DSA Problems SET - 3

14/11/2024 **JANET SHINY V**
22CZ021

1)Given an array arr[] and an integer k where k is smaller than the size of the array, the task is to find the kth 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.

Program :

```
class Solution {
    public static int kthSmallest(int[] arr, int k) {
        int m_e=arr[0];
        for(int i=0;i<arr.length;i++){
            if(arr[i]>m_e) m_e=arr[i];
        }
        int a[]=new int[m_e+1];
        Arrays.fill(a,0);
        for(int i=0;i<arr.length;i++){
            a[arr[i]]++;
        }
        int c=0;
        for(int i=0;i<m_e+1;i++){
```

```
            if(a[i]!=0){
                c+=a[i];
            if(c>=k) return i;
            }

        }
        return -1;
    }

}
```

Output

The 2th smallest element is 5

Time Complexity: O(N + max_element)

Space Complexity: O(max_element)

2) 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.

You can find a slight modification of the problem here.
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.

Program :

```
class Solution {
    public int getMinDiff(int k, int[] arr) {
        int n = arr.length;
        if (n == 1) {
            return 0;
        }

        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 currentMax = Math.max(largest, arr[i] + k);
            int currentMin = Math.min(smallest, arr[i + 1] - k);

            minDiff = Math.min(minDiff, currentMax - currentMin);
        }

        return minDiff;
    }
}
```

Output:

5

Time Complexity: O(n*logn)

Space Complexity: O(n)

3)Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([])"

Output: true

Program :

```
import java.util.Scanner;

import java.util.Stack;

class One {

public static String isBalanced(String str) {

Stack<Character> stack = new Stack<>();

for (char ch : str.toCharArray()) {

if (ch == '(') {

stack.push(ch);

}

else if (ch == ')') {

if (stack.isEmpty()) {
```

```
return "Not Balanced";

}

stack.pop();  }

}

return stack.isEmpty() ? "Balanced" : "Not Balanced";}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string of parentheses: ");

String str = scanner.nextLine();

System.out.println("Output: " + isBalanced(str));}}
```

OUTPUT:

Enter a string of parentheses: ()()()

Output: Balanced

TIMECOMPLEXITY:O(N)

SPACECOMPLEXITY:O(N)

4)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.

Program :

```
class Solution {

    public static int equilibriumPoint(int arr[]) {

        int rsum = 0;

        for(int i : arr) rsum += i;

        int lsum = 0;

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

            if (lsum == (rsum - lsum - arr[i])) return i + 1;

            lsum += arr[i];

        }

        return -1;

    }



    public static void main(String[] args) {

        int arr[] = {1, 3, 5, 2, 2};

        int result = equilibriumPoint(arr);

        System.out.println("Equilibrium Point Index (1-based): " + result);

    }

}
```

Output :

Equilibrium Point Index (1-based): 3

Time Complexity : O(n)

Space Complexity : O(1)

5)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.

Program :

```
class Solution {

    public static int binarySearch(int arr[], int k) {
```

```
    int left = 0;
    int right = arr.length - 1;
    int result = -1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == k) {
            result = mid;
            right = mid - 1;
        } else if (arr[mid] < k) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return result;     }
public static void main(String[] args) {
    int arr1[] = {1, 2, 3, 4, 5};
    int k1 = 4;
```

```
        int result1 = binarySearch(arr1, k1);
        System.out.println("Position of " + k1 + " in array: " + result1);
        int arr2[] = {11, 22, 33, 44, 55};
        int k2 = 445;
        int result2 = binarySearch(arr2, k2);
        System.out.println("Position of " + k2 + " in array: " + result2);
    }
}
```

Output :

Position of 4 in array: 3

Position of 445 in array: -1

Time Complexity:O(log n)

Space Complexity : O(1)

6)The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return *an array* ans *of length* nums1.length *such that* ans[i] *is the next greater element as described above.*

Example 1:

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.

- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

Example 2:

Input: nums1 = [2,4], nums2 = [1,2,3,4]

Output: [3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 2 is underlined in nums2 = [1,2,3,4]. The next greater element is 3.

- 4 is underlined in nums2 = [1,2,3,4]. There is no next greater element, so the answer is -1.

Program :

```
import java.util.*;

class Solution {

    public static int[] nextGreaterElement(int[] nums1, int[] nums2) {

        Map<Integer, Integer> nextGreaterMap = new HashMap<>();

        Stack<Integer> stack = new Stack<>();

        for (int num : nums2) {

            while (!stack.isEmpty() && stack.peek() < num) {
```

```
            nextGreaterMap.put(stack.pop(), num);
        }
        stack.push(num);
    }
    while (!stack.isEmpty()) {
        nextGreaterMap.put(stack.pop(), -1);
    }
    int[] ans = new int[nums1.length];
    for (int i = 0; i < nums1.length; i++) {
        ans[i] = nextGreaterMap.get(nums1[i]);
    }
    return ans;
}

public static void main(String[] args) {
    int[] nums1_1 = {4, 1, 2};
    int[] nums2_1 = {1, 3, 4, 2};
```

```
        System.out.println("Output: " +
Arrays.toString(nextGreaterElement(nums1_1, nums2_1))); // Output: [-1, 3, -1]

        int[] nums1_2 = {2, 4};

        int[] nums2_2 = {1, 2, 3, 4};

        System.out.println("Output: " +
Arrays.toString(nextGreaterElement(nums1_2, nums2_2))); // Output: [3, -1]

    }

}
```

Output :

Output : [-1, 3, -1]

Output : [3, -1]

Time Complexity : O(n+m)

Space Complexity : O(n)

7)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.

Program :

```
import java.util.HashSet;

public class Solution {

    public static int unionCount(int[] a, int[] b) {
```

```
    HashSet<Integer> unionSet = new HashSet<>();
    for (int num : a) {
        unionSet.add(num);
    }
    for (int num : b) {
        unionSet.add(num);
    }
    return unionSet.size();
}
public static void main(String[] args) {
    int[] a1 = {1, 2, 3, 4, 5};
    int[] b1 = {1, 2, 3};
    System.out.println("Output: " + unionCount(a1, b1)); // Output: 5
    int[] a2 = {85, 25, 1, 32, 54, 6};
    int[] b2 = {85, 2};
    System.out.println("Output: " + unionCount(a2, b2)); // Output: 7
    int[] a3 = {1, 2, 1, 1, 2};
    int[] b3 = {2, 2, 1, 2, 1};
```

```
        System.out.println("Output: " + unionCount(a3, b3)); // Output: 2
    }
}
```

Output :

Output: 5

Output: 7

Output: 2

Time Complexity : O(m+n)

Space Complexity : O(m+n)