JAVA

CODING QUESTIONS

CODING QUESTIONS

1. **Two Sum: Given an array of integers, find two numbers that add up to a specific target.**
2. **Reverse a String**: Write a function to reverse a string without using built-in functions.
3. **Palindrome Check**: Determine if a given string is a palindrome.
4. **Merge Two Sorted Lists**: Merge two sorted linked lists and return it as a new sorted list.
5. **Longest Substring Without Repeating Characters:** Find the length of the longest substring without repeating characters.
6. **Valid Parentheses**: Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
7. **Search in Rotated Sorted Array**: Search for a target value in a rotated sorted array.
8. **Container With Most Water:** Given n non-negative integers, find two lines that together with the x-axis form a container, such that the container contains the most water.
9. **3Sum:** Find all unique triplets in the array which gives the sum of zero.
10. **Remove Nth Node From End of List**: Remove the n-th node from the end of a linked list and return its head.
11. **Maximum Subarray**: Find the contiguous subarray with the largest sum.
12. **Climbing Stairs**: You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?
13. **Set Matrix Zeroes:** Given a m x n matrix, if an element is 0, set its entire row and column to 0.
14. **Group Anagrams**: Given an array of strings, group anagrams together.
15. **Merge Intervals**: Given a collection of intervals, merge all overlapping intervals.
16. **Linked List Cycle:** Given a linked list, determine if it has a cycle in it.
17. **Implement Stack using Queues**: Implement a last-in-first-out (LIFO) stack using only two queues.
18. **Minimum Window Substring:** Given two strings s and t, find the minimum window in s which will contain all the characters in t.
19. **Word Search**: Given a 2D board and a word, find if the word exists in the grid.
20. **Longest Increasing Subsequence**: Find the length of the longest increasing subsequence in an array.
21. **Decode Ways:** A message containing letters from A-Z is encoded to numbers using 'A' -> 1, 'B' -> 2, ..., 'Z' -> 26. Given an encoded message, determine the total number of ways to decode it.
22. **Coin Change**: Given coins of different denominations and a total amount of money, find the fewest number of coins needed to make up that amount.
23. **House Robber**: Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

24. **Binary Tree Inorder Traversal**: Given a binary tree, return the inorder traversal of its nodes' values.
25. **Validate Binary Search Tree**: Determine if a given binary tree is a valid binary search tree.
26. **Lowest Common Ancestor of a Binary Tree**: Given a binary tree, find the lowest common ancestor of two given nodes in the tree.
27. **Serialize and Deserialize Binary Tree**: Design an algorithm to serialize and deserialize a binary tree.
28. **Kth Smallest Element in a BST**: Find the kth smallest element in a binary search tree.
29. **Number of Islands**: Given a 2D grid of '1's (land) and '0's (water), count the number of islands.
30. **Course Schedule**: There are a total of numCourses you have to take, labeled from 0 to numCourses-1. Some courses may have prerequisites. Determine if you can finish all courses.
31. **Implement Trie (Prefix Tree):** Implement a trie with insert, search, and startsWith methods.
32. **Add and Search Word - Data structure design**: Design a data structure that supports the addition of words and the search for a word in a dictionary.
33. **Word Ladder**: Given two words (beginWord and endWord), and a dictionary's word list, find the length of the shortest transformation sequence from beginWord to endWord.
34. **Find Median from Data Stream**: The median is the middle value in an ordered integer list. Write a program that finds the median of input data stream.
35. **Sliding Window Maximum**: Given an array and an integer k, find the maximum for each sliding window of size k.
36. **Longest Consecutive Sequence**: Given an unsorted array of integers, find the length of the longest consecutive elements sequence.
37. **Graph Valid Tree**: Given n nodes labeled from 0 to n-1 and a list of undirected edges, determine if these edges form a valid tree.
38. Number of Connected Components in an Undirected Graph

PROGRAMS

1. Two Sum

```java
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement)) {
            return new int[] { map.get(complement), i };
        }
        map.put(nums[i], i);
    }
    return new int[] {};
}
```

2. Reverse a String

```java
public String reverseString(String s) {
    return new StringBuilder(s).reverse().toString();
}
```

3. Palindrome Check

```java
public boolean isPalindrome(String s) {
    int left = 0, right = s.length() - 1;
    while (left < right) {
        if (s.charAt(left++) != s.charAt(right--)) return false;
    }
    return true;
}
```

4. Merge Two Sorted Lists

```java
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    if (l1 == null) return l2;
    if (l2 == null) return l1;
    if (l1.val < l2.val) {
        l1.next = mergeTwoLists(l1.next, l2);
        return l1;
    } else {
        l2.next = mergeTwoLists(l1, l2.next);
        return l2;
    }
}
```

5. Longest Substring Without Repeating Characters

```java
public int lengthOfLongestSubstring(String s) {
    Set<Character> set = new HashSet<>();
    int left = 0, maxLen = 0;
    for (int right = 0; right < s.length(); right++) {
        while (set.contains(s.charAt(right))) {
            set.remove(s.charAt(left++));
        }
        set.add(s.charAt(right));
        maxLen = Math.max(maxLen, right - left + 1);
    }
    return maxLen;
}
```

6. Valid Parentheses

```
public boolean isValid(String s) {
    Stack<Character> stack = new Stack<>();
    for (char c : s.toCharArray()) {
        if (c == '(' || c == '{' || c == '[') {
            stack.push(c);
        } else if (!stack.isEmpty() &&
                   ((c == ')' && stack.peek() == '(') ||
                    (c == '}' && stack.peek() == '{') ||
                    (c == ']' && stack.peek() == '['))) {
            stack.pop();
        } else {
            return false;
        }
    }
    return stack.isEmpty();
}
```

7. Search in Rotated Sorted Array

```
public int search(int[] nums, int target) {
    int left = 0, right = nums.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (nums[mid] == target) return mid;
        if (nums[left] <= nums[mid]) {
            if (nums[left] <= target && target < nums[mid]) right = mid -
1;
            else left = mid + 1;
        } else {
            if (nums[mid] < target && target <= nums[right]) left = mid +
1;
            else right = mid - 1;
        }
    }
    return -1;
}
```

8. Container With Most Water

```
public int maxArea(int[] height) {
    int left = 0, right = height.length - 1, max = 0;
    while (left < right) {
        max = Math.max(max, Math.min(height[left], height[right]) * (right
- left));
        if (height[left] < height[right]) left++;
        else right--;
    }
    return max;
}
```

9. 3Sum

```
public List<List<Integer>> threeSum(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> result = new ArrayList<>();
    for (int i = 0; i < nums.length - 2; i++) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;
        int left = i + 1, right = nums.length - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum == 0) {
                result.add(Arrays.asList(nums[i], nums[left++],
nums[right--]));
                while (left < right && nums[left] == nums[left - 1])
+;
                while (left < right && nums[right] == nums[right + 1]
right--;
            } else if (sum < 0) left++;
            else right--;
        }
    }
    return result;
}
```

10. Remove Nth Node From End of List

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode slow = dummy, fast = dummy;
    for (int i = 0; i <= n; i++) fast = fast.next;
    while (fast != null) {
        slow = slow.next;
        fast = fast.next;
    }
    slow.next = slow.next.next;
    return dummy.next;
}
```

11. Maximum Subarray

```
public int maxSubArray(int[] nums) {
    int max = nums[0], currentSum = nums[0];
    for (int i = 1; i < nums.length; i++) {
        currentSum = Math.max(nums[i], currentSum + nums[i]);
        max = Math.max(max, currentSum);
    }
    return max;
}
```

12. Climbing Stairs

```
public int climbStairs(int n) {
    if (n <= 2) return n;
    int first = 1, second = 2;
    for (int i = 3; i <= n; i++) {
        int third = first + second;
        first = second;
        second = third;
    }
    return second;
}
```

13. Set Matrix Zeroes

```
public void setZeroes(int[][] matrix) {
    boolean firstRow = false, firstCol = false;
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            if (matrix[i][j] == 0) {
                if (i == 0) firstRow = true;
                if (j == 0) firstCol = true;
                matrix[i][0] = 0;
                matrix[0][j] = 0;
            }
        }
    }
    for (int i = 1; i < matrix.length; i++) {
        for (int j = 1; j < matrix[0].length; j++) {
            if (matrix[i][0] == 0 || matrix[0][j] == 0) matrix[i][j] = 0;
        }
    }
    if (firstRow) Arrays.fill(matrix[0], 0);
    if (firstCol) for (int i = 0; i < matrix.length; i++) matrix[i][0] = 0;
}
```

14. Group Anagrams

```
public List<List<String>> groupAnagrams(String[] strs) {
    Map<String, List<String>> map = new HashMap<>();
    for (String s : strs) {
        char[] chars = s.toCharArray();
        Arrays.sort(chars);
        String key = new String(chars);
        map.putIfAbsent(key, new ArrayList<>());
        map.get(key).add(s);
    }
    return new ArrayList<>(map.values());
}
```

15. Merge Intervals

```
public int[][] merge(int[][] intervals) {
    Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
    List<int[]> merged = new ArrayList<>();
    for (int[] interval : intervals) {
        if (merged.isEmpty() || merged.get(merged.size() - 1)[1] <
interval[0]) {
            merged.add(interval);
        } else {
            merged.get(merged.size() - 1)[1] =
Math.max(merged.get(merged.size() - 1)[1], interval[1]);
        }
    }
    return merged.toArray(new int[merged.size()][]);
}
```

16. Linked List Cycle

```
public boolean hasCycle(ListNode head) {
    if (head == null || head.next == null) return false;
    ListNode slow = head, fast = head.next;
    while (slow != fast) {
        if (fast == null || fast.next == null) return false;
        slow = slow.next;
        fast = fast.next.next;
    }
    return true;
}
```

17. Implement Stack using Queues

```
class MyStack {
    Queue<Integer> queue = new LinkedList<>();

    public void push(int x) {
        queue.add(x);
        for (int i = 1; i < queue.size(); i++) {
            queue.add(queue.poll());
        }
    }

    public int pop() {
        return queue.poll();
    }

    public int top() {
        return queue.peek();
    }

    public boolean empty() {
        return queue.isEmpty();
    }
}
```

18. Minimum Window Substring

```
public String minWindow(String s, String t) {
    if (s.length() < t.length()) return "";
    Map<Character, Integer> map = new HashMap<>();
    for (char c : t.toCharArray()) map.put(c, map.getOrDefault(c, 0) + 1);
    int left = 0, count = 0, minLen = Integer.MAX_VALUE, start = 0;
    for (int right = 0; right < s.length(); right++) {
        char c = s.charAt(right);
        if (map.containsKey(c)) {
            map.put(c, map.get(c) - 1);
            if (map.get(c) >= 0) count++;
        }
        while (count == t.length()) {
            if (right - left + 1 < minLen) {
                minLen = right - left + 1;
                start = left;
            }
            char lc = s.charAt(left++);
            if (map.containsKey(lc)) {
                map.put(lc, map.get(lc) + 1);
                if (map.get(lc) > 0) count--;
            }
        }
    }
    return minLen == Integer.MAX_VALUE ? "" : s.substring(start, start +
minLen);
}
```

19. Word Search

```
public boolean exist(char[][] board, String word) {
    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[0].length; j++) {
            if (dfs(board, word, i, j, 0)) return true;
        }
    }
    return false;
}

private boolean dfs(char[][] board, String word, int i, int j, int index) {
    if (index == word.length()) return true;
    if (i < 0 || j < 0 || i >= board.length || j >= board[0].length ||
board[i][j] != word.charAt(index)) return false;
    char temp = board[i][j];
    board[i][j] = '#';
    boolean found = dfs(board, word, i + 1, j, index + 1) ||
                    dfs(board, word, i - 1, j, index + 1) ||
                    dfs(board, word, i, j + 1, index + 1) ||
                    dfs(board, word, i, j - 1, index + 1);
    board[i][j] = temp;
    return found;
}
```

29. Number of Islands

```
public int numIslands(char[][] grid) {
    int count = 0;
    for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[0].length; j++) {
            if (grid[i][j] == '1') {
                count++;
                dfs(grid, i, j);
            }
        }
    }
    return count;
}

private void dfs(char[][] grid, int i, int j) {
    if (i < 0 || i >= grid.length || j < 0 || j >= grid[0].length ||
grid[i][j] == '0') return;
    grid[i][j] = '0';
    dfs(grid, i + 1, j);
    dfs(grid, i - 1, j);
    dfs(grid, i, j + 1);
    dfs(grid, i, j - 1);
}
```

30. Course Schedule

```
public boolean canFinish(int numCourses, int[][] prerequisites) {
    List<List<Integer>> graph = new ArrayList<>();
    for (int i = 0; i < numCourses; i++) graph.add(new ArrayList<>());
    int[] inDegree = new int[numCourses];
    for (int[] prereq : prerequisites) {
        graph.get(prereq[1]).add(prereq[0]);
        inDegree[prereq[0]]++;
    }
    Queue<Integer> queue = new LinkedList<>();
    for (int i = 0; i < numCourses; i++) if (inDegree[i] == 0)
queue.add(i);
    int count = 0;
    while (!queue.isEmpty()) {
        int course = queue.poll();
        count++;
        for (int next : graph.get(course)) {
            if (--inDegree[next] == 0) queue.add(next);
        }
    }
    return count == numCourses;
}
```

31. Implement Trie (Prefix Tree)

```
class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (!node.containsKey(c)) node.put(c, new TrieNode());
            node = node.get(c);
        }
        node.setEnd();
    }

    public boolean search(String word) {
        TrieNode node = searchPrefix(word);
        return node != null && node.isEnd();
    }

    public boolean startsWith(String prefix) {
        return searchPrefix(prefix) != null;
    }

    private TrieNode searchPrefix(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.containsKey(c)) node = node.get(c);
```