GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

DATE: 20-11-2024 **NAME: JASHVARTHINI R – CSBS**

1. 3SUM CLOSEST

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return the sum of the three integers.

You may assume that each input would have exactly one solution.

Example 1:

Input: nums = [-1,2,1,-4], target = 1

Output: 2

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

Example 2:

Input: nums = [0,0,0], target = 1

Output: 0

Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

Constraints:

3 <= nums.length <= 500

-1000 <= nums[i] <= 1000

-104 <= target <= 104

PROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

public class ThreeSumClosest {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int closestSum = Integer.MAX_VALUE;
        int n = nums.length;

        for (int i = 0; i < n; i++) {
            int left = i + 1;
            int right = n - 1;
            while (left < right) {
                int currSum = nums[i] + nums[left] + nums[right];
```

```
                if (currSum == target)
                    return currSum;

                if (Math.abs(currSum - target) < Math.abs(closestSum - target))
                    closestSum = currSum;

                if (currSum > target)
                    --right;
                else
                    ++left;
            }
        }
        return closestSum;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();

        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }

        System.out.print("Enter the target value: ");
        int target = sc.nextInt();

        ThreeSumClosest solution = new ThreeSumClosest();
        int result = solution.threeSumClosest(nums, target);

        System.out.println("The closest sum to the target is: " + result);
    }
}
```

OUTPUT:

```
Enter the number of elements: 4
Enter the elements of the array:
-1
2
1
-4
Enter the target value: 1
The closest sum to the target is: 2
```

TIME COMPLEXITY: O(n^2)

2. **JUMP GAME II**

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

0 <= j <= nums[i] and

i + j < n

Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Example 1:

Input: nums = [2,3,1,1,4]

Output: 2

Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [2,3,0,1,4]

Output: 2

Constraints:

1 <= nums.length <= 104

0 <= nums[i] <= 1000

It's guaranteed that you can reach nums[n - 1].

PROGRAM:

```java
package dsaPracticeProblems;
import java.util.Scanner;

public class JumpGameII {
    public int jump(int[] nums) {
        int steps = 0;
        int lastJumpMaxReach = 0;
        int maxReach = 0;
        int n = nums.length;

        for (int i = 0; i < n - 1; i++) {
            maxReach = Math.max(maxReach, i + nums[i]);
            if (lastJumpMaxReach == i) {
                steps++;
                lastJumpMaxReach = maxReach;
                if (lastJumpMaxReach >= n - 1) {
                    break;
```

```
                }
            }
        }
        return steps;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = sc.nextInt();

        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }

        JumpGameII solution = new JumpGameII();
        int result = solution.jump(nums);

        System.out.println("Minimum number of jumps to reach the last index: " +
result);
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 5
Enter the elements of the array:
2
3
1
1
4
Minimum number of jumps to reach the last index: 2
```

TIME COMPLEXITY: O(n)

3. GROUP ANAGRAMS

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

Example 1:

Input: strs = ["eat","tea","tan","ate","nat","bat"]

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Explanation:

There is no string in strs that can be rearranged to form "bat".

The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.

The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

Example 2:

Input: strs = [""]

Output: [[""]]

Example 3:

Input: strs = ["a"]

Output: [["a"]]

Constraints:

1 <= strs.length <= 104

0 <= strs[i].length <= 100

strs[i] consists of lowercase English letters.

PROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

public class GroupAnagrams {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> anaMap = new HashMap<>();

        for (String word : strs) {
            char[] charArray = word.toCharArray();
            Arrays.sort(charArray);
            String sortedWord = new String(charArray);

            if (!anaMap.containsKey(sortedWord)) {
                anaMap.put(sortedWord, new ArrayList<>());
            }
            anaMap.get(sortedWord).add(word);
        }

        return new ArrayList<>(anaMap.values());
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of strings: ");
        int n = sc.nextInt();
        sc.nextLine();

        String[] strs = new String[n];
        System.out.println("Enter the strings:");
```

```
        for (int i = 0; i < n; i++) {
            strs[i] = sc.nextLine();
        }

        GroupAnagrams solution = new GroupAnagrams();
        List<List<String>> result = solution.groupAnagrams(strs);

        System.out.println("Grouped anagrams:" + result);

    }
}
```

OUTPUT:

```
Enter the number of strings: 6
Enter the strings:
eat
tea
tan
ate
nat
bat
Grouped anagrams:[[eat, tea, ate], [bat], [tan, nat]]
```

TIME COMPLEXITY: O(n. klogk)

4. DECODE WAYS

You have intercepted a secret message encoded as a string of numbers. The message is decoded via the following mapping:

"1" -> 'A'
"2" -> 'B'
...
"25" -> 'Y'
"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

"AAJF" with the grouping (1, 1, 10, 6)

"KJF" with the grouping (11, 10, 6)

The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the number of ways to decode it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

Input: s = "12"

Output: 2

Explanation:

"12" could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: s = "226"

Output: 3

Explanation:

"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Example 3:

Input: s = "06"

Output: 0

Explanation:

"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

Constraints:

1 <= s.length <= 100

s contains only digits and may contain leading zero(s).

PROGRAM:

```
package dsaPracticeProblems;
import java.util.Scanner;

class DecodeWays {
    public int numDecodings(String s) {
        final int n = s.length();
        if (n == 0 || s.charAt(0) == '0') return 0;
        int[] dp = new int[n + 1];
        dp[n] = 1;
        dp[n - 1] = isValid(s.charAt(n - 1)) ? 1 : 0;

        for (int i = n - 2; i >= 0; i--) {
```

```
            if (isValid(s.charAt(i))) {
                dp[i] += dp[i + 1];
            }
            if (i < n - 1 && isValid(s.charAt(i), s.charAt(i + 1))) {
                dp[i] += dp[i + 2];
            }
        }
        return dp[0];
    }

    private boolean isValid(char c) {
        return c != '0';
    }

    private boolean isValid(char c1, char c2) {
        return c1 == '1' || (c1 == '2' && c2 < '7');
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the encoded string: ");
        String s = sc.nextLine();
        DecodeWays solution = new DecodeWays();

        int result = solution.numDecodings(s);

        System.out.println("Number of ways to decode the string: " + result);
    }
}
```

OUTPUT:

```
Enter the encoded string: 226
Number of ways to decode the string: 3
```

TIME COMPLEXITY: O(n)

5. **BEST TIME TO BUY AND SELL STOCKS II**

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day.

Find and return the maximum profit you can achieve.

Example 1:

Input: prices = [7,1,5,3,6,4]

Output: 7

Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

Example 2:

Input: prices = [1,2,3,4,5]

Output: 4

Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Total profit is 4.

Example 3:

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Constraints:

1 <= prices.length <= 3 * 104

0 <= prices[i] <= 104

PROGRAM:

```
package dsaPracticeProblems;
import java.util.Scanner;

public class BestTimeToBuyAndSellStocksII {
    public int maxProfit(int[] prices) {
        int profit = 0;

        for (int i = 1; i < prices.length; i++) {
            if (prices[i] > prices[i - 1]) {
                profit += prices[i] - prices[i - 1];
            }
        }
        return profit;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of days: ");
        int n = sc.nextInt();

        int[] prices = new int[n];
        System.out.println("Enter the stock prices for each day:");
        for (int i = 0; i < n; i++) {
            prices[i] = sc.nextInt();
        }
```

```
        BestTimeToBuyAndSellStocksII calculator = new
BestTimeToBuyAndSellStocksII();
        int maxProfit = calculator.maxProfit(prices);

        System.out.println("The maximum profit is: " + maxProfit);
    }
}
```

OUTPUT:

```
Enter the number of days: 6
Enter the stock prices for each day:
7
1
5
3
6
4
The maximum profit is: 7
```

TIME COMPLEXITY: O(n)

6. NUMBER OF ISLANDS

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input: grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

Output: 1

Example 2:

Input: grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

Output: 3

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 300

grid[i][j] is '0' or '1'.

PROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

public class NumberofIslands {
    static boolean isSafe(char[][] grid, int row, int col,
                          boolean[][] visited) {
        int numRows = grid.length;
        int numCols = grid[0].length;

        return row >= 0 && row < numRows && col >= 0 && col < numCols
               && grid[row][col] == '1' && !visited[row][col];
    }

    static void DFS(char[][] grid, int row, int col, boolean[][] visited) {
        int[] rowNeighbours = { -1, -1, -1, 0, 0, 1, 1, 1 };
        int[] colNeighbours = { -1, 0, 1, -1, 1, -1, 0, 1 };

        visited[row][col] = true;

        for (int i = 0; i < 8; ++i) {
            int newRow = row + rowNeighbours[i];
            int newCol = col + colNeighbours[i];
            if (isSafe(grid, newRow, newCol, visited)) {
                DFS(grid, newRow, newCol, visited);
            }
        }
    }

    static int countIslands(char[][] grid) {
        int numRows = grid.length;
        int numCols = grid[0].length;
```

```
        boolean[][] visited = new boolean[numRows][numCols];

        int islandCount = 0;
        for (int row = 0; row < numRows; ++row) {
            for (int col = 0; col < numCols; ++col) {
                if (grid[row][col] == '1' && !visited[row][col]) {
                    DFS(grid, row, col, visited);
                    ++islandCount;
                }
            }
        }
        return islandCount;
    }

    public static void main(String[] args) {
        char[][] grid = {
            { '1', '1', '0', '0', '0' },
            { '0', '1', '0', '0', '1' },
            { '1', '0', '0', '1', '1' },
            { '0', '0', '0', '0', '0' },
            { '1', '0', '1', '1', '0' }
        };

        System.out.println("The Number of Islands is " + countIslands(grid));
    }
}
```

OUTPUT:

```
The Number of Islands is 4
```

TIME COMPLEXITY: O(row*column)

7. QUICK SORT

Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, arr[] in ascending order. Given an array, arr[], with starting index low and ending index high, complete the functions partition() and quickSort(). Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

Note: The low and high are inclusive.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.

Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]

Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.

Input: arr[] = [5, 5, 5, 5]

Output: [5, 5, 5, 5]
Explanation: All elements are identical, so the array remains unchanged.

Constraints:
1 <= arr.size() <= 103
1 <= arr[i] <= 104

PROGRAM:

```
package dsaPracticeProblems;
import java.util.Scanner;

public class QuickSort {
    static void quickSort(int arr[], int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);

            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        quickSort(arr, 0, n - 1);

        System.out.println("Sorted array:");
```

```
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter the elements of the array:
2 5 3 0 9
Sorted array:
0 2 3 5 9
```

TIME COMPLEXITY: O(n logn)

8. MERGE SORT

Given an array arr[], its starting position l and its ending position r. Sort the array using the merge sort algorithm.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Input: arr[] = [1, 3 , 2]

Output: [1, 2, 3]

Constraints:
1 <= arr.size() <= 105
1 <= arr[i] <= 105

PROGRAM:

```
package dsaPracticeProblems;
import java.util.Scanner;

public class MergeSort {
    public static void merge(int arr[], int l, int m, int r) {
        int n1 = m - l + 1;
        int n2 = r - m;

        int L[] = new int[n1];
        int R[] = new int[n2];

        for (int i = 0; i < n1; i++)
            L[i] = arr[l + i];
```

```
        for (int j = 0; j < n2; j++)
            R[j] = arr[m + 1 + j];

        int i = 0, j = 0;
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    public static void sort(int arr[], int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void printArray(int arr[]) {
        for (int num : arr)
            System.out.print(num + " ");
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        sort(arr, 0, n - 1);

        System.out.println("Sorted array:");
```

```
        printArray(arr);
    }
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter the elements of the array:
2
8
4
6
1
Sorted array:
1 2 4 6 8
```

TIME COMPLEXITY: O(n logn)

9. TERNARY SEARCH

Given a sorted array arr[] of size N and an integer K. The task is to check if K is present in the array or not using ternary search.

Ternary Search- It is a divide and conquer algorithm that can be used to find an element in an array. In this algorithm, we divide the given array into three parts and determine which has the key (searched element).

Example 1:

Input:

N = 5, K = 6

arr[] = {1,2,3,4,6}

Output: 1

Exlpanation: Since, 6 is present in

the array at index 4 (0-based indexing),

output is 1.

Example 2:

Input:

N = 5, K = 2

arr[] = {1,3,4,5,6}

Output: -1

Exlpanation: Since, 2 is not present

in the array, output is -1.

Your Task:
You don't need to read input or print anything. Complete the function ternarySearch() which takes the sorted array arr[], its size N and the element K as input parameters and returns 1 if K is present in the array, else it returns -1.

Expected Time Complexity: O(Log3N)
Expected Auxiliary Space: O(1)

Constraints:
1 < N < 106
1 < K < 106
1 < arr[i] < 106

PROGRAM:

```
package dsaPracticeProblems;

import java.util.Scanner;

public class TernarySearch {
	public static int ternarySearch(int[] arr, int low, int high, int x) {
		while(high>=low) {
			int mid1 = low + (high - low)/3;
			int mid2 = high - (high - low)/3;

			if(arr[mid1]==x) {
				return mid1;
			}

			if(arr[mid2]==x)
				return mid2;

			if(x<=arr[mid1]) {
				return ternarySearch(arr, low, mid1-1, x);
			}

			else if(x>arr[mid2]) {
				return ternarySearch(arr, mid1+1, high, x);
			}

			else {
				return ternarySearch(arr, mid1+1, mid2-1, x);
			}
		}
		return -1;
	}
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
```

```
		System.out.println("Enter the number of elements: ");
		int n = scanner.nextInt();

		int[] arr = new int[n];
		System.out.println("Enter the elements in the array: ");
		for(int i = 0; i<n ; i++) {
			arr[i] = scanner.nextInt();
		}

		System.out.println("Enter the element to be searched: ");
		int x = scanner.nextInt();

		int low = 0;
		int high = n-1;

		int ans = ternarySearch(arr, low, high, x);

		if(ans!=-1) {
			System.out.println("The Element is found at "+ ans);
		}

		else {
			System.out.println("The Element is not found");
		}

		scanner.close();
	}
}
```

OUTPUT:

```
Enter the number of elements:
5
Enter the elements in the array:
5 6 7 8 9
Enter the element to be searched:
7
The Element is found at 2
```

TIME COMPLEXITY: O(log n)

10. INTERPOLATION SEARCH

PROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

public class InterpolationSearch {
	public static int interpolationSearch(int[] arr, int low, int high, int x)
{

		if(low<=high && x>arr[low] && x<=arr[high]) {
			int pos;
			pos = low + (((high-low)*(x-arr[low]))/(arr[high]-arr[low]));
```

```
                if(arr[pos]==x) {
                        return pos;
                }

                if(arr[pos]<x) {
                        return interpolationSearch(arr, pos+1, high, x);
                }

                else
                return interpolationSearch(arr, low, pos-1, x);
        }

        return -1;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements in the array: ");
        for(int i = 0; i<n ; i++) {
                arr[i] = scanner.nextInt();
        }

        System.out.println("Enter the element to be searched: ");
        int x = scanner.nextInt();

        int low = 0;
        int high = n-1;

        int ans = interpolationSearch(arr, low, high, x);

        if(ans!=-1) {
                System.out.println("The Element is found at "+ ans);
        }

        else {
                System.out.println("The Element is not found");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements:
5
Enter the elements in the array:
1
2
9
5
6
Enter the element to be searched:
5
The Element is found at 3
```

TIME COMPLEXITY: O(log(log n))