**Karthick M\_AI&DS\_Day-4\_Practice**

**1.Kth smallest element**

**Program:**

import java.util.PriorityQueue;

import java.util.Scanner;

public class Solution {

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

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

for (int num : arr) {

minHeap.add(num);

}

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

minHeap.poll();

}

return minHeap.peek();

}

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();

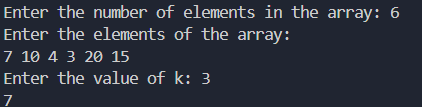
int result = kthSmallest(arr, k);

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

}

}

**Output:**

****

**Time Complexity:** O(nlogn)

**2. Minimise the Heights II**

**Program:**

import java.util.Arrays;

import java.util.Scanner;

class MinimumHeights {

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

Arrays.sort(arr);

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

int minHeight = arr[0] + k;

int maxHeight = arr[n - 1] - k;

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

if (arr[i] < k) continue;

int minElement = Math.min(minHeight, arr[i] - k);

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

result = Math.min(result, maxElement - minElement);

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of towers: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the heights of the towers:");

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

arr[i] = scanner.nextInt();

}

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

int k = scanner.nextInt();

MinimumHeights sol = new MinimumHeights();

int result = sol.getMinDiff(arr, n, k);

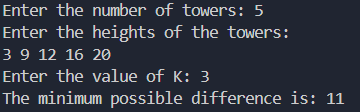
System.out.println("The minimum possible difference is: " + result);

scanner.close();

}

}

**Output:**

****

**Time Complexity:** O(nlogn)

**3. Parentheses Checker**

**Program:**

import java.util.Stack;

import java.util.Scanner;

public class Solution {

public static boolean isBalanced(String s) {

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

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

if (ch == '{' || ch == '(' || ch == '[') {

stack.push(ch);

} else if (ch == '}' || ch == ')' || ch == ']') {

if (stack.isEmpty()) {

return false;

}

char top = stack.pop();

if ((ch == '}' && top != '{') || (ch == ')' && top != '(') || (ch == ']' && top != '[')) {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the expression: ");

String s = scanner.nextLine();

System.out.println(isBalanced(s));

}

}

**Output:**

****

**Time Complexity:** O(n)

**4. Equllibrium Point**

**Program:**

import java.util.Scanner;

public class Solution {

public static int findEquilibriumPoint(int[] arr) {

int totalSum = 0;

int leftSum = 0;

for (int num : arr) {

totalSum += num;

}

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

totalSum -= arr[i];

if (leftSum == totalSum) {

return i + 1;

}

leftSum += arr[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();

}

int result = findEquilibriumPoint(arr);

if (result == -1) {

System.out.println("No equilibrium point found");

} else {

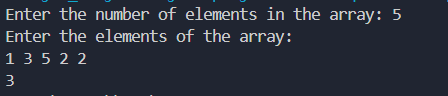
System.out.println("The equilibrium point is at index: " + result);

}

}

}

**Output:**

****

**Time Complexity:** O(n)

**5. Binary Search**

**Program:**

import java.util.Scanner;

public class Solution {

public static 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;

}

if (arr[mid] > target) {

right = mid - 1;

} else {

left = 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 sorted array:");

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

arr[i] = scanner.nextInt();

}

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

int target = scanner.nextInt();

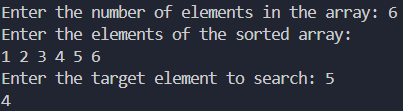
int result = binarySearch(arr, target);

System.out.println(result);

}

}

**Output:**

****

**Time Complexity:** O(1)

**6. Next greater element**

**Program:**

import java.util.Stack;

import java.util.\*;

public class Problem18 {

public static void printNGE(int[] arr) {

int n = arr.length;

int[] nge = new int[n];

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

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

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

stack.pop();

}

nge[i] = stack.isEmpty() ? -1 : stack.peek();

stack.push(arr[i]);

}

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

System.out.println(arr[i] + " -> " + nge[i]);

}

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("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();

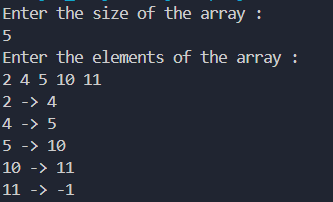
}

printNGE(arr);

}

}

**Output:**

****

**Time Complexity:** O(n)

**7.Union of two arrays with duplicate elements**

**Program:**

import java.util.HashSet;

import java.util.Scanner;

public class ArrayUnion {

public static int findUnion(int[] a, int[] b) {

HashSet<Integer> uList = new HashSet<>();

for (int num : a) {

uList.add(num);

}

for (int num : b) {

uList.add(num);

}

return uList.size();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements in array 1: ");

int n1 = sc.nextInt();

int[] arr1 = new int[n1];

System.out.println("Enter elements of array 1:");

for (int i = 0; i < n1; i++) {

arr1[i] = sc.nextInt();

}

System.out.print("Enter number of elements in array 2: ");

int n2 = sc.nextInt();

int[] arr2 = new int[n2];

System.out.println("Enter elements of array 2:");

for (int i = 0; i < n2; i++) {

arr2[i] = sc.nextInt();

}

int unionSize = findUnion(arr1, arr2);

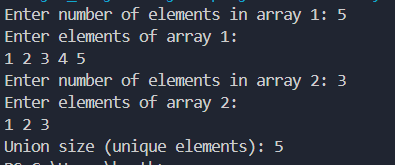
System.out.println("Union size (unique elements): " + unionSize);

sc.close();

}

}

**Output:**

****

**Time Complexity:** O(n+m)