# 13.11.2024

1. **kth smallest** element in the given array

CODE :

import java.util.Arrays;

import java.util.Scanner;

public class Solution {

public static int kthSmallest(int[] arr, int k) {

int ma = 0;

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

ma = Math.max(ma, arr[i]);

}

int[] fre = new int[ma + 1];

Arrays.fill(fre, 0);

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

fre[arr[i]] += 1;

}

int cnt = 0;

for (int i = 0; i < ma + 1; i++) {

if (fre[i] != 0) {

cnt += fre[i];

}

if (cnt >= k) {

return i;

}

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

System.out.print("Enter the value of k: ");

int k = scanner.nextInt();

int result = kthSmallest(arr, k);

if (result != -1) {

System.out.println("The " + k + "th smallest element is: " + result);

} else {

System.out.println("The value of k is out of range.");

}

}

}

Output :

Enter the number of elements in the array: 6

Enter the elements of the array:

3 1 2 1 4 5

Enter the value of k: 3

The 3th smallest element is: 2

**Time Complexity** : O(n + m) where m is the max element in the array

**Space Complexity** : O(m)

1. Minimize the Heights II

CODE:

import java.util.\*;

class Main {

static int getMinDiff(int[] arr, int k) {

int n = arr.length;

Arrays.sort(arr);

int res = arr[n - 1] - arr[0];

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

if (arr[i] - k < 0)

continue;

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

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

res = Math.min(res, max - min);

}

return res;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the value of k: ");

int k = scanner.nextInt();

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

int ans = getMinDiff(arr, k);

System.out.println("The minimized maximum difference is: " + ans);

}

}

Output :

Enter the value of k: 6

Enter the number of elements in the array: 6

Enter the elements of the array: 12 6 4 15 17 10

The minimized maximum difference is: 9

Time Complexity: O(n logn)

Space Complexity: O(n)

1. Parenthesis Checker

CODE :

import java.util.\*;

class Main {

public boolean isValid(String s) {

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

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

if (c == '(')

stack.push(')');

else if (c == '{')

stack.push('}');

else if (c == '[')

stack.push(']');

else if (stack.isEmpty() || stack.pop() != c)

return false;

}

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String input = scanner.nextLine();

Main solution = new Main();

boolean isValid = solution.isValid(input);

System.out.println("Is the string valid? " + isValid);

scanner.close();

}

}

Output:

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

Is the string valid? True

Time Complexity: O(n)

Space Complexity: O(n)

1. Equilibrium Point

CODE :

import java.util.\*;

class Main {

public int equilibriumPoint(int[] arr) {

int n = arr.length;

int[] pre = new int[n];

int[] suf = new int[n];

int p = 0, s = 0;

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

p += arr[i];

pre[i] = p;

s += arr[n - i - 1];

suf[n - i - 1] = s;

}

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

if (pre[i] == suf[i]) {

return i + 1;

}

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

Main solution = new Main();

int equilibriumIndex = solution.equilibriumPoint(arr);

System.out.println("The equilibrium point is at position: " + equilibriumIndex);

}

}

Output:

Enter the number of elements in the array: 4

Enter the elements of the array:

1 2 3 4

The equilibrium point is at position: -1

Time Complexity: O(n)

Space Complexity : O(2n) 🡪 O(n)

5 . Binary Search

CODE :

import java.util.Scanner;

import java.util.Arrays;

class Solution {

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

int left = 0;

int right = arr.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

} else if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array in sorted order:");

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

arr[i] = scanner.nextInt();

}

System.out.print("Enter the target element to search for: ");

int target = scanner.nextInt();

Solution solution = new Solution();

int result = solution.binarySearch(arr, target);

if (result != -1) {

System.out.println("Element found at index: " + result);

} else {

System.out.println("Element not found in the array.");

}

}

}

Output:

Enter the number of elements in the array: 5

Enter the elements of the array in sorted order:

1 3 5 7 9

Enter the target element to search for: 5

Element found at index: 2

Time Complexity : O(log n)

Space Complexity : O(1)

6 . **Union of Two Arrays with Duplicate Elements**

CODE :

import java.util.HashSet;

import java.util.Scanner;

class Main{

public int findUnion(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[] sasta) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the first array: ");

int n = scanner.nextInt();

int[] a = new int[n];

System.out.println("Enter the elements of the first array:");

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

a[i] = scanner.nextInt();

}

System.out.print("Enter the number of elements in the second array: ");

int m = scanner.nextInt();

int[] b = new int[m];

System.out.println("Enter the elements of the second array:");

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

b[i] = scanner.nextInt();

}

Main solution = new Main();

int unionCount = solution.findUnion(a, b);

System.out.println("The count of elements in the union of both arrays is: " + unionCount);

scanner.close();

}

}

Output :

Enter the number of elements in the first array: 4

Enter the elements of the first array:

1 2 3 4

Enter the number of elements in the second array: 5

Enter the elements of the second array:

3 4 5 6 7

The count of elements in the union of both arrays is: 7

Time Complexity: O(n)

Space Complexity : O(n)