**1.Bubble Sort:  
CODE:**

public class BubbleSort{

    void bubbleSort(int arr[]) {

        int i=0,j=0,temp;

        boolean swap=false;

        int n=arr.length;

        for(i=0;i<n;i++){

            for(j=0;j<n-i-1;j++){

                if(arr[j]>arr[j+1]){

                    temp=arr[j];

                    arr[j]=arr[j+1];

                    arr[j+1]=temp;

                    swap=true;

                }

            }

            if(swap==false){

                break;

            }

        }

    }

    public static void main(String[] args) {

        QuickSort q=new QuickSort();

        int[] arr1={4, 1, 3, 9, 7};

        int[] arr2={10, 9, 8, 7, 6, 5, 4, 3, 2, 1};

        q.bubbleSort(arr1);

        q.bubbleSort(arr2);

        for(int i=0;i<arr1.length;i++){

            System.out.print(arr1[i]+" ");

        }

        System.err.println();

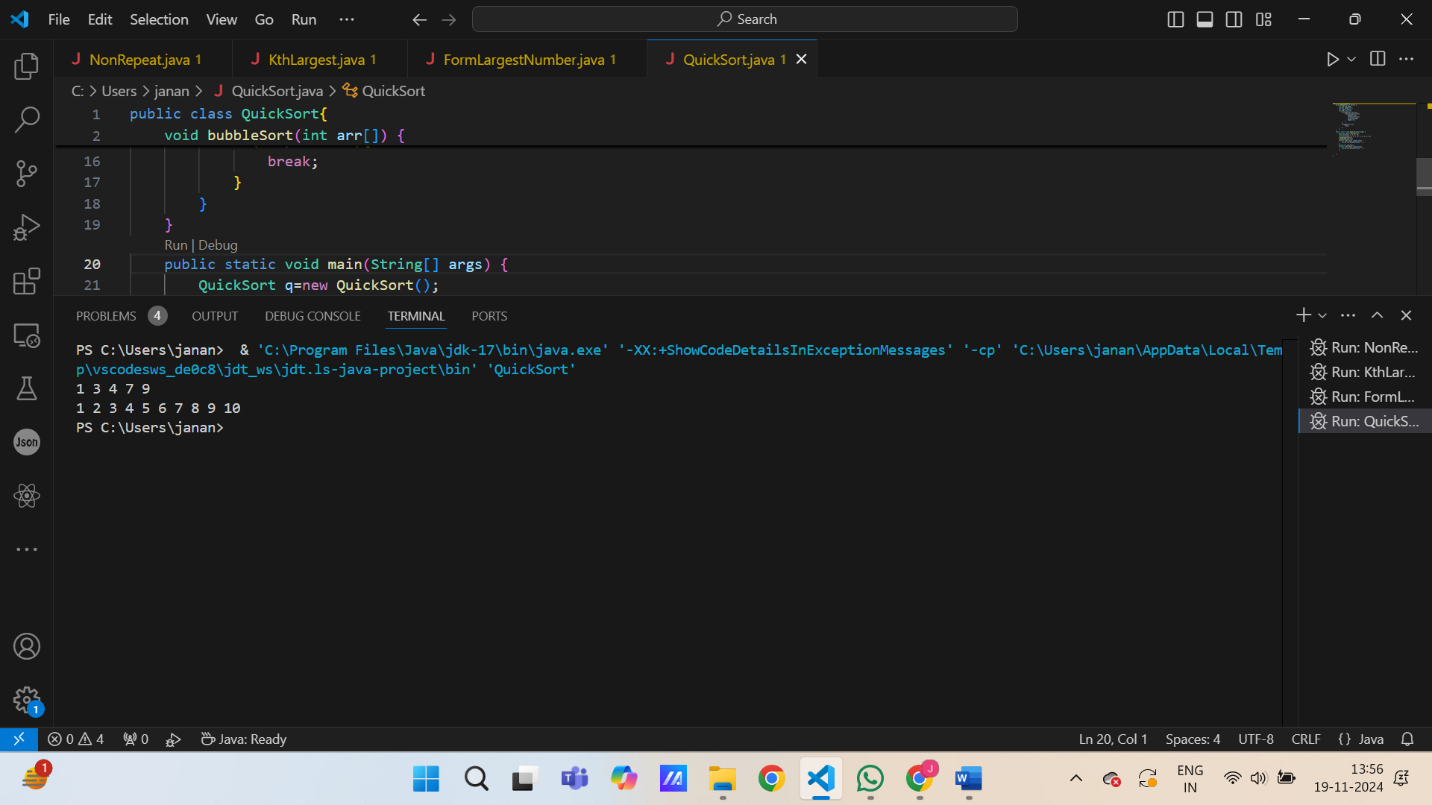
        for(int i=0;i<arr2.length;i++){

            System.out.print(arr2[i]+" ");

        }

    }

}

**OUTPUT:**

**TIME COMPLEXITY:** O(n^2) **SPACE COMPLEXITY:** O(1)

**2.Quick Sort:**

static void quickSort(int arr[], int low, int high) {

if(low<high){

int pi=partition(arr,low,high);

quickSort(arr,low,pi-1);

quickSort(arr,pi+1,high);

}

}

static int partition(int arr[], int low, int high) {

int p=arr[high];

int i=low-1;

for(int j=low;j<high;j++){

if(arr[j]<p){

i++;

swap(arr,i,j);

}

}

swap(arr, i + 1, high);

return i+1;

}

static void swap(int[] arr, int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

**TIME COMPLEXITY:** O(nlogn)

**SPACE COMPLEXITY:** O(1)

**3.Non Repeating Character:**

**CODE:**

import java.util.HashMap;

public class NonRepeat{

    char nonRepeatingChar(String s) {

        HashMap<Character,Integer> h=new HashMap<Character,Integer>();

        int n=s.length();

        for(int i=0;i<n;i++){

            h.put(s.charAt(i),h.getOrDefault(s.charAt(i),0)+1);

        }

        for(int i=0;i<n;i++){

            if(h.get(s.charAt(i))==1){

                return s.charAt(i);

            }

        }

        return '$';

    }

    public static void main(String args[]){

        String s1="geeksforgeeks";

        String s2="racecar";

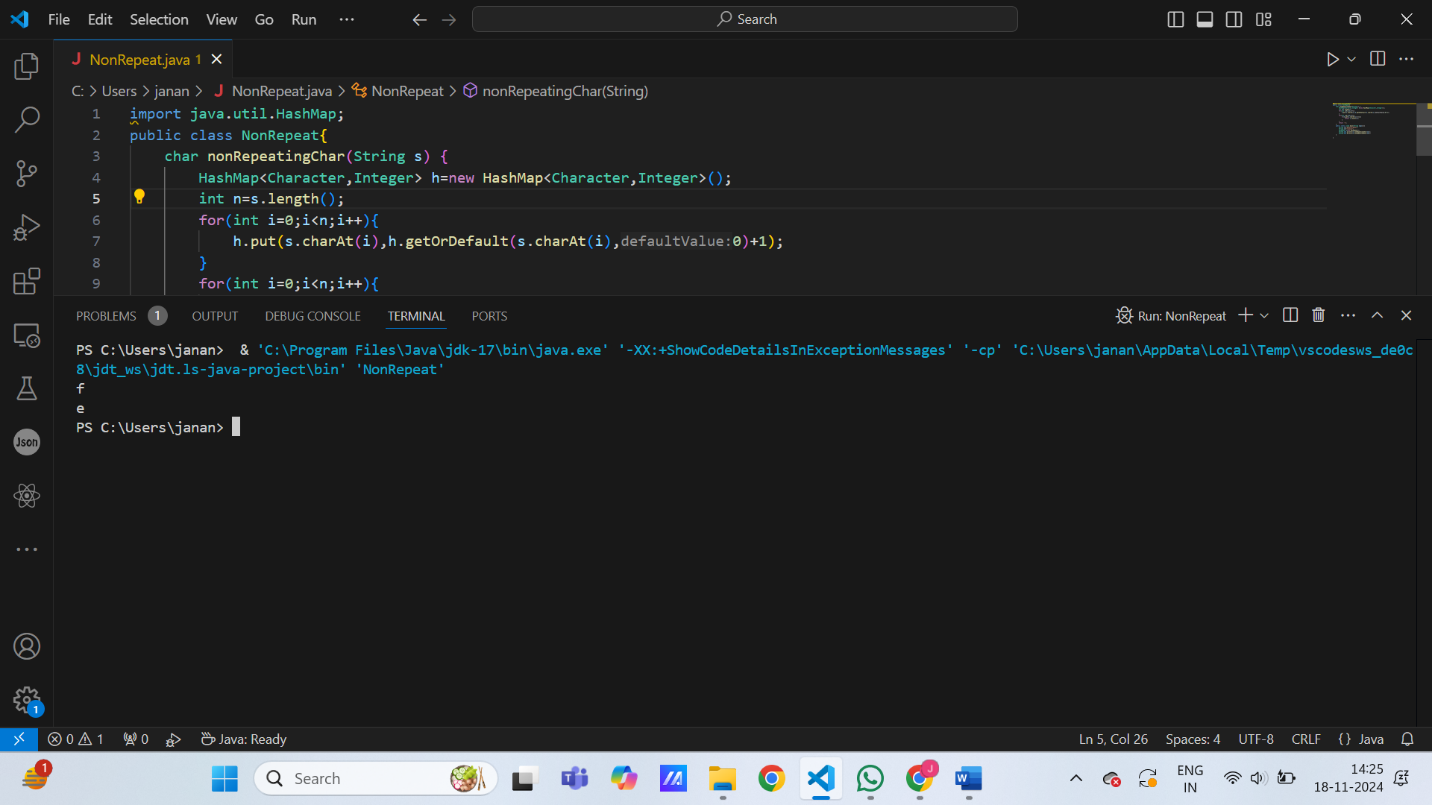
        NonRepeat n=new NonRepeat();

        System.out.println(n.nonRepeatingChar(s1));

        System.out.println(n.nonRepeatingChar(s2));}}

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

**OUTPUT:**



**4.Edit distance**

**CODE:**

//{ Driver Code Starts

import java.io.\*;

import java.lang.\*;

import java.util.\*;

class GFG {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int T = Integer.parseInt(br.readLine().trim());

while (T-- > 0) {

String s1 = br.readLine();

String s2 = br.readLine();

Solution ob = new Solution();

int ans = ob.editDistance(s1, s2);

System.out.println(ans);

}

}

}

// } Driver Code Ends

class Solution {

public int editDistance(String s1, String s2) {

int m=s1.length();

int n=s2.length();

int[][] dp=new int[m+1][n+1];

dp[0][0]=0;

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

dp[i][0]=i;

}

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

dp[0][j]=j;

}

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

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

if(s1.charAt(i-1)==s2.charAt(j-1)){

dp[i][j] = dp[i-1][j-1];

}

else{

dp[i][j]=Math.min(Math.min(dp[i-1][j-1],dp[i][j-1]),dp[i-1][j])+1;

}

}

}

return dp[m][n];

}

}

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

**5. Kth Largest Element:**import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.stream.Collectors;

public class KthLargest{

    List<Integer> kLargest(int arr[], int k) {

            mergesort(arr);

            List<Integer> reversed = Arrays.stream(arr).boxed().collect(Collectors.toList());

            Collections.reverse(reversed);

            return reversed.subList(0, k);

            }

            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) {

        KthLargest k=new KthLargest();

        int[] arr1={12, 5, 787, 1, 23};

        int k1=2;

        int[] arr2={1, 23, 12, 9, 30, 2, 50};

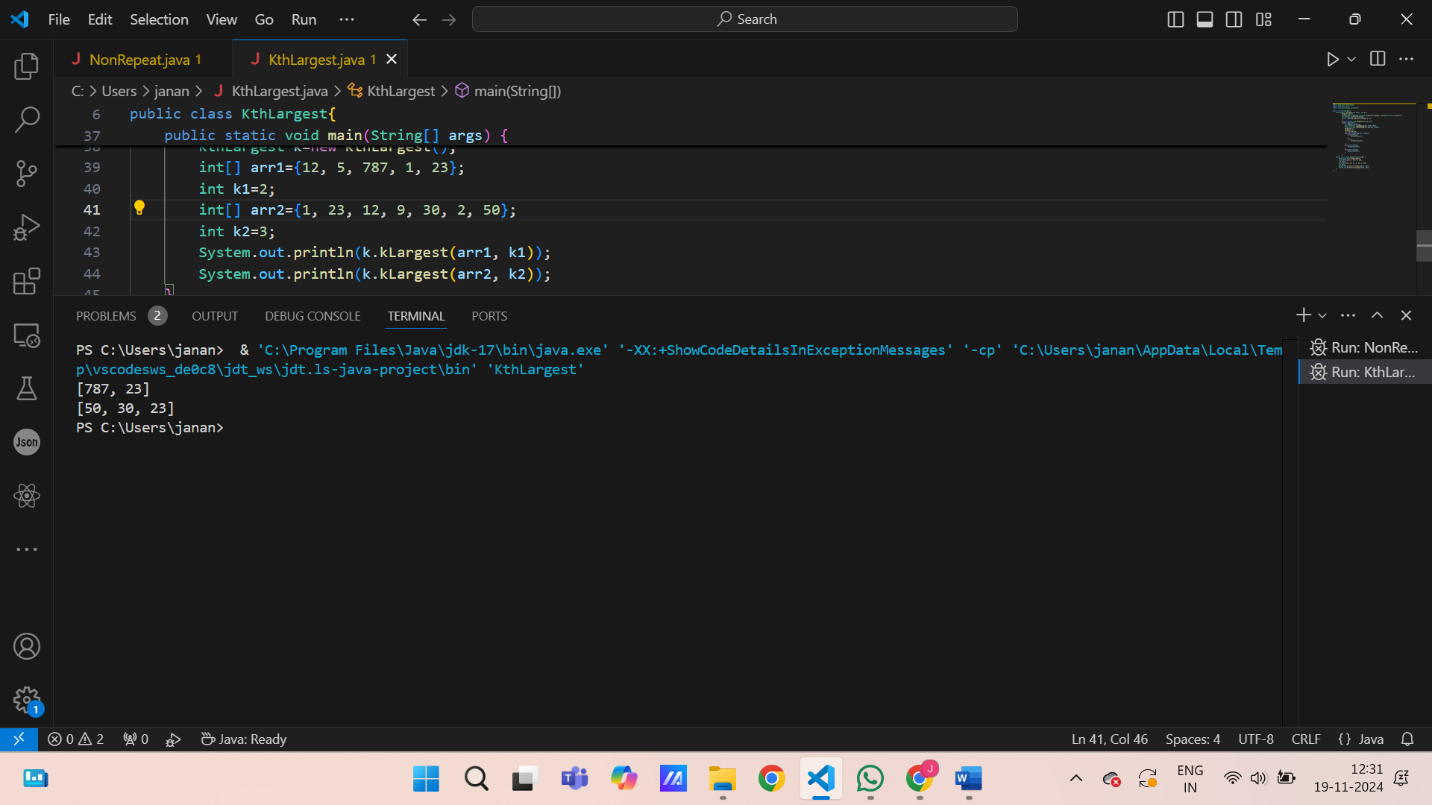
        int k2=3;

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

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

    }}

**OUTPUT:**



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

**6.Form Largest Number:  
CODE:**

import java.util.Arrays;

public class FormLargestNumber{

    String printLargest(int[] arr) {

        String[] nums = new String[arr.length];

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

            nums[i] = Integer.toString(arr[i]);

        }

        Arrays.sort(nums, (a, b) -> (b + a).compareTo(a + b));

        if (nums[0].equals("0")) {

            return "0";

        }

        StringBuilder result = new StringBuilder();

        for (String num : nums) {

            result.append(num);

        }

        return result.toString();

    }

    public static void main(String[] args) {

        FormLargestNumber f=new FormLargestNumber();

        int[] arr1={3, 30, 34, 5, 9};

        int[] arr2={54, 546, 548, 60};

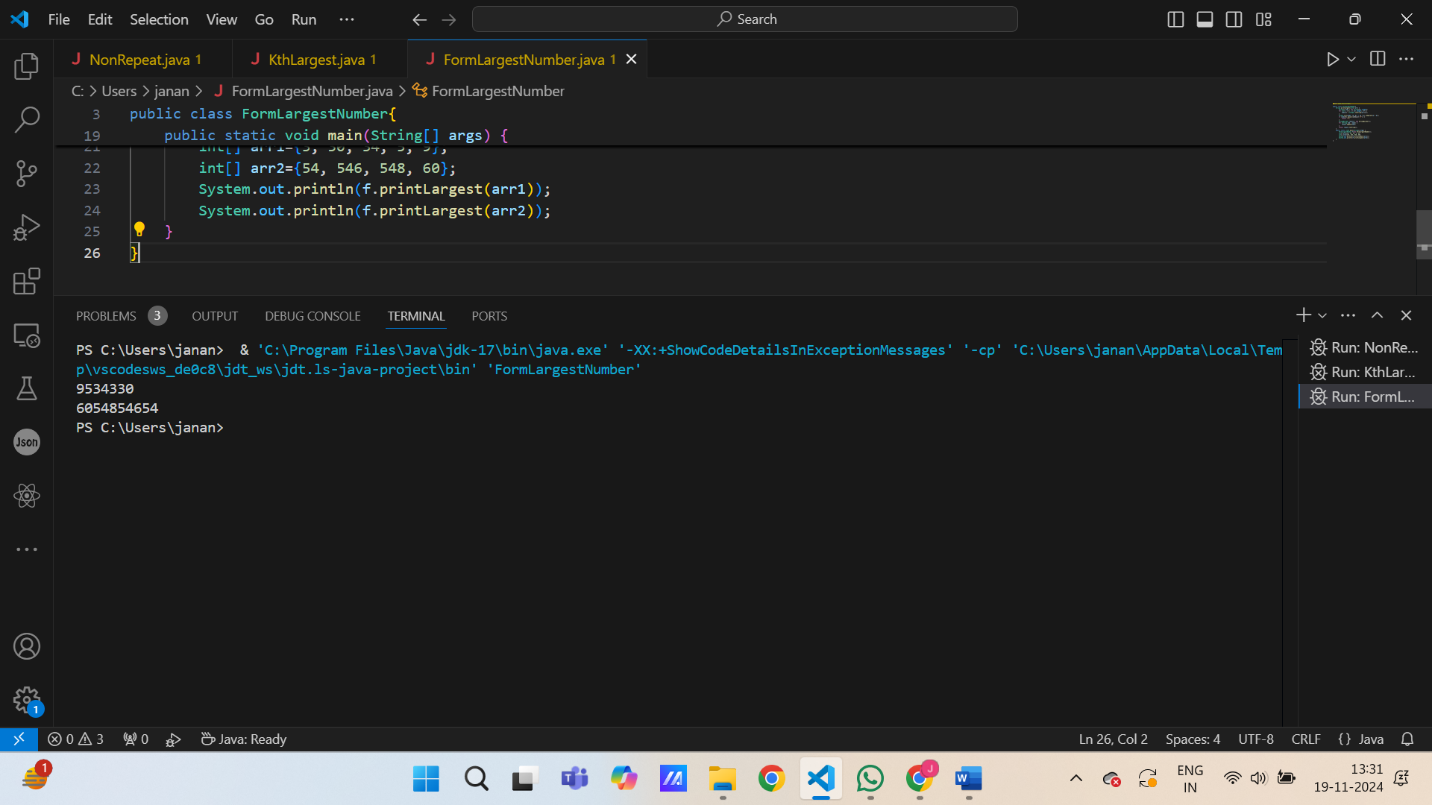
        System.out.println(f.printLargest(arr1));

        System.out.println(f.printLargest(arr2));

    }

}

**OUTPUT:**



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