():
            j -= 1
        else:
            s[i], s[j] = s[j], s[i]
```

```
            i += 1
            j -= 1
    return ''.join(s)

string = "ab@cd#ef$gh"
print(reverse_alphabet_only(string))  # Output: "hg@fe#dc$ba"
```

---

32. Question: Write a function to find the first non-repeated character in a string.

Answer:

```
def first_unique_char(s):
    char_count = {}
    for char in s:
        char_count[char] = char_count.get(char, 0) + 1
    for char in s:
        if char_count[char] == 1:
            return char
    return None

string = "swiss"
print(first_unique_char(string))  # Output: "w"
```

---

33. Question: Write a function to find all the prime numbers less than a given number `n`.
Answer:

```
def find_primes(n):
    if n <= 2:
        return []
    primes = [True] * n
    primes[0], primes[1] = False, False
    for i in range(2, int(n ** 0.5) + 1):
        if primes[i]:
            for j in range(i * i, n, i):
                primes[j] = False
    return [i for i, val in enumerate(primes) if val]

number = 30
print(find_primes(number))  # Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

---

34. Question: Given two strings `s` and `t`, write a function to check if `t` is an anagram of `s`.
Answer:

```
def is_anagram(s, t):
    return sorted(s) == sorted(t)

s1 = "listen"
t1 = "silent"
print(is_anagram(s1, t1))  # Output: True
```

---

35. Question: Write a function to compute the factorial of a number using iteration.

Answer:

```
def factorial_iterative(n):
    result = 1
    for i in range(2, n+1):
        result *= i
    return result

number = 5
print(factorial_iterative(number))  # Output: 120
```

As always, understanding the underlying logic and being able to explain your solution is crucial during an interview. It's beneficial to discuss the trade-offs, potential optimizations, and edge cases of each solution.

---

36. Question: Write a function that checks if a given word is an isogram (a word with no repeating letters).

Answer:

```
def is_isogram(word):
    word = word.lower()
    return len(word) == len(set(word))

word = "background"
print(is_isogram(word))  # Output: True
```

---

37. Question: Write a function to rotate an array to the right by `k` steps, where `k` is non-negative.

Answer:

```
def rotate(nums, k):
    k = k % len(nums)  # in case k is larger than the length of nums
    nums[:] = nums[-k:] + nums[:-k]
    return nums

array = [1,2,3,4,5,6,7]
steps = 3
print(rotate(array, steps))  # Output: [5,6,7,1,2,3,4]
```

---

38. Question: Write a function to convert a given integer to its Roman numeral representation.

Answer:

```
def int_to_roman(num):
    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4, 1
    ]
    syms = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
    ]
    roman_num = ''
    i = 0
    while  num > 0:
        for _ in range(num // val[i]):
            roman_num += syms[i]
            num -= val[i]
        i += 1
    return roman_num

number = 3549
print(int_to_roman(number))  # Output: "MMMDXLIX"
```

---

39. Question: Write a function that finds the longest common subsequence (LCS) of two strings.

Answer:

```
def lcs(X, Y):
    m = len(X)
    n = len(Y)
    dp = [[None] * (n + 1) for i in range(m + 1)]

    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0 or j == 0:
                dp[i][j] = 0
            elif X[i-1] == Y[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])
    return dp[m][n]

str1 = "ABCBDAB"
str2 = "BDCAB"
print(lcs(str1, str2))  # Output: 4 (because "BCAB" is a common subsequence)
```

---

40. Question: Write a function to find the square root of a number using the Newton-Raphson method.

Answer:

```
def sqrt_newton(n, tolerance=1e-10, guess=1.0):
    while True:
        better_guess = (guess + n / guess) / 2
        if abs(better_guess - guess) < tolerance:  # Close enough
            return better_guess
        guess = better_guess

number = 25
print(sqrt_newton(number))  # Output: 5.0 (or very close to it)
```

When practicing for interviews, ensure you understand each problem and its solution deeply. The key is to not only get the correct answer but to also explain the reasoning behind your approach, its complexities, and potential improvements.

---

41. Question: Write a function that detects a cycle in a linked list.

Answer:

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

def has_cycle(head):
    slow, fast = head, head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False

# Example Usage:
# node1 = ListNode(1)
# node2 = ListNode(2)
# node3 = ListNode(3)
# node1.next = node2
# node2.next = node3
# node3.next = node1  # Creates a cycle
# print(has_cycle(node1))  # Output: True
```

---

42. Question: Write a function that finds the intersection point of two linked lists.

Answer:

```
def get_intersection_node(headA, headB):
    if not headA or not headB:
        return None

    ptrA, ptrB = headA, headB

    while ptrA != ptrB:
        ptrA = ptrA.next if ptrA else headB
        ptrB = ptrB.next if ptrB else headA

    return ptrA

# Assuming ListNode class definition from the previous question
# Example Usage:
# A: 1 -> 2 -> 3 -> 4
```

```
#                   ↘
#                    5 -> 6 -> 7
#                   ↗
# B:       8 -> 9
# print(get_intersection_node(A, B).value)  # Output: 5
```

---

43. Question: Write a function that computes the power of a number without using the built-in power function or the ** operator.

Answer:

```
def power(base, exp):
    if exp == 0:
        return 1
    if exp < 0:
        base = 1 / base
        exp = -exp

    result = 1
    current_product = base
    while exp > 0:
        if exp % 2 == 1:
            result = result * current_product
        current_product = current_product * current_product
        exp //= 2
    return result

print(power(2, 3))  # Output: 8
print(power(3, -2)) # Output: 0.1111 (or close to it)
```

---

44. Question: Write a function to validate if a given string contains only balanced parentheses. (Only '(' and ')' are considered).

Answer:

```
def is_balanced(s):
    stack = []
    for char in s:
        if char == '(':
            stack.append(char)
        elif char == ')':
            if not stack:
                return False
            stack.pop()
```

```
    return len(stack) == 0

print(is_balanced("(())"))       # Output: True
print(is_balanced("()()"))       # Output: True
print(is_balanced("(()"))        # Output: False
print(is_balanced(")("))         # Output: False
```

---

45. Question: Write a function that returns the longest substring without repeating characters.

Answer:

```
def length_of_longest_substring(s):
    n = len(s)
    ans = 0
    char_index = {}  # Current index of character
    i = 0  # The sliding window left pointer
    for j in range(n):
        if s[j] in char_index:
            i = max(char_index[s[j]], i)
        ans = max(ans, j - i + 1)
        char_index[s[j]] = j + 1
    return ans

print(length_of_longest_substring("abcabcbb"))  # Output: 3 (because "abc" is the
longest substring without repeating characters)
```

When working on these questions, it's always a good idea to explore the problems in depth, analyze different approaches, and understand the time and space complexities of your solutions.

---

46. Question: Given a string `s` and a string `t`, find all the start indices of `t`'s anagrams in `s`. Strings consist of lowercase English letters only and the length of both strings `s` and `t` will not be larger than 20,000.
Answer:

```
from collections import Counter

def find_anagrams(s, t):
```

```
    t_counter = Counter(t)
    s_counter = Counter(s[:len(t)-1])
    res = []
    for i in range(len(t)-1, len(s)):
        s_counter[s[i]] += 1   # include a new char in the window
        if s_counter == t_counter:   # This step is O(1), as there are at most 26
English letters
            res.append(i-len(t)+1)   # append the starting index
        s_counter[s[i-len(t)+1]] -= 1   # decrease the count of oldest char in the
window
        if s_counter[s[i-len(t)+1]] == 0:
            del s_counter[s[i-len(t)+1]]   # remove the count if it is 0
    return res

s = "cbaebabacd"
t = "abc"
print(find_anagrams(s, t))  # Output: [0, 6]
```

---

47. Question: Given an unsorted integer array, find the smallest missing positive integer.

Answer:

```
def first_missing_positive(nums):
    n = len(nums)

    # First, mark all negative values as 'n + 1'
    for i in range(n):
        if nums[i] <= 0:
            nums[i] = n + 1

    # Place each number in its correct position
    for num in nums:
        if 1 <= num <= n:
            nums[num-1], num = num, nums[num-1]

    # The first place where its number is not correct
    for i, num in enumerate(nums, 1):
        if num != i:
            return i

    return n + 1

nums = [3, 4, -1, 1]
print(first_missing_positive(nums))  # Output: 2
```

---

48. Question: Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary). You may assume that the intervals were initially sorted according to their start times.

Answer:

```
def insert_interval(intervals, new_interval):
    merged = []
    i, n = 0, len(intervals)

    # Add all the intervals starting before new_interval
    while i < n and intervals[i][1] < new_interval[0]:
        merged.append(intervals[i])
        i += 1

    # Merge all overlapping intervals to one considering new_interval
    while i < n and intervals[i][0] <= new_interval[1]:
        new_interval[0] = min(new_interval[0], intervals[i][0])
        new_interval[1] = max(new_interval[1], intervals[i][1])
        i += 1

    # Add the union of intervals we got
    merged.append(new_interval)

    # Add all the rest
    while i < n:
        merged.append(intervals[i])
        i += 1

    return merged

intervals = [[1, 3], [6, 9]]
new_interval = [2, 5]
print(insert_interval(intervals, new_interval))  # Output: [[1, 5], [6, 9]]
```

---

49. Question: Implement a basic calculator to evaluate a simple expression string containing non-negative integers, '+', '-', '*', and '/' operators. You can assume the given expression is always valid.

Answer:

```
def calculate(s):
    if not s:
        return 0

    stack, num, sign = [], 0, "+"
    for i in range(len(s)):
        if s[i].isdigit():
```

```
            num = num * 10 + int(s[i])
        if s[i] in "+-*/" or i == len(s) - 1:
            if sign == "+":
                stack.append(num)
            elif sign == "-":
                stack.append(-num)
            elif sign == "*":
                stack.append(stack.pop() * num)
            else:  # division
                top = stack.pop()
                if top < 0:
                    stack.append(-(-top // num))
                else:
                    stack.append(top // num)
            num = 0
            sign = s[i]
    return sum(stack)

expression = "3+2*2"
print(calculate(expression))  # Output: 7
```

---

50. Question: Design a data structure that supports the following two operations:

- `void addWord(word)`
- `bool search(word)`

The `search` function should be able to search a literal word or a regular expression string containing only letters `a-z` or `.`. The `.` period should be able to represent any one letter.
Answer:

```
class WordDictionary:

    def __init__(self):
        self.trie = {}

    def addWord(self, word):
        node = self.trie
        for w in word:
            if w not in node:
                node[w] = {}
            node = node[w]
        node['$'] = True

    def search(self, word):
        def search_in_node(word, node):
            for i, ch in enumerate(word):
                if not ch in node:
                    # If the current character is '.', check all possible nodes at
this level
```

```
                    if ch == '.':
                        for x in node:
                            if x != '$' and search_in_node(word[i + 1:], node[x]):
                                return True
                    # if no nodes lead to answer, or the current character != '.'
                    return False
                # if the character is found, go down to the next level in trie
                node = node[ch]
            return '$' in node

        return search_in_node(word, self.trie)

# Example Usage:
# dictionary = WordDictionary()
# dictionary.addWord("bad")
# dictionary.addWord("dad")
# dictionary.addWord("mad")
# print(dictionary.search("pad"))  # Output: False
# print(dictionary.search("bad"))  # Output: True
# print(dictionary.search(".ad"))  # Output: True
# print(dictionary.search("b.."))  # Output: True
```

The complexity of problems you could be asked during an interview can vary significantly based on the company and role. Always be prepared for both algorithmic challenges and more practical problems that may be closer to the actual work in the job you're applying for.

---

51. Question: Given a list of words, group the anagrams together.

Answer:

```
from collections import defaultdict

def group_anagrams(words):
    anagrams = defaultdict(list)
    for word in words:
        # Use sorted word as a key. All anagrams will result in the same key.
        sorted_word = ''.join(sorted(word))
        anagrams[sorted_word].append(word)
    return list(anagrams.values())

words = ["eat", "tea", "tan", "ant", "bat"]
print(group_anagrams(words))  # Output: [['eat', 'tea'], ['tan', 'ant'], ['bat']]
```

Time Complexity:

- Sorting each word takes O(KlogK) where K is the maximum length of a word.
- Doing this for all words takes O(NKlogK) where N is the number of words.

---

52. Question: Given an array nums and a target value, find the two numbers in the array that sum up to the target value.

Answer:

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

nums = [2, 7, 11, 15]
target = 9
print(two_sum(nums, target))  # Output: [0, 1]
```

Time Complexity:

- O(N) where N is the number of elements in the array.

---

53. Question: Find the longest palindromic substring in a string.

Answer:

```
def longest_palindrome(s):
    if not s:
        return ""

    longest = ""
    for i in range(len(s)):
        # Odd length palindromes
        p1 = expand_from_center(s, i, i)
        if len(p1) > len(longest):
            longest = p1

        # Even length palindromes
        p2 = expand_from_center(s, i, i + 1)
        if len(p2) > len(longest):
```

```
            longest = p2

    return longest

def expand_from_center(s, l, r):
    while l >= 0 and r < len(s) and s[l] == s[r]:
        l -= 1
        r += 1
    return s[l + 1:r]

s = "babad"
print(longest_palindrome(s))  # Output: "bab" or "aba"
```

Time Complexity:

- O(N^2) where N is the length of the string.

---

54. Question: Implement a function to serialize and deserialize a binary tree.

Answer:

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

def serialize(root):
    def helper(node):
        if not node:
            return ["null"]
        return [str(node.val)] + helper(node.left) + helper(node.right)

    return ','.join(helper(root))

def deserialize(data):
    def helper(nodes):
        val = nodes.pop(0)
        if val == "null":
            return None
        node = TreeNode(int(val))
        node.left = helper(nodes)
        node.right = helper(nodes)
        return node

    nodes = data.split(',')
    return helper(nodes)

# Usage:
```

```
# node = TreeNode(1, TreeNode(2), TreeNode(3, TreeNode(4), TreeNode(5)))
# s = serialize(node)
# print(s)  # Output: "1,2,null,null,3,4,null,null,5,null,null"
# new_node = deserialize(s)
```

Time Complexity:

- Both serialization and deserialization are O(N) where N is the number of nodes in the tree.

---

55. Question: Determine if a given binary tree is a valid binary search tree.

Answer:

```
def is_valid_bst(root):
    def helper(node, lower=float('-inf'), upper=float('inf')):
        if not node:
            return True

        val = node.val
        if val <= lower or val >= upper:
            return False

        if not helper(node.right, val, upper):
            return False
        if not helper(node.left, lower, val):
            return False

        return True

    return helper(root)

# Assuming TreeNode class definition from the previous question
# Example Usage:
# node = TreeNode(2, TreeNode(1), TreeNode(3))
# print(is_valid_bst(node))  # Output: True
```

Time Complexity:

- O(N) where N is the number of nodes in the tree.

---

56. Question: Given an array of integers, find out whether there are two distinct indices `i` and `j` in the array such that the absolute difference

between `nums[i]` and `nums[j]` is at most `t` and the absolute difference between `i` and `j` is at most `k`.

Answer:

```
from sortedcontainers import SortedList

def contains_nearby_almost_duplicate(nums, k, t):
    if t < 0: return False
    slist, n = SortedList(), len(nums)
    for i in range(n):
        if i > k: slist.remove(nums[i - k - 1])
        pos1 = slist.bisect_left(nums[i] - t)
        pos2 = slist.bisect_right(nums[i] + t)

        if pos1 != pos2:
            return True

        slist.add(nums[i])
    return False

nums = [1, 2, 3, 1]
k = 3
t = 0
print(contains_nearby_almost_duplicate(nums, k, t))  # Output: True
```

Time Complexity:

- O(N log(min(N, K))) where N is the number of elements in the array.

---

57. Question: Find the kth largest element in an unsorted array.

Answer:

```
import heapq

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

nums = [3, 2, 1, 5, 6, 4]
k = 2
print(find_kth_largest(nums, k))  # Output: 5
```

Time Complexity:

- O(N log k) where N is the number of elements in the array.

---

58. Question: Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

Answer:

```
def word_break(s, wordDict):
    wordSet, n = set(wordDict), len(s)
    dp = [False] * (n + 1)
    dp[0] = True

    for i in range(1, n + 1):
        for j in range(i):
            if dp[j] and s[j:i] in wordSet:
                dp[i] = True
                break
    return dp[-1]

s = "leetcode"
wordDict = ["leet", "code"]
print(word_break(s, wordDict))  # Output: True
```

Time Complexity:

- O(N^2) where N is the length of the string.

---

59. Question: Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

Answer:

```
def search_insert(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 left

nums = [1, 3, 5, 6]
target = 5
```

```
print(search_insert(nums, target))  # Output: 2
```

Time Complexity:

- O(log N) where N is the number of elements in the array.

---

60. Question: Rotate an array to the right by k steps.

Answer:

```
def rotate(nums, k):
    n = len(nums)
    k %= n
    nums[:] = nums[-k:] + nums[:-k]

nums = [1, 2, 3, 4, 5, 6, 7]
k = 3
rotate(nums, k)
print(nums)  # Output: [5, 6, 7, 1, 2, 3, 4]
```

Time Complexity:

- O(N) where N is the number of elements in the array.

Remember, these solutions are efficient for the given problems, but some might have alternative methods to approach them. Always try to think about edge cases, and sometimes, the interviewer might be interested in a specific method even if it's not the most optimal one.

---

61. Question: Given an array of integers, every element appears twice except for one. Find that single one.

Answer:

```
def singleNumber(nums):
    result = 0
    for num in nums:
        result ^= num
    return result
```

```
nums = [4, 1, 2, 1, 2]
print(singleNumber(nums))  # Output: 4
```

Time Complexity:

- O(N) where N is the number of elements in the array.

---

62. Question: Write a function to determine the number of bits you would need to flip to convert integer A to integer B.

Answer:

```
def bitSwapRequired(A, B):
    count = 0
    c = A ^ B  # c will have 1s wherever A and B are different
    while c:
        count += c &amp;amp; 1
        c >>= 1
    return count

A = 29  # 11101
B = 15  # 01111
print(bitSwapRequired(A, B))  # Output: 2
```

Time Complexity:

- O(1), since the size of the integers is fixed (typically 32 or 64 bits).

---

63. Question: Given two strings, write a method to decide if one is a permutation of the other.

Answer:

```
from collections import Counter

def is_permutation(str1, str2):
    return Counter(str1) == Counter(str2)

str1 = "listen"
str2 = "silent"
print(is_permutation(str1, str2))  # Output: True
```

Time Complexity:

- O(N), where N is the length of the longer string.

---

64. Question: You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

Answer:

```
def rotate(matrix):
    n = len(matrix)
    # Transpose the matrix
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

    # Reverse the columns
    for row in matrix:
        row.reverse()

matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
]
rotate(matrix)
print(matrix)  # Output: [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

Time Complexity:

- O(N^2), where N is the number of rows (or columns) in the matrix.

---

65. Question: Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if the brackets are closed in the correct order.

Answer:

```
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

s = "{[]}"
print(isValid(s))  # Output: True
```

Time Complexity:

- O(N), where N is the length of the string.

As always, during an interview, focus not only on the solution but also on communicating your thought process, justifying your choices, and being open to suggestions or alternative approaches.

---

66. Question: Write a function to detect a cycle in a linked list.

Answer:

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

def hasCycle(head):
    slow, fast = head, head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

Time Complexity:

- O(N), where N is the number of nodes in the linked list.

---

67. Question: Given a sorted linked list, delete all duplicates such that each element appears only once.

Answer:

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

def deleteDuplicates(head):
    current = head
    while current and current.next:
        if current.next.val == current.val:
            current.next = current.next.next
        else:
            current = current.next
    return head
```

Time Complexity:

- O(N), where N is the number of nodes in the linked list.

---

68. Question: Implement a basic calculator to evaluate a simple expression string containing non-negative integers, '+', '-', '*', and '/' operators. Assume the expression is always valid.

Answer:

```
def calculate(s):
    stack, num, sign = [], 0, '+'
    for i, c in enumerate(s):
        if c.isdigit():
            num = num * 10 + int(c)
        if c in "+-*/" or i == len(s) - 1:
            if sign == '+':
                stack.append(num)
            elif sign == '-':
                stack.append(-num)
            elif sign == '*':
                stack[-1] *= num
            elif sign == '/':
                stack[-1] = int(stack[-1] / num)
            num, sign = 0, c
    return sum(stack)

s = "3+2*2"
```

```
print(calculate(s))  # Output: 7
```

Time Complexity:

- O(N), where N is the length of the string.

---

69. Question: Design and implement a TwoSum class. It should support the following operations: `add` and `find`.

- `add`: Add the number to an internal data structure.
- `find`: Find if there exists any pair of numbers which sum is equal to the value.

Answer:

```
class TwoSum:

    def __init__(self):
        self.data = {}

    def add(self, number):
        if number in self.data:
            self.data[number] += 1
        else:
            self.data[number] = 1

    def find(self, value):
        for num in self.data:
            complement = value - num
            if complement in self.data:
                if complement != num or self.data[num] > 1:
                    return True
        return False
```

Time Complexity:

- `add` operation: O(1)
- `find` operation: O(N), where N is the number of unique numbers added so far.

---

70. Question: Write a function to flatten a nested dictionary. Namespace the keys with a period.

Answer:

```
def flatten_dictionary(d, parent_key='', sep='.'):
    items = {}
    for k, v in d.items():
        new_key = f"{parent_key}{sep}{k}" if parent_key else k
        if isinstance(v, dict):
            items.update(flatten_dictionary(v, new_key, sep=sep))
        else:
            items[new_key] = v
    return items

nested_dict = {
    "a": 1,
    "b": {
        "c": 2,
        "d": {
            "e": 3
        }
    }
}
print(flatten_dictionary(nested_dict))  # Output: {'a': 1, 'b.c': 2, 'b.d.e': 3}
```

Time Complexity:

- O(N), where N is the total number of keys in the dictionary (including nested keys).

---

71. Question: Find the longest substring without repeating characters.

Answer:

```
def length_of_longest_substring(s):
    n = len(s)
    set_ = set()
    ans = 0
    i, j = 0, 0
    while i < n and j < n:
        if s[j] not in set_:
            set_.add(s[j])
            j += 1
            ans = max(ans, j - i)
        else:
            set_.remove(s[i])
            i += 1
    return ans

s = "abcabcbb"
```

```
print(length_of_longest_substring(s))  # Output: 3
```

Time Complexity:

- O(2N) = O(N), where N is the length of the string. The worst case will be checking each character twice with i and j.

---

72. Question: Serialize and deserialize a binary tree.

Answer:

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

class Codec:

    def serialize(self, root):
        if not root:
            return 'None'
        return str(root.val) + ',' + self.serialize(root.left) + ',' +
self.serialize(root.right)

    def deserialize(self, data):
        def helper(data_list):
            if data_list[0] == 'None':
                data_list.pop(0)
                return None
            root = TreeNode(data_list[0])
            data_list.pop(0)
            root.left = helper(data_list)
            root.right = helper(data_list)
            return root

        data_list = data.split(',')
        return helper(data_list)
```

Time Complexity:

- Serialize: O(N), where N is the number of nodes.
- Deserialize: O(N), where N is the number of nodes.

---

73. Question: Write a function to match string s against pattern p, where p can have characters and also `.` which matches any character, and `*` which matches zero or more of the preceding element.

Answer:

```
def is_match(s, p):
    if not p:
        return not s

    first_match = bool(s) and p[0] in {s[0], '.'}

    if len(p) >= 2 and p[1] == '*':
        return (is_match(s, p[2:]) or
                first_match and is_match(s[1:], p))
    else:
        return first_match and is_match(s[1:], p[1:])

s = "mississippi"
p = "mis*is*p*."
print(is_match(s, p))  # Output: False
```

Time Complexity:

- In the worst case, the time complexity is O((N+P)2^(N+P/2)), where N is the length of the string and P is the length of the pattern.

---

74. Question: Find the peak element in an array. A peak element is an element which is greater than or equal to its neighbors. Assume the array is sorted in ascending order, and then a peak is found, then it is sorted in descending order. Also, assume the array may have duplicates.

Answer:

```
def find_peak_element(nums):
    l, r = 0, len(nums) - 1
    while l < r:
        mid = (l + r) // 2
        if nums[mid] < nums[mid + 1]:
            l = mid + 1
        else:
            r = mid
    return l

nums = [1, 2, 3, 4, 5, 6, 7, 5, 4, 3, 2]
print(find_peak_element(nums))  # Output: 6
```

Time Complexity:

- O(log N), where N is the number of elements in the array.

---

75. Question: Implement the `strStr()` function. Return the index of the first occurrence of needle in haystack, or `-1` if needle is not part of haystack.
Answer:

```
def strStr(haystack, needle):
    if not needle:
        return 0
    needle_length = len(needle)
    for i in range(len(haystack) - needle_length + 1):
        if haystack[i:i + needle_length] == needle:
            return i
    return -1

haystack = "hello"
needle = "ll"
print(strStr(haystack, needle))  # Output: 2
```

Time Complexity:

- O((N-M) * M), where N is the length of the haystack and M is the length of the needle.

I hope these questions and solutions help in preparing for technical interviews. Remember to explain your approach and thought process before diving into code. It's as much about problem-solving as it is about coding.

---

76. Question: Find the shortest path in a binary matrix from the top-left corner to the bottom-right corner. You can move up, down, left, right, and diagonally if the adjacent cells contain a 0. The path should avoid cells with a 1.

Answer:

```
from collections import deque
```

```
def shortest_path_binary_matrix(grid):
    if not grid or not grid[0] or grid[0][0] or grid[-1][-1]:
        return -1

    n, m = len(grid), len(grid[0])
    directions = [(0, 1), (1, 0), (1, 1), (-1, -1), (0, -1), (-1, 0), (1, -1), (-1,
1)]
    queue = deque([(0, 0, 1)])

    while queue:
        x, y, dist = queue.popleft()
        if x == n - 1 and y == m - 1:
            return dist
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < n and 0 <= ny < m and not grid[nx][ny]:
                grid[nx][ny] = 1
                queue.append((nx, ny, dist + 1))

    return -1

grid = [[0,0,0],[1,1,0],[1,1,0]]
print(shortest_path_binary_matrix(grid))  # Output: 4
```

Time Complexity:

- O(N*M), where N and M are the dimensions of the grid.

---

77. Question: Design a data structure that supports the following two operations: `void addWord(word)` and `bool search(word)`. The search method can search a literal word or a regular expression string containing only letters `a-z` or `.`. A `.` means it can represent any one-letter.

Answer:

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end = False

class WordDictionary:
    def __init__(self):
        self.root = TrieNode()

    def addWord(self, word):
        node = self.root
        for ch in word:
            if ch not in node.children:
                node.children[ch] = TrieNode()
```

```
            node = node.children[ch]
        node.is_end = True

    def search(self, word):
        return self.match(word, 0, self.root)

    def match(self, word, index, node):
        if index == len(word):
            return node.is_end
        if word[index] != '.':
            return word[index] in node.children and self.match(word, index + 1,
node.children[word[index]])
        for child in node.children.values():
            if self.match(word, index + 1, child):
                return True
        return False

wd = WordDictionary()
wd.addWord("bad")
wd.addWord("dad")
wd.addWord("mad")
print(wd.search("pad"))  # Output: False
print(wd.search("bad"))  # Output: True
print(wd.search(".ad"))  # Output: True
```

Time Complexity:

- `addWord`: O(L), where L is the length of the word.
- `search`: In the worst case, O(N*26^L), where L is the length of the word and N is the number of inserted words.

---

78. Question: Find the kth largest element in an unsorted array.

Answer:

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

nums = [3,2,3,1,2,4,5,5,6]
k = 4
print(findKthLargest(nums, k))  # Output: 4
```

Time Complexity:

- O(Nlogk), where N is the length of the nums array.

---

79. Question: Given a list of integers, return the number of good pairs. A pair `(i, j)` is called good if `nums[i] == nums[j]` and `i < j`.

Answer:

```
def numIdenticalPairs(nums):
    from collections import Counter
    count = Counter(nums)
    return sum(v*(v-1)//2 for v in count.values())

nums = [1,2,3,1,1,3]
print(numIdenticalPairs(nums))  # Output: 4
```

Time Complexity:

- O(N), where N is the length of the nums list.

---

80. Question: Find if a given string can be formed by a sequence of one or more palindrome strings.

Answer:

```
def can_form_palindrome(s):
    from collections import Counter
    count = Counter(s)
    return sum(v % 2 for v in count.values()) <= 1

s = "aabb"
print(can_form_palindrome(s))  # Output: True
```

Time Complexity:

- O(N), where N is the length of the string s.