Gopika shree M

22AD034

AI & DS

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.

**Expected Time Complexity:**O(n+(max\_element) )

**Expected Auxiliary Space:**O(max\_element)

**Constraints:**  
1 <= arr.size <= 106  
1<= arr[i] <= 1061 <= k <= n

import java.util.Arrays;

public class Main {

public int kthLargest(int[] arr, int k) {

mergeSort(arr);

return arr[arr.length - k];

}

public void mergeSort(int[] arr) {

if (arr.length > 1) {

int mid = arr.length / 2;

int[] left = Arrays.*copyOfRange*(arr, 0, mid);

int[] right = Arrays.*copyOfRange*(arr, mid, arr.length);

mergeSort(left);

mergeSort(right);

int i = 0, j = 0, m = 0;

while (i < left.length && j < right.length) {

if (left[i] < right[j]) {

arr[m++] = left[i++];

} else {

arr[m++] = right[j++];

}

}

while (i < left.length) {

arr[m++] = left[i++];

}

while (j < right.length) {

arr[m++] = right[j++];

}

}

}

public static void main(String[] args) {

Main obj = new Main();

int result = obj.kthLargest(new int[] {7, 10, 4, 3, 20, 15}, 4);

System.***out***.println(result);

}

}

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](https://practice.geeksforgeeks.org/problems/minimize-the-heights-i/1/).  
**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.

**Expected Time Complexity:** O(n\*logn)  
**Expected Auxiliary Space:** O(n)

**Constraints**  
1 ≤ k ≤ 107  
1 ≤ n ≤ 105  
1 ≤ arr[i] ≤ 107

import java.util.Arrays;

public class Main {

public void getMinDiff(int[] arr, int k ) {

Arrays.*sort*(arr);

int n = arr.length;

int min\_diff = arr[n-1] - arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] - k > 0){

int min\_element = Math.*min*(arr[0] + k, arr[i] - k);

int max\_element = Math.*max*(arr[n-1] - k, arr[i - 1] + k);

min\_diff = Math.*min*(min\_diff, max\_element - min\_element);

}

}

System.***out***.println(min\_diff);

}

public static void main(String[] args) {

Main obj = new Main();

obj.getMinDiff(new int[] {3,9,12,16,20}, 3);

}

}

3) Given a string s, composed of different combinations of '(' , ')', '{', '}', '[', ']', verify the validity of the arrangement.  
An input string is valid if:

         1. Open brackets must be closed by the same type of brackets.  
         2. Open brackets must be closed in the correct order.

**Example 1:**

**Input:**

S = ()[]{}

**Output:** 1

**Explanation:** The arrangement is valid, as both the conditions are followed here.

**Example 2:**

**Input:**

S = ())({}

**Output:** 0

**Explanation:** Arrangement is not valid, as for the bold closing bracket in ()**)**({}, there is no opening bracket of similar kind, before it.

**Your Task:**  
You dont need to read input or print anything. Complete the function **valid()** which takes **s** as input and returns a boolean value denoting whether the arrangement is valid or not.  
  
**Expected Time Complexity:**O(N) where N is the length of s.  
**Expected Auxiliary Space:**O(N)   
  
**Constraints:**  
1 <= N <= 105

import java.util.Stack;

public class Main {

public void validParenthesis(String s) {

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

for (char i: s.toCharArray()) {

if (!st.isEmpty() && st.peek() == '(' && i == ')') {

st.pop();

}

else if (!st.isEmpty() && st.peek() == '[' && i == ']') {

st.pop();

}

else if (!st.isEmpty() && st.peek() == '{' && i == '}') {

st.pop();

}

else st.add(i);

}

System.***out***.println(st.isEmpty());

}

public static void main(String[] args) {

Main obj = new Main();

obj.validParenthesis("()[]{");

}

}

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 beforethat index is the same as the sumof elements afterit.

**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 <= 106  
0 <= arr[i] <= 109

import java.util.Arrays;

import java.util.Stack;

public class Main {

public int equilibiurmPoint(int[] arr) {

int n = arr.length;

int sum = Arrays.*stream*(arr).sum();

int left = arr[0];

for (int i = 1; i< n; i++){

int right = sum - left - arr[i];

if (left == right) return i +1;

left += arr[i];

}

return -1;

}

public static void main(String[] args) {

Main obj = new Main();

int res = obj.equilibiurmPoint(new int[] {1,2,3,5,2,4});

System.***out***.println(res);

}

}

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.

*Note: Try to solve this problem in constant space i.e O(1)*

**Constraints:**1 <= arr.size() <= 1051 <= arr[i] <= 1061 <= k <= 106

import java.util.Arrays;

import java.util.Stack;

public class Main {

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

int n = arr.length;

int left = 0;

int right = n-1;

while (left <= right) {

int mid = (left+right)/2;

if (arr[mid] == k) return mid;

else if (k < arr[mid]) right = mid - 1;

else left = mid + 1;

}

return -1;

}

public static void main(String[] args) {

Main obj = new Main();

int res = obj.binarySearch(new int[] {1, 2, 3, 4, 5}, 4);

System.***out***.println(res);

}

}

6) 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() ≤ 1060 ≤ a[i], b[i] < 105

import java.util.HashMap;

public class Main {

public int unionOfArrays(int[] a, int[] b) {

HashMap<Integer,Integer>temp = new HashMap<>();

int count =0;

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

temp.put(a[i],count++);

}

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

temp.put(b[i],count++);

}

int size = temp.size();

return size;

}

public static void main(String[] args) {

Main obj = new Main();

int res = obj.unionOfArrays(new int[] {1, 2, 3, 4, 5}, new int[] {1,2,3});

System.***out***.println(res);

}

}

7) 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 ≤ arr.size() ≤ 106  
0 ≤ arr[i] ≤ 109

Try more examples

import java.util.ArrayList;

import java.util.Stack;

public class Main {

public ArrayList<Integer> nextLargerElement(int[] arr) {

int n = arr.length;

ArrayList<Integer> result = new ArrayList<>(n);

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

for (int i = 0; i < n; i++) {

result.add(-1);

}

for (int i = n - 1; i >= 0; i--) {

while (!stack.isEmpty() && stack.peek() <= arr[i]) {

stack.pop();

}

if (!stack.isEmpty()) {

result.set(i, stack.peek());

}

stack.push(arr[i]);

}

return result;

}

public static void main(String[] args) {

Main obj = new Main();

ArrayList<Integer> res = obj.nextLargerElement(new int[] {1, 2, 3, 4, 5});

System.***out***.println(res);

}

}