DSA TRAINING DAY4(13.11.24)

Name: Sundara Vinayagam V

Dept: B.TECH-CSBS 3rd Year

1.Kth Smallest Element:

Given an array **arr[]** and an integer **k** where k is smaller than the size of the array, the task is to find the **k**th **smallest** element in the given array.

Follow up: Don't solve it using the inbuilt sort function.

Examples :

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3

Output: 7

Explanation: 3rd smallest element in the given array is 7.

Input: arr[] = [2, 3, 1, 20, 15], k = 4

Output: 15

Explanation: 4th smallest element in the given array is 15.

Constraints:
1 <= arr.size <= 10^6
1<= arr[i] <= 10^6
1 <= k <= n

Solution:

```
import java.util.*;

public class KthSmallestElement {
    public static int kthSmallest(int[] arr, int k) {
        int maxElement = Arrays.stream(arr).max().getAsInt();
        int[] freq = new int[maxElement + 1];

        for (int num : arr) {
            freq[num]++;
        }

        int count = 0;
        for (int i = 0; i <= maxElement; i++) {
            if (freq[i] > 0) {
                count += freq[i];
                if (count >= k) {
                    return i;
                }
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        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();
        }

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

        int result = kthSmallest(arr, k);
        if (result != -1) {
            System.out.println("The " + k + "th smallest element is: " + result);
        } else {
            System.out.println("Invalid input.");
        }
    }
}
```

Time Complexity: O(n+(max_element))

Space Complexity: O(max_element)

2. **Minimize the Heights II**

Given an array **arr[]** denoting heights of **N** towers and a positive integer **K.**

For **each** tower, you must perform **exactly one** of the following operations **exactly once**.

- **Increase** the height of the tower by **K**
- **Decrease** the height of the tower by **K**

Find out the **minimum** possible difference between the height of the shortest and tallest towers after you have modified each tower.
Note: It is **compulsory** to increase or decrease the height by K for each tower. **After** the operation, the resultant array should **not** contain any **negative integers**.

Examples :

Input: k = 2, arr[] = {1, 5, 8, 10}

Output: 5

Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

Input: k = 3, arr[] = {3, 9, 12, 16, 20}

Output: 11

Explanation: The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.

Solution:

```
import java.util.*;

public class Solution {
    public static int getMinDiff(int[] arr, int n, int k) {
        Arrays.sort(arr);
        int minDiff = arr[n - 1] - arr[0];
        int smallest = arr[0] + k;
        int largest = arr[n - 1] - k;

        for (int i = 0; i < n - 1; i++) {
            int minVal = Math.min(smallest, arr[i + 1] - k);
            int maxVal = Math.max(largest, arr[i] + k);
            if (minVal < 0) continue;
            minDiff = Math.min(minDiff, maxVal - minVal);
        }
return minDiff;
    }
 public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of towers (n): ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the heights of the towers:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter the value of K: ");
        int k = sc.nextInt();

        int result = getMinDiff(arr, n, k);
        System.out.println("The minimum possible difference is: " + result);
    }
}
```

Time Complexity: O(n*logn)
Space Complexity: O(n)

3.Parenthesis Checker
You are given a string **s** representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.
Examples :
Input: s = "{([])}"
Output: true
Explanation:
- In this expression, every opening bracket has a corresponding closing bracket.
- The first bracket { is closed by }, the second opening bracket (is closed by), and the third opening bracket [is closed by].
- As all brackets are properly paired and closed in the correct order, the expression is considered balanced.
Input: s = "()"
Output: true
Explanation:
- This expression contains only one type of bracket, the parentheses (and).

- The opening bracket (is matched with its corresponding closing bracket).
- Since they form a complete pair, the expression is balanced.

Input: s = "([]"
Output: false
Explanation:
- This expression contains only one type of bracket, the parentheses (and).
- The opening bracket (is matched with its corresponding closing bracket).
- Since they form a complete pair, the expression is balanced.

Solution:

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

class Paranthesis{
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        while (t-- > 0) {
           String st = sc.next();
            if (new Solution().isParenthesisBalanced(st) == true)
                System.out.println("true");
            else
                System.out.println("false");

            System.out.println("~");
        }
    }
}
class Solution {
    static boolean isParenthesisBalanced(String s) {
     Stack<Character> stack=new Stack<Character>();
      for (int i = 0; i < s.length(); i++) {
            char current = s.charAt(i);
            if (current == '(' || current == '{' || current == '[') {
                stack.push(current);
            }
            else if (current == ')' || current == '}' || current == ']') {
                if (stack.isEmpty()) {
                    return false;
                }
             char top = stack.pop();
                if ((current == ')' && top != '(') ||
                    (current == '}' && top != '{') ||
                    (current == ']' && top != '[')) {
                    return false;
                }
         }

     }
    return stack.empty();
     }
}
```

Time Complexity:O(n)
Space Complexity:O(n)

4.Equilibrium Point
Given an array **arr** of non-negative numbers. The task is to find the first **equilibrium point** in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements before that index is the same as the sum of elements after it.
Note: Return equilibrium point in 1-based indexing. Return -1 if no such point exists.
Examples:
Input: arr[] = [1, 3, 5, 2, 2]
Output: 3
Explanation: The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).
Input: arr[] = [1]
Output: 1
Explanation: Since there's only one element hence it's only the equilibrium point.

Input: arr[] = [1, 2, 3]
Output: -1
Explanation: There is no equilibrium point in the given array.
Expected Time Complexity: O(n)
Expected Auxiliary Space: O(1)
Constraints:
1 <= arr.size <= 10^6
0 <= arr[i] <= 10^9

Solution:
```
import java.io.*;
import java.util.*;
class Main {
  public static void main(String[] args) throws IOException {
       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
       int t = Integer.parseInt(br.readLine().trim());
       while (t-- > 0) {
            String line = br.readLine();
          String[] tokens = line.split(" ");
          ArrayList<Integer> array = new ArrayList<>();
         for (String token : tokens) {
              array.add(Integer.parseInt(
                  token));
          }
            int[] arr = new int[array.size()];
          int idx = 0;
          for (int i : array) arr[idx++] = i;
            Solution obj = new Solution();
          System.out.println(obj.equilibriumPoint(arr));
          System.out.println("~");
       }
    }
}
class Solution {
    public static int equilibriumPoint(int arr[]) {
       int rs=0;
       for(int i:arr) rs+=i;
       int ls=0;
       for(int i=0;i<arr.length;i++){
          if(ls==(rs-ls-arr[i])) return i+1;
          ls+=arr[i];
       }
       return -1;
    }
}
```
Time Complexity: O(n)
Space Complexity: O(1)

5.Binary Search
Given a sorted array **arr** and an integer **k**, find the position(0-based indexing) at which k is present in the array using binary search.
Note: If multiple occurrences are there, please return the smallest index.
Examples:
Input: arr[] = [1, 2, 3, 4, 5], k = 4
Output: 3
Explanation: 4 appears at index 3.
Input: arr[] = [11, 22, 33, 44, 55], k = 445
Output: -1
Explanation: 445 is not present.

Solution:
```
class Solution {
    public int binarysearch(int[] arr, int k) {
        // Code Here
        int n=arr.length;
        int left=0, mid, right=n-1;
       while(left<=right)
        {
           mid=(left+right)/2;
           if(arr[mid]>k)
           right=mid-1;
           if(arr[mid]<k)
           left=mid+1;
           if(arr[mid]==k)
           return mid;
        }

        return -1;

    }
}
```
Time complexity: O(n logn)

Next Greater Element
Given an array **arr[ ]** of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.
Examples

Input: arr[] = [1, 3, 2, 4]
Output: [3, 4, 4, -1]
Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.
Input: arr[] = [6, 8, 0, 1, 3]
Output: [8, -1, 1, 3, -1]
Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.
Input: arr[] = [10, 20, 30, 50]
Output: [20, 30, 50, -1]
Explanation: For a sorted array, the next element is next greater element also exxept for the last element.
Input: arr[] = [50, 40, 30, 10]
Output: [-1, -1, -1, -1]
Explanation: There is no greater element for any of the elements in the array, so all are -1.
Constraints:
$1 \leq arr.size() \leq 10^6$
$0 \leq arr[i] \leq 10^9$

Solution:
```
class Solution {
   public ArrayList<Integer> nextLargerElement(int[] arr) {
        int n=arr.length;
        ArrayList<Integer>result=new ArrayList<>(n);
        for(int i=0;i<n;i++){
           result.add(-1);
        }

        for(int i=0;i<n;i++){
           for(int j=i+1;j<n;j++){
              if(arr[j]>arr[i]){
                 result.set(i,arr[j]);
                 break;
              }
           }
        }
        return result;
    }
}
```
Time Complextiy: O(n)

6.Next Greater Element

Given an array **arr[ ]** of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Examples

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

Output: [3, 4, 4, -1]

Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

Input: arr[] = [6, 8, 0, 1, 3]

Output: [8, -1, 1, 3, -1]

Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Input: arr[] = [10, 20, 30, 50]

Output: [20, 30, 50, -1]

Explanation: For a sorted array, the next element is next greater element also exxept for the last element.

Input: arr[] = [50, 40, 30, 10]

Output: [-1, -1, -1, -1]

Explanation: There is no greater element for any of the elements in the array, so all are -1.

Constraints:
$1 \le arr.size() \le 10^6$
$0 \le arr[i] \le 10^9$

Solution:

```
class Solution {
    public ArrayList<Integer> nextLargerElement(int[] arr) {
        int n=arr.length;
        ArrayList<Integer>result=new ArrayList<>(n);
        for(int i=0;i<n;i++){
            result.add(-1);
        }
for(int i=0;i<n;i++){
            for(int j=i+1;j<n;j++){
                if(arr[j]>arr[i]){
                    result.set(i,arr[j]);
                    break;
                }
            }
        }
        return result;    }
}
```

Time Complexity:O(n)

Space Complexity:O(n)

7.Union of Two Arrays with Duplicate Elements

Given two arrays **a[]** and **b[]**, the task is to find the number of elements in the union between these two arrays.

The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union.
Note: Elements are not necessarily distinct.
Examples
Input: a[] = [1, 2, 3, 4, 5], b[] = [1, 2, 3]
Output: 5
Explanation: 1, 2, 3, 4 and 5 are the elements which comes in the union setof both arrays. So count is 5.
Input: a[] = [85, 25, 1, 32, 54, 6], b[] = [85, 2]
Output: 7
Explanation: 85, 25, 1, 32, 54, 6, and 2 are the elements which comes in the union set of both arrays. So count is 7.
Input: a[] = [1, 2, 1, 1, 2], b[] = [2, 2, 1, 2, 1]
Output: 2
Explanation: We need to consider only distinct. So count is 2.
Constraints:
1 ≤ a.size(), b.size() ≤ 10^6
0 ≤ a[i], b[i] < 10^5

Solution:

```
class Solution {
   public static int findUnion(int a[], int b[]) {
      Set<Integer> set = new HashSet<>();

      for(int i=0;i<a.length;i++) set.add(a[i]);
      for(int i=0;i<b.length;i++) set.add(b[i]);

      return set.size();
   }
}
```
Time Complexity:O(n)s