**Day 4-DSA practice**

1)Kth smallest Element:

Given an array **arr[]**of **N** distinct elements and a number **K,** where **K** is smaller than the size of the array. Find the **K’th** smallest element in the given array.

**Examples:**

***Input****: arr[] = {7, 10, 4, 3, 20, 15}, K = 3****Output****: 7*

***Input****: arr[] = {7, 10, 4, 3, 20, 15}, K = 4****Output****: 10*

*Program:*

*public class KthSmallestElement*

*{*

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

*int min=0;*

*while(k!=0){*

*min=Integer.MAX\_VALUE;*

*int j=0;*

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

*if(min>arr[i]){*

*min=arr[i];*

*j=i;*

*}*

*}*

*arr[j]=Integer.MAX\_VALUE;*

*k--;*

*}*

*return min;*

*}*

*public static void main(String[] args){*

*KthSmallestElement obj=new KthSmallestElement();*

*int[] arr = { 12, 3, 5, 7, 19 };*

*int K = 2;*

*System.out.println(obj.kthSmallest(arr,K));*

*}*

*}*

Time Complexity:O(n\*2)

Space Complexity:O(1)

Using MinHeap:

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

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

for(int n:arr){

minheap.add(n);

}

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

minheap.poll();

}

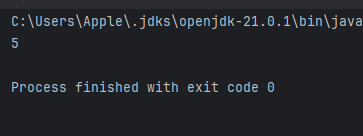
return minheap.poll();

}

Time Complexity:O(n\*log n)

Space Complexity:O(n)

output :



**3)minimize the Height ii**

Program;

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.

Program:

import java.util.Arrays;

public class Minimumheight {

public static void main(String[] args){

int k = 6;

int[] arr = {12, 6, 4, 15, 17, 10};

Minimumheight obj=new Minimumheight();

System.out.println(obj.getMinDiff(arr,k));

}

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

// code here

Arrays.sort(arr);

int n=arr.length;

if(n==0){

return 0;

}

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

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

if(arr[i]<k){

continue;

}

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

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

mindiff=Math.min(mindiff,maxval-minval);

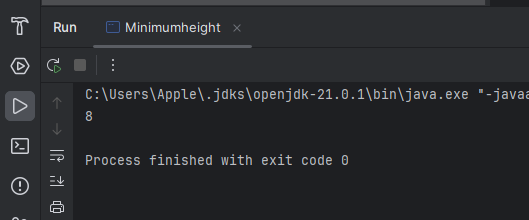
}

return mindiff;

}

}

Output:



Time Complexity:O(n log \*n)

Space Complexity:O(1)

**3)Paranthesis Checkup:**

Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not

Input: str = “((()))()()”

Output: Balanced

Input: str = “())((())”

Output: Not Balanced

Program:

import java.util.Scanner;

public class BalancedParanthesis {

public static void main(String[] args){

Scanner sc =new Scanner(System.in);

String s=sc.nextLine();

BalancedParanthesis obj=new BalancedParanthesis();

obj.balanced(s);

}

public void balanced(String str){

int count=0;

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

char c=str.charAt(i);

if(c=='('){

count++;

}

else if(c==')'){

count--;

}

}

if(count<0){

System.out.println("Not Balanced");

}

else{

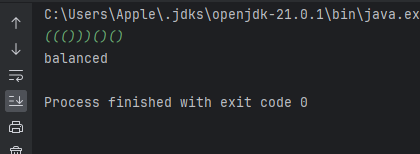
System.out.println("balanced");

}

}

}

Output:



**TimeComplexity** :O(n) **Space Complexity**:O(1)

**4)Equlibrium point:**

Given an array **arr[]**of size**n**, return an equilibrium index (if any) or -1 if no equilibrium index exists. The **equilibrium index of an array**is an index such that the sum of elements at lower indexes equals the sum of elements at higher indexes.

**Note:** Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

**Examples:**

***Input****: arr[] = {-7, 1, 5, 2, -4, 3, 0}****Output****: 4****Explanation:*** *In 1-based indexing, 4 is an equilibrium index, because: arr[1] + arr[2] + arr[3] = arr[5] + arr[6] + arr[7]*

Program:

public class equlibriumpoin {

public static void main(String[] args) {

equlibriumpoin obj=new equlibriumpoin();

int[] arr = { -7, 1, 5, 2, -4, 3, 0 };

System.out.println(obj.equilibriumPoint(arr));

}

public static int equilibriumPoint(int arr[]) {

// code here

int rsum=0;

int lsum=0;

for(int i:arr){

rsum+=i;

}

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

rsum=rsum-arr[j];

if(lsum==rsum){

return j+1;

}

lsum=lsum+arr[j];

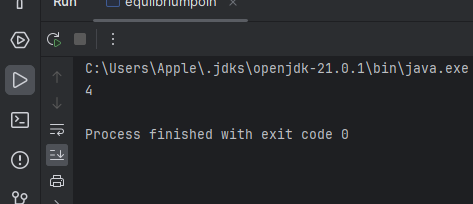
}

return -1;

}

}

Output:



Time Complexity:O(n)

Space Complexity:O(1)

**5)Binary Search**

public class BinarySearch {

public static void main(String[] args) {

BinarySearch obj=new BinarySearch();

int[] arr={2, 3, 4, 10, 40};

int x=10;

System.out.println(obj.binarysearch(arr,x));

}

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

// Code Here

int low=0;

int high=arr.length-1;

while(low<=high){

int mid=low+(high-low)/2;

if(arr[mid]==k){

return mid;

}

else if(arr[mid]>k){

high=mid-1;

}

else{

low=mid+1;

}

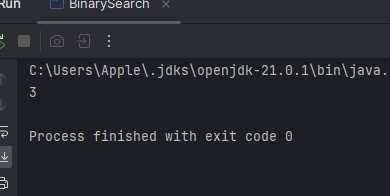
}

return -1;

}

}

Output:



Time Complexity:O(log n)

Space Complexity:O(1)

**6)Next Greater Element:**

Program:

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Stack;

public class NextgreaterElement {

public static void main(String[] args) {

NextgreaterElement obj = new NextgreaterElement();

int[] arr = {4, 5, 2, 10, 8};

int n = arr.length;

System.out.println(Arrays.toString(obj.greater(arr, n))); // Print result as a string

}

public int[] greater(int[] arr, int n) {

int[] result = new int[n];

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

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

result[i] = -1;

}

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

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

int index = stack.pop();

result[index] = arr[i]; // Update the result for that index

}

// Push the current index onto the stack

stack.push(i);

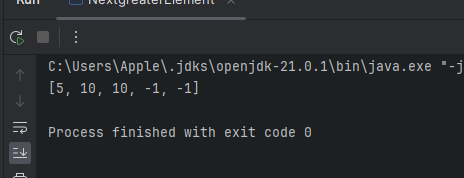
}

return result;

}

}

Output:



Time Complexity:O( n)

Space Complexity:O(n)

7)Union of two array with Duplicate

Program:

import java.util.HashSet;

public class UnionoftwoArray {

public static void main(String[] args) {

UnionoftwoArray obj=new UnionoftwoArray();

int[] a={1,2,3,4,5};

int[] b={1,2,3};

System.out.println(obj.findUnion(a,b));

}

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

// code here

int l1=a.length;

int l2=b.length;

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

for(int i:a){

set.add(i);

}

for(int j:b){

set.add(j);

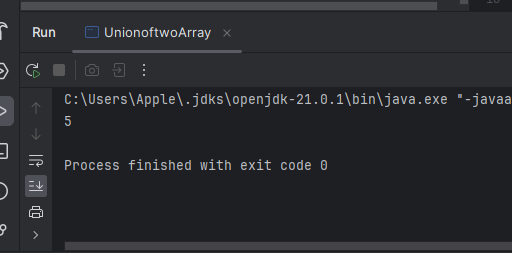
}

return set.size();

}

}

Output:



Time Complexity:O( n)

Space Complexity:O(n)