DSA PRACTICE -8

1. 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 <= 10^4
- 0 <= nums[i] <= 1000
- It's guaranteed that you can reach nums[n - 1]

Solution:

```
import java.util.Scanner;

class Jump {
    public int jump(int[] nums) {
        int near = 0, far = 0, jumps = 0;
        while (far < nums.length - 1) {
            int farthest = 0;
            for (int i = near; i <= far; i++) {
                farthest = Math.max(farthest, i + nums[i]);
```

```
            }
            near = far + 1;
            far = farthest;
            jumps++;
        }
        return jumps;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }
        Jump solution = new Jump();
        int result = solution.jump(nums);
        System.out.println("Minimum number of jumps: " + result);
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac Jump.java

F:\DSA PRACTICE\day -8>java Jump
Enter the number of elements in the array:
4
Enter the elements of the array:
1 3 5 4
Minimum number of jumps: 2
```

TimeComplexity:O(n)

SpaceComplexity:O(1)

2. 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:

- $1 <= strs.length <= 10^4$
- 0 <= strs[i].length <= 100
- strs[i] consists of lowercase English letters.

Solution:

```
import java.util.*;
class Anagram {
   public List<List<String>> groupAnagrams(String[] strs) {
      Map<String, List<String>> map = new HashMap<>();
```

```
        for (String word : strs) {
            char[] chars = word.toCharArray();
            Arrays.sort(chars);
            String sortedWord = new String(chars);
            if (!map.containsKey(sortedWord)) {
                map.put(sortedWord, new ArrayList<>());
            }
            map.get(sortedWord).add(word);
        }
        return new ArrayList<>(map.values());
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of strings:");
        int n = scanner.nextInt();
        scanner.nextLine();
        String[] strs = new String[n];
        System.out.println("Enter the strings:");
        for (int i = 0; i < n; i++) {
            strs[i] = scanner.nextLine();
        }
        Anagram solution = new Anagram();
        List<List<String>> result = solution.groupAnagrams(strs);
        System.out.println("Grouped Anagrams:");
        for (List<String> group : result) {
            System.out.println(group);
        }
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac Anagram.java

F:\DSA PRACTICE\day -8>java Anagram
Enter the number of strings:
4
Enter the strings:
ate word cat dog
cat copy tam lad
air ure rye rwe
rieuinrei erj rieru reu
Grouped Anagrams:
[cat copy tam lad]
[ate word cat dog]
[rieuinrei erj rieru reu]
[air ure rye rwe ]
```

Time complexity : O(n * k log k)

Space complexity:O(n * k)

3. 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
- -10^4 <= target <= 10^4

Solution:

```
import java.util.Scanner;
class 3SumClosest{
    public int threeSumClosest(int[] nums, int target) {
        int closest = nums[0] + nums[1] + nums[2];
        for (int i = 0; i < nums.length - 2; i++) {
            for (int j = i + 1; j < nums.length - 1; j++) {
                for (int k = j + 1; k < nums.length; k++) {
                    int currentSum = nums[i] + nums[j] + nums[k];
                    if (Math.abs(target - currentSum) < Math.abs(target - closest)) {
                        closest = currentSum;
                    }
                }
            }
        }
        return closest;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }
```

```
        System.out.println("Enter the target value:");
        int target = scanner.nextInt();
        3SumClosest solution = new 3SumClosest ();
        int result = solution.threeSumClosest(nums, target);
        System.out.println("Closest sum to the target: " + result);
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac SumClosest.java

F:\DSA PRACTICE\day -8>java SumClosest
Enter the number of elements in the array:
4
Enter the elements of the array:
2 4 5 6
Enter the target value:
11
Closest sum to the target: 11

F:\DSA PRACTICE\day -8>
```

TimeComplexity: O(n^3)

SpaceComplexity: **O(1)**.

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).

Solution:

```
import java.util.Scanner;

public class DecodeWays {
    public int numDecodings(String s) {
        if (s == null || s.length() == 0 || s.charAt(0) == '0') {
            return 0;
        }
        int n = s.length();
        int[] dp = new int[n + 1];
        dp[0] = 1;
        dp[1] = s.charAt(0) != '0' ? 1 : 0;
        for (int i = 2; i <= n; i++) {
            if (s.charAt(i - 1) != '0') {
                dp[i] += dp[i - 1];
            }
            int twoDigit = Integer.parseInt(s.substring(i - 2, i));
            if (twoDigit >= 10 && twoDigit <= 26) {
                dp[i] += dp[i - 2];
            }
```

```
        }
        return dp[n];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the encoded message 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);
        sc.close();
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>java DecodeWays
Enter the encoded message string: 13
Number of ways to decode the string: 2
```

Time Complexity : O(n)

Space Complexity : O(n)

5. . Best Time to Buy and Sell Stock II:

You are given an integer array prices where prices[i] is the price of a given stock on the i^{th} 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*.

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.

Solution:

```
import java.util.Scanner;
class BestTimeToBuyAndSellStockTwo {
    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 scanner = new Scanner(System.in);
        System.out.print("Enter the number of days: ");
        int n = scanner.nextInt();
        int[] prices = new int[n];
        System.out.println("Enter the stock prices:");
        for (int i = 0; i < n; i++) {
            prices[i] = scanner.nextInt();
        }
        BestTimeToBuyAndSellStockTwo sol = new BestTimeToBuyAndSellStockTwo();
        int result = sol.maxProfit(prices);
        System.out.println("Maximum profit: " + result);
        scanner.close();
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac  BestTimeToBuyAndSellStockTwo.java

F:\DSA PRACTICE\day -8>java  BestTimeToBuyAndSellStockTwo
Enter the number of days: 5
Enter the stock prices:
3 4 7 2 5
Maximum profit: 7
```

Time Complexity : O(n)

Space 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

Solution:

```
import java.util.Scanner;

class NumberIslands {

    public int numIslands(char[][] grid) {

        if (grid == null || grid.length == 0) {

            return 0;

        }

        int c = 0;

        for (int i = 0; i < grid.length; i++) {

            for (int j = 0; j < grid[0].length; j++) {
```

```
            if (grid[i][j] == '1') {
                c++;
                dfs(grid, i, j);
            }
        }
    }
    return c;
}
private void dfs(char[][] grid, int i, int j) {
    if (i < 0 || j < 0 || i >= grid.length || 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);
}
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of rows:");
    int rows = sc.nextInt();
    System.out.println("Enter the number of columns:");
    int cols = sc.nextInt();
    System.out.println("Enter the grid (0 for water, 1 for land):");
    char[][] grid = new char[rows][cols];
    sc.nextLine();
    for (int i = 0; i < rows; i++) {
        String row = sc.nextLine();
```

```
            grid[i] = row.toCharArray();
        }
        NumberIslands solution = new NumberIslands();
        int numberOfIslands = solution.numIslands(grid);
        System.out.println("Number of islands: " + numberOfIslands);
        sc.close();
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac NumberIslands.java

F:\DSA PRACTICE\day -8>java NumberIslands
Enter the number of rows:
3
Enter the number of columns:
3
Enter the grid (0 for water, 1 for land):
1 0 1
1 1 1
0 0 0
Number of islands: 1
```

Time Complexity : O(m*n)

Space Complexity : O(m*n)

7.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]

Solution:

```
import java.io.*;
import java.lang.*;
import java.util.*;
class MergeSort {
```

```
    public static void main(String args[]) throws IOException {
        BufferedReader read = new BufferedReader(new InputStreamReader(System.in));
        int t = Integer.parseInt(read.readLine());
        while (t-- > 0) {
            ArrayList<Integer> array1 = new ArrayList<Integer>();
            String line = read.readLine();
            String[] tokens = line.split(" ");
            for (String token : tokens) {
                array1.add(Integer.parseInt(token));
            }
            ArrayList<Integer> v = new ArrayList<Integer>();
            int[] arr = new int[array1.size()];
            int idx = 0;
            for (int i : array1) arr[idx++] = i;
            new Solution().mergeSort(arr, 0, arr.length - 1);
            for (int i = 0; i < arr.length; i++) System.out.print(arr[i] + " ");
            System.out.println();
            System.out.println("~");
        }
    }
}
class Solution {
    public void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
```

```
    }
    private void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];
        for (int i = 0; i < n1; i++)
            leftArray[i] = arr[left + i];
        for (int i = 0; i < n2; i++)
            rightArray[i] = arr[mid + 1 + i];
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                arr[k++] = leftArray[i++];
            } else {
                arr[k++] = rightArray[j++];
            }
        }
        while (i < n1) {
            arr[k++] = leftArray[i++];
        }
        while (j < n2) {
            arr[k++] = rightArray[j++];
        }
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>java MergeSort
5
1 4 6 7 4
1 4 4 6 7
```

Time Complexity : O(n log n)

Space Complexity : O(n)

8.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.

Solution:

```
import java.io.*;

import java.util.*;

class QuickSort {

    static void printArray(int arr[]) {

        int n = arr.length;

        for (int i = 0; i < n; ++i) System.out.print(arr[i] + " ");

        System.out.println();

    }

    public static void main(String args[]) throws IOException {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        int t = Integer.parseInt(br.readLine());

        while (t-- > 0) {

            String inputLine[] = br.readLine().trim().split(" ");
```

```
        int n = inputLine.length;
        int arr[] = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = Integer.parseInt(inputLine[i]);
        }
        new Solution().quickSort(arr, 0, n - 1);
        printArray(arr);
    }
  }
}
class Solution {
  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;
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac QuickSort.java

F:\DSA PRACTICE\day -8>java QuickSort
5
5 3 6 3 1
1 3 3 5 6
```

Time Complexity : O(n^2)

Space Complexity : O(log n)

9.Searching an element in a sorted array (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

Explanation: Since, 6 is present in

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

output is 1.

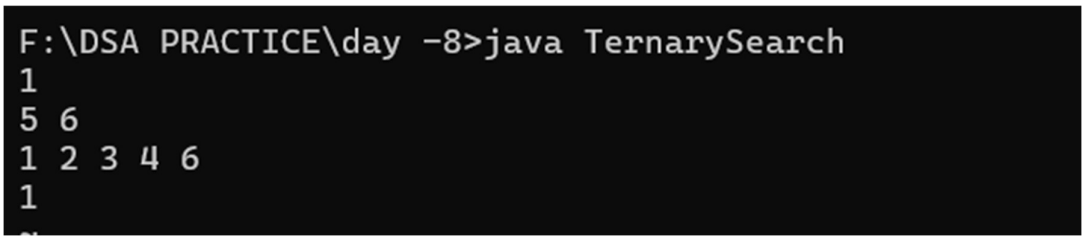
Solution:

```
import java.util.*;
import java.lang.*;
```

```
import java.io.*;
class TernarySearch{
   public static void main(String args[])throws IOException{
      BufferedReader read = new BufferedReader(new InputStreamReader(System.in));
      int t = Integer.parseInt(read.readLine());
      while(t-- > 0){
         String s[] = read.readLine().trim().split("\\s+");
         int N = Integer.parseInt(s[0]);
         int K = Integer.parseInt(s[1]);
         int arr[] = new int[N];
         String st[] = read.readLine().trim().split("\\s+");
         for(int i = 0; i < N; i++){
            arr[i] = Integer.parseInt(st[i]);
         }
         Solution obj = new Solution();
         System.out.println(obj.ternarySearch(arr, N, K));
System.out.println("~");
}
   }
}
class Solution{
   public static int ternarySearch(int arr[], int N, int K) {
      int left = 0, right = N - 1;
      while (left <= right) {
         int mid1 = left + (right - left) / 3;
         int mid2 = right - (right - left) / 3;
         if (arr[mid1] == K) {
            return 1;
         }
```

```
            if (arr[mid2] == K) {
                return 1;
            }
            if (K < arr[mid1]) {
                right = mid1 - 1;
            } else if (K > arr[mid2]) {
                left = mid2 + 1;
            } else {
                left = mid1 + 1;
                right = mid2 - 1;
            }
        }
        return -1;
    }
}
```

Output:



10. Interpolation Search

Solution:

```
import java.util.*;
class InterpolationSearch{
        public static int interpolationSearch(int arr[], int lo, int hi, int x){
                int pos;
                if (lo <= hi && x >= arr[lo] && x <= arr[hi]) {
                        pos = lo+ (((hi - lo) / (arr[hi] - arr[lo]))* (x - arr[lo]));
```

```
                if (arr[pos] == x)
                        return pos;
                if (arr[pos] < x)
                        return interpolationSearch(arr, pos + 1, hi,x);
                if (arr[pos] > x)
                        return interpolationSearch(arr, lo, pos - 1,x);
        }
        return -1;
    }
    public static void main(String[] args)
    {
        int arr[] = { 10, 12, 13, 16, 18, 19, 20, 21,22, 23, 24, 33, 35, 42, 47 };
        int n = arr.length;
        int x = 18;
        int index = interpolationSearch(arr, 0, n - 1, x);
        if (index != -1)
            System.out.println("Element found at index "+ index);
        else
            System.out.println("Element not found.");
    }
}
```

Output:

```
F:\DSA PRACTICE\day -8>javac InterpolationSearch.java

F:\DSA PRACTICE\day -8>java InterpolationSearch
Element found at index 4
```

Time Complexity : O(n)

Space Complexity : O(1)