**1.** **K-th smallest element**

**CODE:**

import java.util.Arrays;

public class KthSmallest{

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

        mergesort(arr);

        return arr[k-1];

    }

    private static void mergesort(int[] arr){

            if(arr.length>1){

                int mid=arr.length/2;

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

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

                mergesort(l);

                mergesort(r);

                int i=0,j=0,m=0;

                while(i<l.length && j<r.length){

                    if(l[i]<r[j]){

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

                    }

                    else {

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

                    }

                }

                while(i<l.length){

                    arr[m++]=l[i++];

                }

                while(j<r.length){

                    arr[m++]=r[j++];

                }

            }

        }

    public static void main(String[] args) {

        KthSmallest k=new KthSmallest();

        int arr1[]={7, 10, 4, 3, 20, 15};

        int k1=3;

        int arr2[]={2, 3, 1, 20, 15};

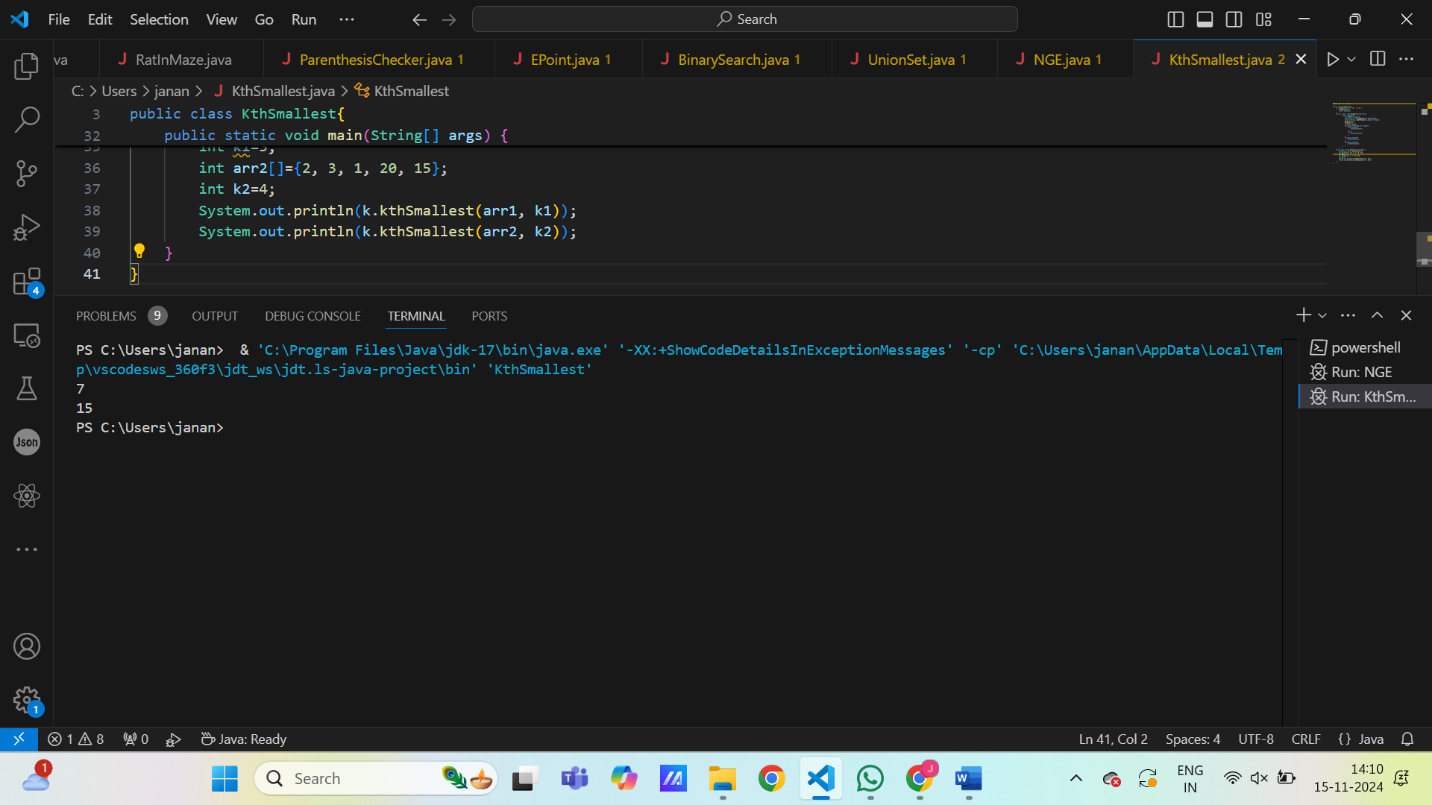
        int k2=4;

        System.out.println(k.kthSmallest(arr1, k1));

        System.out.println(k.kthSmallest(arr2, k2));

    }

}

**OUTPUT:**

**TIME COMPLEXITY:** O(nlogn)

**SPACE COMPLEXITY:** O(n)

**2.Minimize the heights-II  
CODE:**

import java.util.Arrays;

public class MinimizeHeights{

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

        Arrays.sort(arr);

        int n=arr.length;

        int min=arr[0];

        int max=arr[n-1];

        int diff=max-min;

        for(int i=1;i<n;i++){

            if(arr[i]-k>0){

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

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

            diff=Math.min(maxi-mini,diff);

            }

        }

        return diff;

    }

    public static void main(String[] args) {

        MinimizeHeights m=new MinimizeHeights();

        int arr1[]={1, 5, 8, 10};

        int k1=2;

        int arr2[]={3, 9, 12, 16, 20};

        int k2=3;

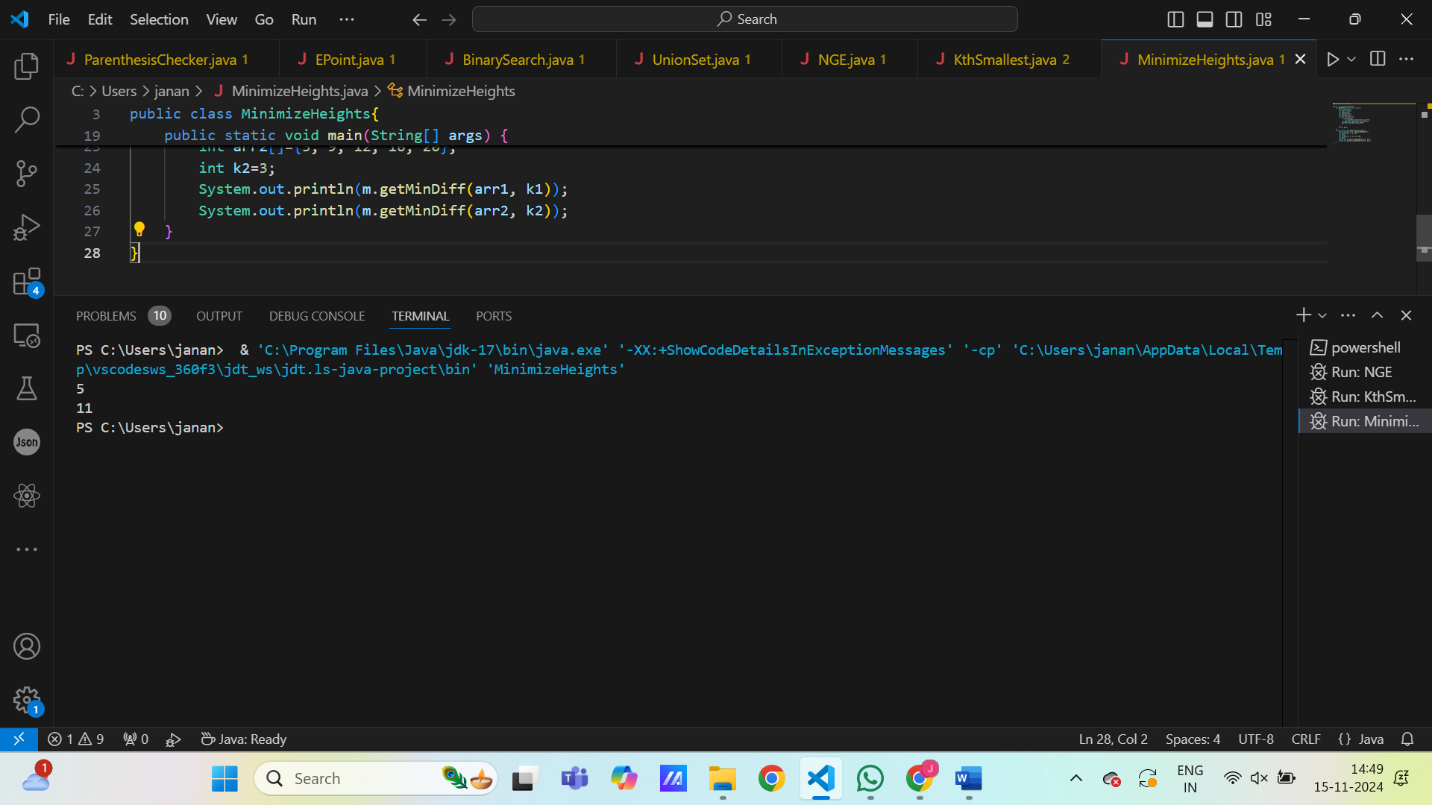
        System.out.println(m.getMinDiff(arr1, k1));

        System.out.println(m.getMinDiff(arr2, k2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(1)

**3.** **Parenthesis checker**

**CODE:**

import java.util.Stack;

public class ParenthesisChecker{

    boolean checker(String s){

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

        int n=s.length();

        char curr;

        char prev;

        st.push(s.charAt(0));

        for(int i=1;i<n;i++){

            if(st.isEmpty()){

                 st.push(s.charAt(i));

                 continue;

            }

            prev=st.pop();

            curr=s.charAt(i);

            if((prev=='[' && curr==']') || (prev=='{' && curr=='}') || (prev=='(' && curr==')') ){

                continue;

            }

            else{

                st.push(prev);

                st.push(curr);

            }

        }

        return st.isEmpty();

    }

    public static void main(String args[]){

        ParenthesisChecker p=new ParenthesisChecker();

        String s1="{([])}";

        String s2="()";

        String s3="((())))";

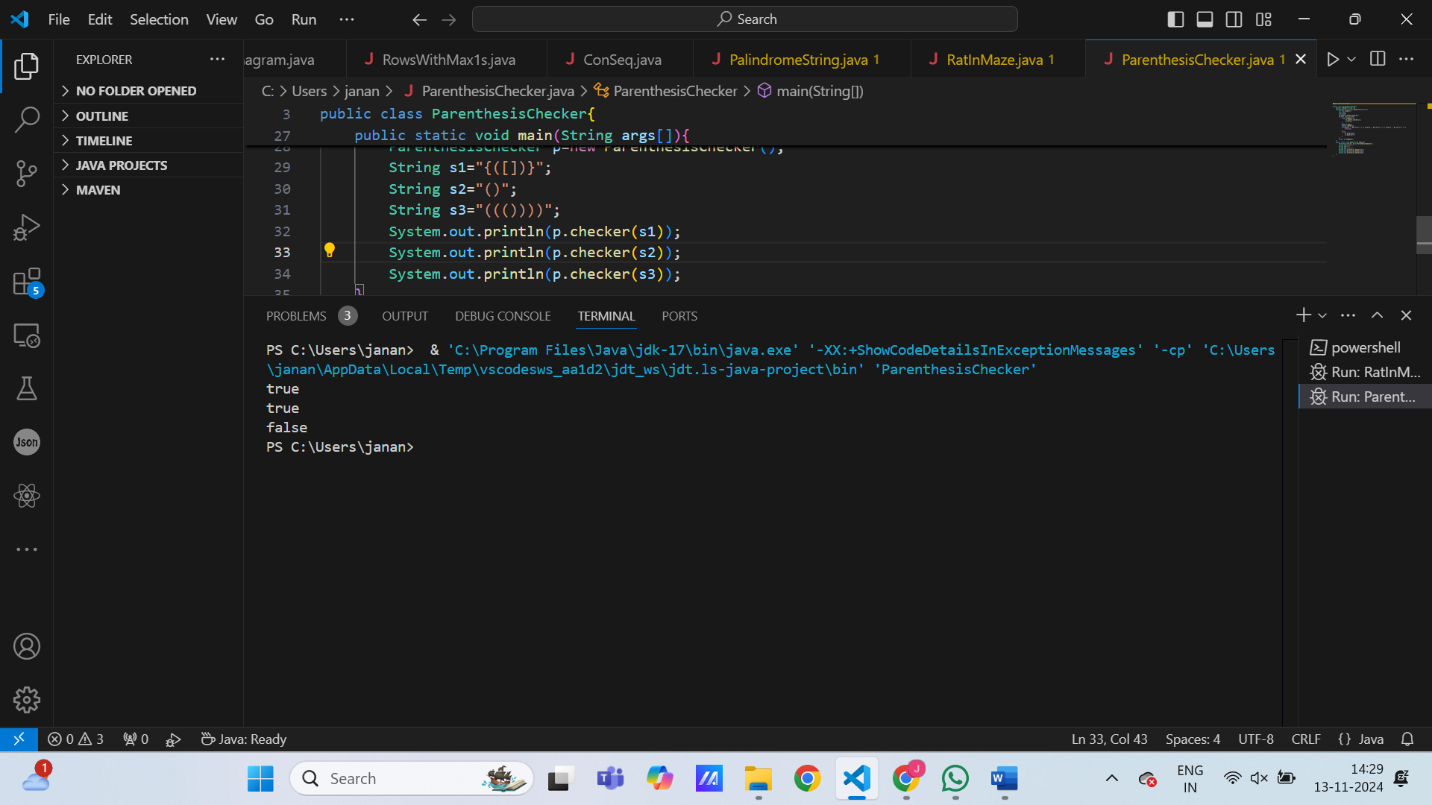
        System.out.println(p.checker(s1));

        System.out.println(p.checker(s2));

        System.out.println(p.checker(s3));

    }}

**OUTPUT:**



**TIME COMPLEXITY:** O(n)

**SPACE COMPLEXITY:** O(n)

**4.** **Equilibrium point**

**CODE:**

public class EPoint{

    int findEPoint(int[] arr){

        int n=arr.length;

        int sum=0;

        int lsum=0;

        for(int i=0;i<n;i++){

            sum+=arr[i];

        }

        for(int i=0;i<n;i++){

            int rsum=sum-lsum-arr[i];

            if(lsum==rsum){

                return i+1;

            }

            lsum+=arr[i];

        }

        return -1;

    }

    public static void main(String args[]){

        int[] arr1={1, 3, 5, 2, 2};

        int[] arr2={1};

        int[] arr3={1, 2, 3};

        EPoint e=new EPoint();

        System.out.println(e.findEPoint(arr1));

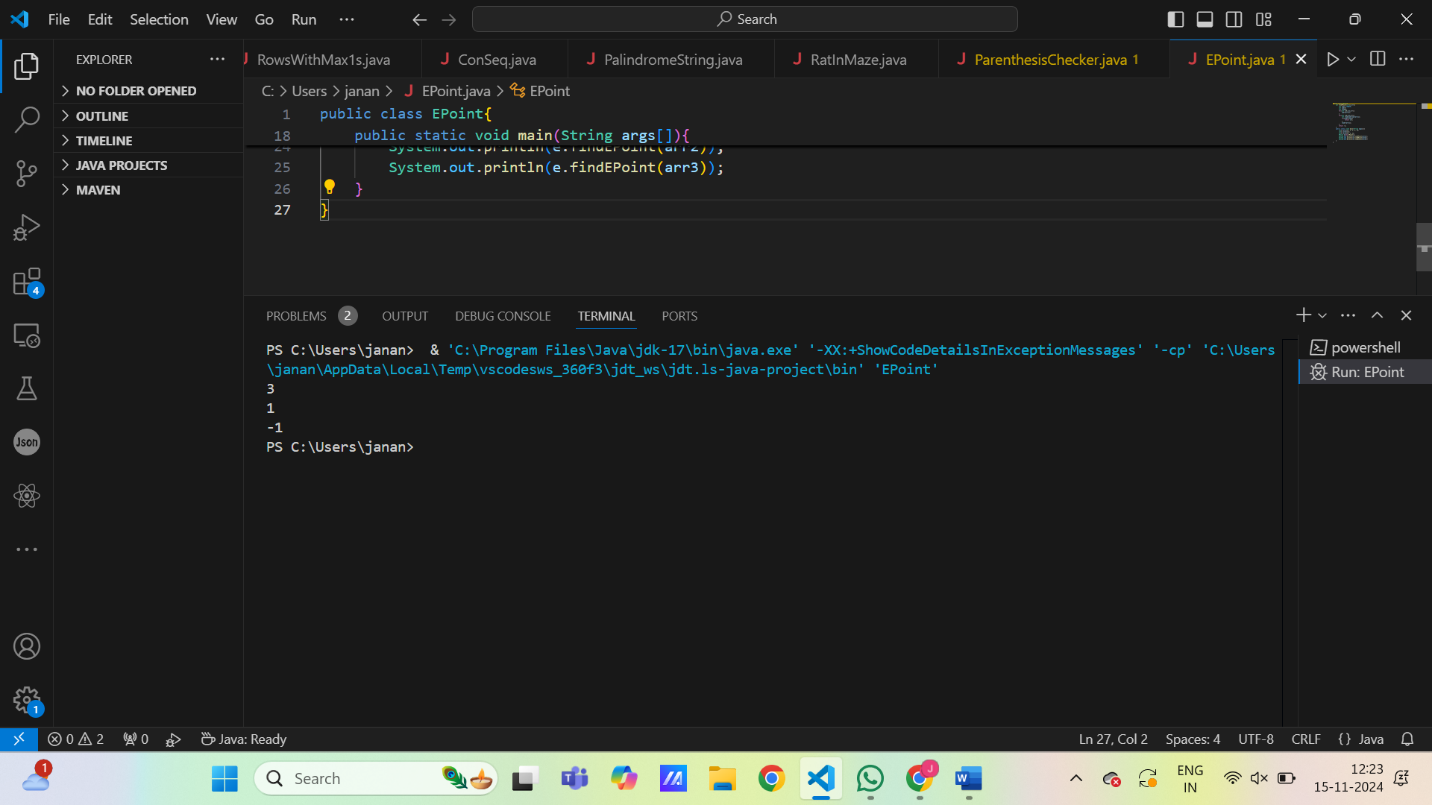
        System.out.println(e.findEPoint(arr2));

        System.out.println(e.findEPoint(arr3));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n)

**SPACE COMPLEXITY:** O(1)

**5.Binary Search:**

**CODE:**

public class BinarySearch{

    int search(int arr[],int k){

        int low=0;

        int high=arr.length-1;

        int mid;

        while(low<=high){

            mid=(low+high)/2;

            if(arr[mid]==k){

                return mid;

            }

            if(k<arr[mid]){

                high=mid-1;

            }

            if(k>arr[mid]){

                low=mid+1;

            }

        }

        return -1;

    }

    public static void main(String args[]){

        int[] arr1={1, 2, 3, 4, 5};

        int[] arr2={11, 22, 33, 44, 55};

        BinarySearch b=new BinarySearch();

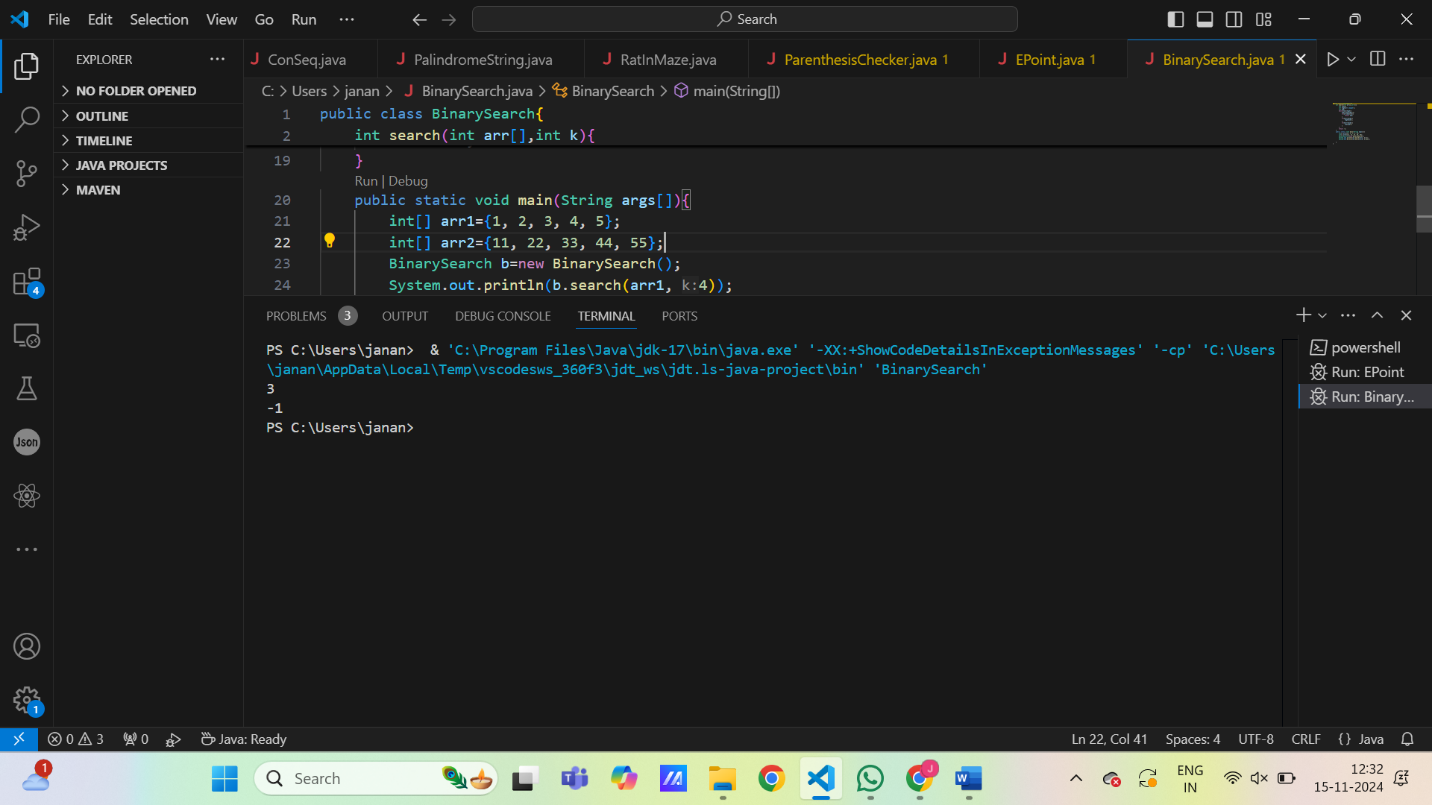
        System.out.println(b.search(arr1, 4));

        System.out.println(b.search(arr2, 445));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(nlogn) **SPACE COMPLEXITY:** O(1)

**6.Next Greatest Element**

**CODE:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Stack;

public class NGE{

    ArrayList<Integer> nge(int arr[]){

        int n = arr.length;

        ArrayList<Integer> nge = new ArrayList<>();

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

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

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

                stack.pop();

            }

            if(stack.isEmpty()){

                nge.add(-1);

            }

            else{

                nge.add(stack.peek());

            }

            stack.push(arr[i]);

        }

        Collections.reverse(nge);

        return nge;

    }

    public static void main(String[] args) {

        NGE N=new NGE();

        int[] arr1={1, 3, 2, 4};

        int[] arr2={6, 8, 0, 1, 3};

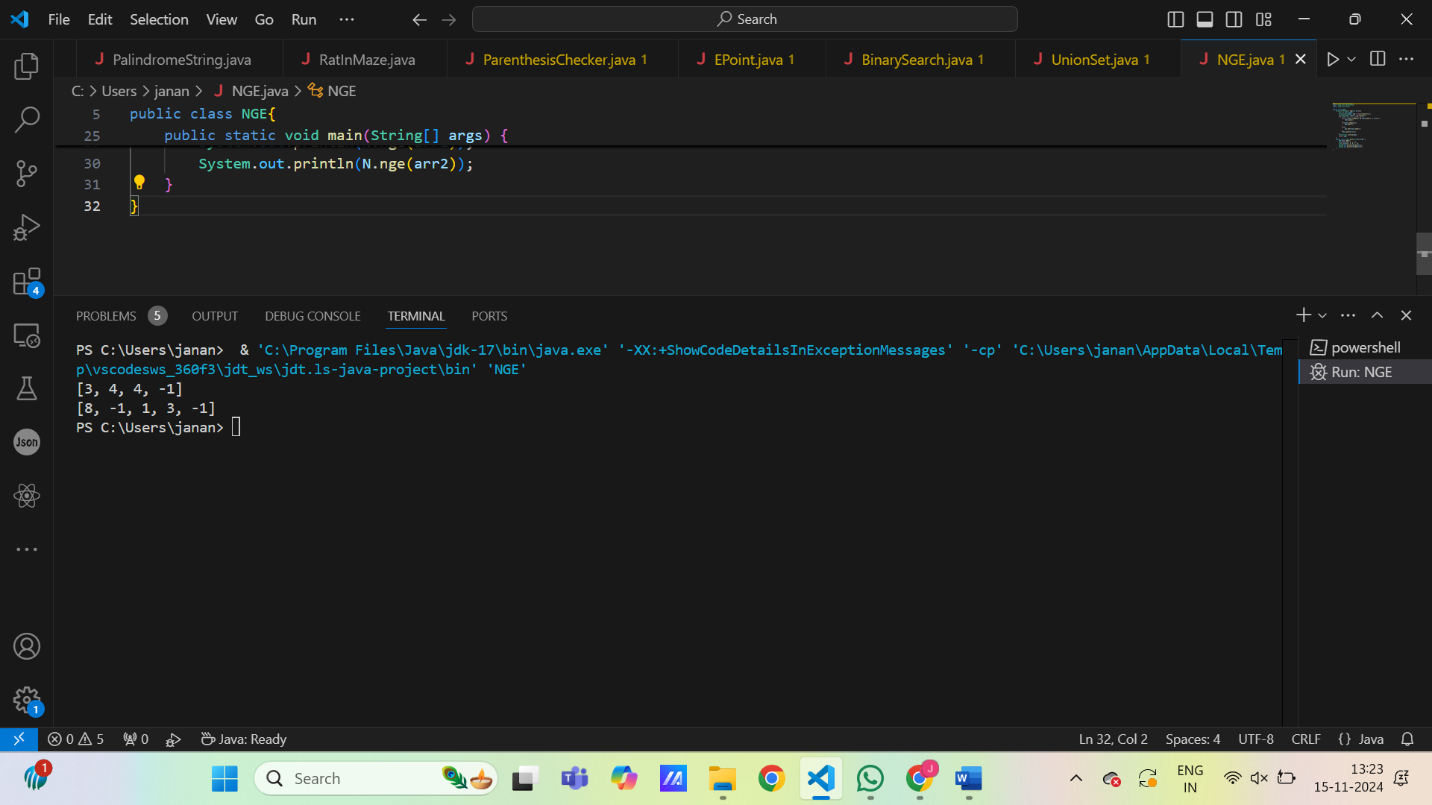
        System.out.println(N.nge(arr1));

        System.out.println(N.nge(arr2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(n)

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

**CODE:**

import java.util.HashSet;

public class UnionSet{

    int count(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};

        int a2[]={85, 25, 1, 32, 54, 6};

        int b2[]={85, 2};

        UnionSet u=new UnionSet();

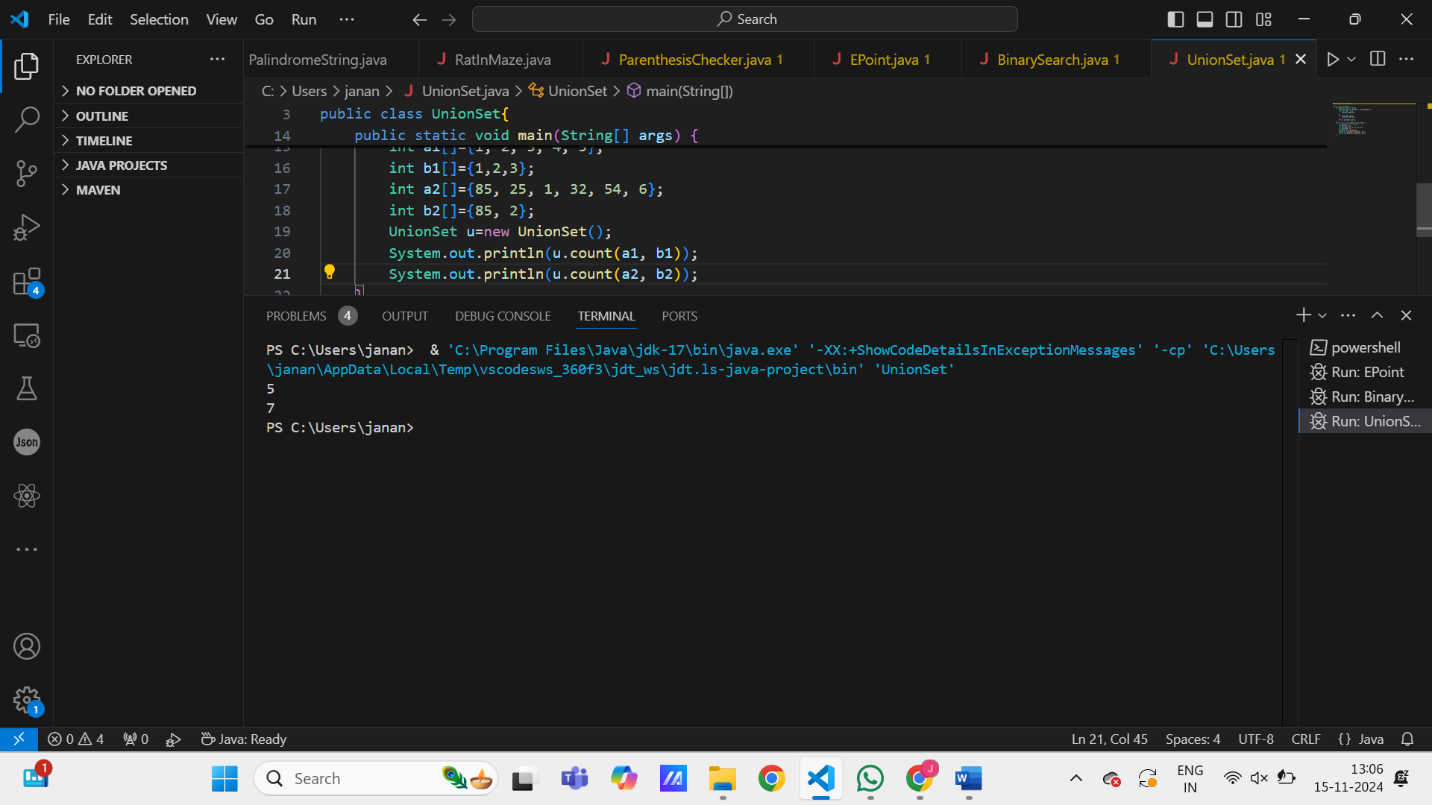
        System.out.println(u.count(a1, b1));

        System.out.println(u.count(a2, b2));

    }

}

**OUTPUT:**

 **TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(n)