Here's a comprehensive list of interview questions on data structures and algorithms, along with their answers and Python code:

1. Reverse a String

- **Question**: Reverse a given string.
- **Answer**:

```
def reverse_string(s):
    return s[::-1]
```

2. Check Palindrome

- **Question**: Check if a string is a palindrome.
- **Answer**:

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

3. Find Factorial

- **Question**: Calculate the factorial of a number.
- **Answer**:

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

4. Fibonacci Series

- **Question**: Generate the nth Fibonacci number.
- **Answer**:

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

5. Linked List Cycle Detection

- **Question**: Detect if there is 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(node):
    slow, fast = node, node
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

6. Merge Two Sorted Lists

- **Question**: Merge two sorted linked lists.
- **Answer**:

```
def merge_sorted_lists(l1, l2):
    dummy = ListNode(0)
    current = dummy

    while l1 and l2:
        if l1.value < l2.value:
            current.next, l1 = l1, l1.next
        else:
            current.next, l2 = l2, l2.next
        current = current.next

    current.next = l1 or l2
    return dummy.next
```

7. Find the Middle of Linked List

- **Question**: Find the middle element of a linked list.
- **Answer**:

```
def find_middle(node):
    slow, fast = node, node
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
    return slow
```

8. Maximum Subarray Sum

- **Question**: Find the maximum subarray sum using Kadane's algorithm.
- **Answer**:

```
def max_subarray(nums):
    max_current = max_global = nums[0]
    for i in range(1, len(nums)):
        max_current = max(nums[i], max_current + nums[i])
        max_global = max(max_global, max_current)
    return max_global
```

9. Check if a Tree is Balanced

- **Question**: Check if a binary tree is balanced.

- **Answer**:

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

def is_balanced(root):
    def check_balance(node):
        if not node:
            return 0, True
        left_height, left_balanced = check_balance(node.left)
        right_height, right_balanced = check_balance(node.right)
        return max(left_height, right_height) + 1, left_balanced and
right_balanced and abs(left_height - right_height) <= 1

    return check_balance(root)[1]
```

10. Breadth-first Search (BFS) in Graph

- **Question**: Implement BFS for a graph.
- **Answer**:

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
    return visited
```

11. Depth-first Search (DFS) in Graph

- **Question**: Implement DFS for a graph.
- **Answer**:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    for vertex in graph[start] - visited:
        dfs(graph, vertex, visited)
    return visited
```

12. Implement a Stack

- **Question**: Implement a stack using linked list.
- **Answer**:

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

class Stack:
    def __init__(self):
        self.top = None

    def push(self, value):
        self.top = StackNode(value, self.top)

    def pop(self):
        if not self.top:
            return None
        value = self.top.value
        self.top = self.top.next
        return value

    def peek(self):
        return None if not self.top else self.top.value

    def is_empty(self):
        return self.top is None
```

13. Implement a Queue

- **Question**: Implement a queue using two stacks.
- **Answer**:

```
class Queue:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, value):
        self.stack1.append(value)

    def dequeue(self):
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop() if self.stack2 else None
```

14. Implement a Priority Queue

- **Question**: Implement a priority queue using a heap.
- **Answer**:

```
import heapq

class PriorityQueue:
    def __init__(self):
        self.queue = []

    def enqueue(self, value, priority=0):
        heapq.heappush(self.queue, (priority, value))

    def dequeue(self):
        return heapq.heappop(self.queue)[1] if self.queue else None
```

15. Implement Hashmap

- **Question**: Implement a simple hashmap.
- **Answer**:

```
class Hashmap:
    def __init__(self):
        self.size = 1000
        self.map = [None] * self.size

    def _hash(self, key):
        return hash(key) % self.size

    def put(self, key, value):
        key_hash = self._hash(key)
        self.map[key_hash] = value

    def get(self, key):
        key_hash = self._hash(key)
        return self.map[key_hash]

    def remove(self, key):
        key_hash = self._hash(key)
        self.map[key_hash] = None
```

16. Binary Search

- **Question**: Implement binary search for a sorted list.
- **Answer**:

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

17. Implement Trie (Prefix Tree)

- **Question**: Implement a basic trie for word insert, search and prefix search.
- **Answer**:

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

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

    def insert(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True

    def search(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                return False
            node = node.children[char]
        return node.is_end_of_word

    def starts_with(self, prefix):
        node = self.root
        for char in prefix:
            if char not in node.children:
                return False
            node = node.children[char]
        return True
```

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

- **Question**: Given a sorted array of integers and a target value, find the starting and ending position of the target value using binary search.
- **Answer**:

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

    left, right = find_left_boundary(nums, target),
find_left_boundary(nums, target + 1) - 1
    if left <= right:
        return [left, right]
    return [-1, -1]
```

19. Topological Sort

- **Question**: Implement a topological sort for a directed graph.
- **Answer**:

```
from collections import defaultdict, deque

def topological_sort(vertices, edges):
    graph = defaultdict(list)
    in_degree = {v: 0 for v in vertices}

    for u, v in edges:
        graph[u].append(v)
        in_degree[v] += 1

    queue = deque([v for v, d in in_degree.items() if d == 0])
    order = []

    while queue:
        vertex = queue.popleft()
        order.append(vertex)
        for neighbor in graph[vertex]:
            in_degree[neighbor] -= 1
            if in_degree[neighbor] == 0:
                queue.append(neighbor)

    return order if len(order) == len(vertices) else []
```

20. Check if a String Contains All Binary Codes of Size K

- **Question**: Given a binary string `s` and an integer `k`, check if all binary codes of length `k` is a substring of `s`.
- **Answer**:

```
def has_all_codes(s, k):
    needed = 1 << k
    seen = set()

    for i in range(len(s) - k + 1):
        substring = s[i:i+k]
        if substring not in seen:
            seen.add(substring)
            needed -= 1
            if needed == 0:
                return True

    return False
```

21. Implement QuickSort

- **Question**: Implement the quicksort algorithm.
- **Answer**:

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)
```

22. Implement MergeSort

- **Question**: Implement the mergesort algorithm.
- **Answer**:

```
def mergesort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]
    return merge(mergesort(left), mergesort(right))

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
```

23. Maximum Depth of Binary Tree

- **Question**: Find the maximum depth of a binary tree.
- **Answer**:

```
def max_depth(root):
    if not root:
        return 0
    left_depth = max_depth(root.left)
    right_depth = max_depth(root.right)
    return max(left_depth, right_depth) + 1
```

24. Count the Number of Islands

- **Question**: Given a 2D grid consisting of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically.
- **Answer**:

```
def num_islands(grid):
    if not grid:
        return 0

    count = 0
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1':
                dfs(grid, i, j)
                count += 1
    return count

def dfs(grid, i, j):
    if (i < 0 or i >= len(grid) or j < 0 or j >= len(grid[0]) or grid[i][j]
!= '1'):
        return
    grid[i][j] = '#'
    dfs(grid, i-1, j)
    dfs(grid, i+1, j)
    dfs(grid, i, j-1)
    dfs(grid, i, j+1)
```

25. Trapping Rain Water

- **Question**: Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.
- **Answer**:

```
def trap(height):
    if not height:
        return 0

    n = len(height)
    left_max = [0] * n
    right_max = [0] * n

    left_max[0] = height[0]
    for i in range(1, n):
        left_max[i] = max(height[i], left_max[i-1])

    right_max[n-1] = height[n-1]
    for i in range(n-2, -1, -1):
        right_max[i] = max(height[i], right_max[i+1])

    ans = 0
    for i in range(n):
        ans += min(left_max[i], right_max[i]) - height[i]

    return ans
```

26. Coin Change

- **Question**: You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.
- **Answer**:

```
def coin_change(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
```

27. Decode Ways

- **Question**: A message containing letters from A-Z is being encoded to numbers using the following mapping: 'A' -> '1', 'B' -> '2', ..., 'Z' -> '26'. Given a non-empty string containing only digits, determine the total number of ways to decode it.
- **Answer**:

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

    n = len(s)
    dp = [0] * (n + 1)
    dp[0] = 1
    dp[1] = 0 if s[0] == '0' else 1

    for i in range(2, n + 1):
        first = int(s[i-1:i])
        second = int(s[i-2:i])
        if 1 <= first <= 9:
            dp[i] += dp[i-1]
        if 10 <= second <= 26:
            dp[i] += dp[i-2]

    return dp[n]
```

28. Container With Most Water

- **Question**: Given n non-negative integers `a1, a2, ..., an` , where each represents a point at coordinate `(i, ai)`. n vertical lines are drawn such that the two endpoints of the line `i` is at `(i, ai)` and `(i, 0)`. Find two lines, which, together with the x-axis forms a container, such that the container contains the most water.
- **Answer**:

```
def max_area(height):
    left, right = 0, len(height) - 1
    max_area = 0

    while left < right:
        h = min(height[left], height[right])
        max_area = max(max_area, h * (right - left))
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1

    return max_area
```

29. Fibonacci Series

- **Question**: Implement a function to compute the nth Fibonacci number.
- **Answer**:

```
def fibonacci(n):
    if n <= 1:
        return n
    a, b = 0, 1
    for _ in range(2, n + 1):
        a, b = b, a + b
    return b
```

30. Longest Increasing Subsequence

- **Question**: Given an unsorted array of integers, find the length of longest increasing subsequence.
- **Answer**:

```
def length_of_lis(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)
```

31. Check for Palindrome

- **Question**: Write a function that checks whether a given word or phrase is palindrome.
- **Answer**:

```
def is_palindrome(s):
    s = ''.join([c for c in s if c.isalnum()]).lower()
    return s == s[::-1]
```

32. Maximum Subarray

- **Question**: Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.
- **Answer**:

```
def max_sub_array(nums):
    if not nums:
        return 0
    cur_sum = max_sum = nums[0]
    for num in nums[1:]:
        cur_sum = max(num, cur_sum + num)
        max_sum = max(max_sum, cur_sum)
    return max_sum
```

33. Longest Common Prefix

- **Question**: Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".
- **Answer**:

```
def longest_common_prefix(strs):
    if not strs:
        return ""

    prefix = strs[0]
    for s in strs[1:]:
        i = 0
        while i < len(prefix) and i < len(s) and prefix[i] == s[i]:
            i += 1
        prefix = prefix[:i]
    return prefix
```

34. Check for Balanced Parentheses

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

```
def is_valid(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
```

35. Reverse a String

- **Question**: Implement a function that reverses a string.
- **Answer**:

```
def reverse_string(s):
    return s[::-1]
```

36. Intersection of Two Arrays II

- **Question**: Given two arrays, write a function to compute their intersection. Each element in the result should appear as many times as it shows in both arrays. The result can be in any order.
- **Answer**:

```
from collections import Counter

def intersect(nums1, nums2):
    c1, c2 = Counter(nums1), Counter(nums2)
    return list((c1 &amp; c2).elements())
```

37. Single Number

- **Question**: Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.
- **Answer**:

```
def single_number(nums):
    res = 0
    for num in nums:
        res ^= num
    return res
```

38. Rotate Array

- **Question**: Given an array, rotate the array to the right by k steps, where k is non-negative.
- **Answer**:

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

39. Product of Array Except Self

- **Question**: Given an array nums of n integers where n > 1, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].
- **Answer**:

```
def product_except_self(nums):
    length = len(nums)
    output = [1] * length
    left, right = 1, 1
    for i in range(length):
        output[i] *= left
        left *= nums[i]
        output[-1 - i] *= right
        right *= nums[-1 - i]

    return output
```

40. Implement `strStr()`

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

```
def str_str(haystack, needle):
    if not needle:
        return 0
    return haystack.find(needle)
```

41. First Unique Character in a String

- **Question**: Given a string, find the first non-repeating character in it and return its index. If it doesn't exist, return `-1`.
- **Answer**:

```
from collections import Counter

def first_uniq_char(s):
    count = Counter(s)
    for idx, ch in enumerate(s):
        if count[ch] == 1:
            return idx
    return -1
```

42. Two Sum

- **Question**: Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.
- **Answer**:

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

43. Count Primes

- **Question**: Count the number of prime numbers less than a non-negative number, n.
- **Answer**:

```
def count_primes(n):
    if n <= 2:
        return 0
    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 sum(primes)
```

44. Majority Element

- **Question**: Given an array of size n, find the majority element. The majority element is the element that appears more than `n/2` times.
- **Answer**:

```
from collections import Counter

def majority_element(nums):
    counts = Counter(nums)
    return max(counts.keys(), key=counts.get)
```

45. Valid Sudoku

- **Question**: Determine if a 9x9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

- Each row must contain the digits `1-9` without repetition.
- Each column must contain the digits `1-9` without repetition.
- Each of the 9 `3x3` sub-boxes of the grid must contain the digits `1-9` without repetition.
- **Answer**:

```
def is_valid_sudoku(board):
    rows = [set() for _ in range(9)]
    cols = [set() for _ in range(9)]
    boxes = [set() for _ in range(9)]

    for i in range(9):
        for j in range(9):
            val = board[i][j]
            if val == '.':
                continue

            if val in rows[i]:
                return False
            rows[i].add(val)

            if val in cols[j]:
                return False
            cols[j].add(val)

            box_idx = (i // 3) * 3 + j // 3
            if val in boxes[box_idx]:
                return False
            boxes[box_idx].add(val)

    return True
```

46. Move Zeroes

- **Question**: Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.
- **Answer**:

```
def move_zeroes(nums):
    pos = 0
    for i in range(len(nums)):
        if nums[i] != 0:
            nums[pos], nums[i] = nums[i], nums[pos]
            pos += 1
```

47. Best Time to Buy and Sell Stock

- **Question**: Say you have an array for which the `i`th element is the price of a given stock on day `i`. If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.
- **Answer**:

```
def max_profit(prices):
    if not prices:
        return 0

    min_price = prices[0]
    max_profit = 0

    for price in prices:
        if price < min_price:
            min_price = price
        else:
            max_profit = max(max_profit, price - min_price)

    return max_profit
```

48. Valid Anagram

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

```
from collections import Counter

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

49. Maximum Depth of Binary Tree

- **Question**: Given a binary tree, find its maximum depth. The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

- **Answer**:

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

def max_depth(root):
    if not root:
        return 0
    return 1 + max(max_depth(root.left), max_depth(root.right))
```

50. Palindrome Linked List

- **Question**: Given a singly linked list, determine if it is a palindrome.
- **Answer**:

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

def is_palindrome(head):
    vals = []
    current = head
    while current:
        vals.append(current.val)
        current = current.next
    return vals == vals[::-1]
```

These are just some examples, and the above solutions can be optimized and extended in various ways, depending on the exact constraints and requirements.

51. Merge Two Sorted Lists

- **Question**: Merge two sorted linked lists and return it as a new sorted list.
- **Answer**:

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

def merge_two_lists(l1, l2):
    dummy = ListNode(-1)
    prev = dummy

    while l1 and l2:
        if l1.val <= l2.val:
            prev.next = l1
            l1 = l1.next
        else:
            prev.next = l2
            l2 = l2.next
        prev = prev.next

    prev.next = l1 if l1 else l2
    return dummy.next
```

52. Implement Stack using Queues

- **Question**: Implement a stack using only instances of a queue.
- **Answer**:

```
from collections import deque

class MyStack:
    def __init__(self):
        self.q = deque()

    def push(self, x):
        self.q.append(x)
        for _ in range(len(self.q) - 1):
            self.q.append(self.q.popleft())

    def pop(self):
        return self.q.popleft()

    def top(self):
        return self.q[0]

    def empty(self):
        return not self.q
```

53. Min Stack

- **Question**: Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
- **Answer**:

```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, x):
        self.stack.append(x)
        if not self.min_stack or x <= self.min_stack[-1]:
            self.min_stack.append(x)

    def pop(self):
        if self.stack.pop() == self.min_stack[-1]:
            self.min_stack.pop()

    def top(self):
        return self.stack[-1]

    def getMin(self):
        return self.min_stack[-1]
```

54. Binary Tree Level Order Traversal

- **Question**: Given a binary tree, return the level order traversal of its nodes' values.
- **Answer**:

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

def level_order(root):
    if not root:
        return []
    result, queue = [], [root]
    while queue:
        level, level_len = [], len(queue)
        for i in range(level_len):
            node = queue.pop(0)
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.append(level)
    return result
```

55. Linked List Cycle

- **Question**: Given a linked list, determine if it has a cycle in it.
- **Answer**:

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

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
```

56. Invert Binary Tree

- **Question**: Invert 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 invert_tree(root):
    if not root:
        return None
    root.left, root.right = invert_tree(root.right), invert_tree(root.left)
    return root
```

57. Serialize and Deserialize Binary Tree

- **Question**: Design an algorithm to 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):
        def dfs(node):
            if not node:
                return ["None"]
            return [str(node.val)] + dfs(node.left) + dfs(node.right)
        return ",".join(dfs(root))

    def deserialize(self, data):
        data_list = data.split(',')
        def dfs():
            val = data_list.pop(0)
            if val == "None":
                return None
            node = TreeNode(int(val))
            node.left = dfs()
            node.right = dfs()
            return node
        return dfs()
```

58. Top K Frequent Elements

- **Question**: Given a non-empty array of integers, return the k most frequent elements.
- **Answer**:

```
from collections import Counter
import heapq

def top_k_frequent(nums, k):
    count = Counter(nums)
    return heapq.nlargest(k, count.keys(), key=count.get)
```

59. Decode String

- **Question**: Given an encoded string, return its decoded string. The encoding rule is: k[encoded_string], where the encoded_string inside the square brackets is being repeated exactly k times.
- **Answer**:

```
def decode_string(s):
    stack, curr_num, curr_str = [], 0, ''
    for char in s:
        if char == '[':
            stack.append(curr_str)
            stack.append(curr_num)
            curr_str, curr_num = '', 0
        elif char == ']':
            num = stack.pop()
            prev_str = stack.pop()
            curr_str = prev_str + num * curr_str
        elif char.isdigit():
            curr_num = curr_num * 10 + int(char)
        else:
            curr_str += char
    return curr_str
```

60. Number of Islands

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

```
def num_islands(grid):
    if not grid:
        return 0

    rows, cols = len(grid), len(grid[0])
    visited = [[0 for _ in range(cols)] for _ in range(rows)]
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    islands = 0

    def dfs(row, col):
        if row < 0 or col < 0 or row >= rows or col >= cols or visited[row]
[col] or grid[row][col] == '0':
            return
        visited[row][col] = 1
        for dr, dc in directions:
            dfs(row + dr, col + dc)

    for r in range(rows):
        for c in range(cols):
            if grid[r][c] == '1' and not visited[r][c]:
                dfs(r, c)
                islands += 1

    return islands
```

The explanations have been excluded for brevity, but it's essential to understand the logic behind each code snippet and be prepared to explain the solution's approach and its time and space complexity.

61. Maximum Depth of Binary Tree

- **Question**: Given a binary tree, find its maximum depth.
- **Answer**:

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

def max_depth(root):
    if not root:
        return 0
    left = max_depth(root.left)
    right = max_depth(root.right)
    return max(left, right) + 1
```

62. Two Sum

- **Question**: Given an array of integers, return indices of the two numbers such that they add up to a specific target.
- **Answer**:

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

63. Rotate Image

- **Question**: Given an image represented by an NxN matrix, rotate the image by 90 degrees.
- **Answer**:

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

64. Palindrome Linked List

- **Question**: Check if a singly linked list is a palindrome.
- **Answer**:

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

def is_palindrome(head):
    values = []
    current = head
    while current:
        values.append(current.val)
        current = current.next
    return values == values[::-1]
```

65. Product of Array Except Self

- **Question**: Given an array nums of n integers where n > 1, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].
- **Answer**:

```
def product_except_self(nums):
    n = len(nums)
    left, right, output = [1] * n, [1] * n, [1] * n

    for i in range(1, n):
        left[i] = nums[i - 1] * left[i - 1]

    for i in range(n - 2, -1, -1):
        right[i] = nums[i + 1] * right[i + 1]

    for i in range(n):
        output[i] = left[i] * right[i]

    return output
```

66. Find All Duplicates in an Array

- **Question**: Given an array of integers, where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once. Find all the elements that appear twice.
- **Answer**:

```
def find_duplicates(nums):
    result = []
    for num in nums:
        index = abs(num) - 1
        if nums[index] < 0:
            result.append(index + 1)
        nums[index] = -nums[index]
    return result
```

67. Coin Change

- **Question**: You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`.
- **Answer**:

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

68. House Robber

- **Question**: Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob without alerting the police (you cannot rob two adjacent houses).
- **Answer**:

```
def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    prev = nums[0]
    curr = max(nums[0], nums[1])
    for i in range(2, len(nums)):
        temp = curr
        curr = max(prev + nums[i], curr)
        prev = temp
    return curr
```

69. Longest Palindromic Substring

- **Question**: Given a string, find the longest substring which is a palindrome.
- **Answer**:

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

    def expand_center(left, right):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return s[left + 1:right]

    result = ""
    for i in range(len(s)):
        # Odd length palindrome
        palindrome1 = expand_center(i, i)
        # Even length palindrome
        palindrome2 = expand_center(i, i + 1)

        if len(palindrome1) > len(result):
            result = palindrome1
        if len(palindrome2) > len(result):
            result = palindrome2
    return result
```

70. Implement Trie (Prefix Tree)

- **Question**: Implement a trie with insert, search, and startsWith methods.
- **Answer**:

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

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

    def insert(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_word = True

    def search(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                return False
            node = node.children[char]
        return node.is_word

    def starts_with(self, prefix):
        node = self.root
        for char in prefix:
            if char not in node.children:
                return False
            node = node.children[char]
        return True
```

These solutions are just the code implementations. It's important to understand the logic, approach, and potentially the time and space complexities behind each one.