**GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026**

**DATE: 18-11-2024 NAME: JASHVARTHINI R – CSBS**

1. **BUBBLE SORT**

**PROGRAM:**

package dsaPracticeProblems;

import java.util.Scanner;

class BubbleSort {

public static void bubbleSort(int arr[]) {

int n = arr.length;

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

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

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

static void printArray(int arr[]) {

int n = arr.length;

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

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

}

System.***out***.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("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();

}

*bubbleSort*(arr);

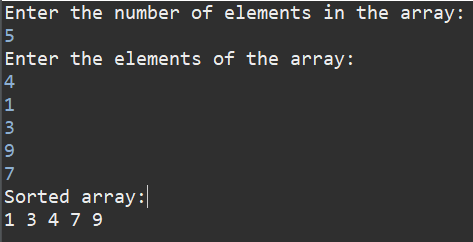
System.***out***.println("Sorted array:");

*printArray*(arr);

}

}

**OUTPUT:**

****

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

1. **QUICK SORT**

**PROGRAM:**

package dsaPracticeProblems;

import java.util.Scanner;

public class BubbleSort {

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

if (low < high) {

int pivotIndex = *partition*(arr, low, high);

*quickSort*(arr, low, pivotIndex - 1);

*quickSort*(arr, pivotIndex + 1, high);

}

}

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

int pivot = arr[high];

int i = low - 1;

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

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("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();

}

*quickSort*(arr, 0, n - 1);

System.***out***.println("Sorted array:");

for (int num : arr) {

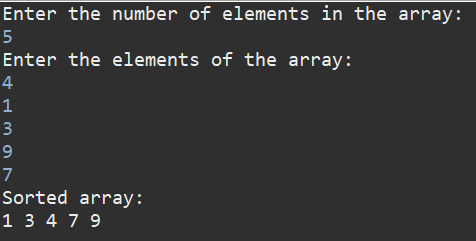
System.***out***.print(num + " ");

}

}

}

**OUTPUT:**

****

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

1. **NON-REPEATING CHARACTERS**

**PROGRAM:**

package dsaPracticeProblems;

import java.util.HashMap;

import java.util.Scanner;

class RepeatingString {

static char nonRepeatingChar(String s) {

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

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

charCount.put(c, charCount.getOrDefault(c, 0) + 1);

}

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

if (charCount.get(c) == 1) {

return c;

}

}

return '$';

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Enter a string:");

String input = scanner.nextLine();

char result = *nonRepeatingChar*(input);

if (result != '$') {

System.***out***.println("The first non-repeating character is: " + result);

} else {

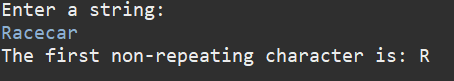
System.***out***.println("No non-repeating character found.");

}

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(n)**

1. **EDIT DISTANCE**

**PROGRAM:**

package dsaPracticeProblems;

import java.util.\*;

public class EditDistance {

public static int editDistRec(String s1, String s2, int m, int n) {

if (m == 0) return n;

if (n == 0) return m;

if (s1.charAt(m - 1) == s2.charAt(n - 1))

return *editDistRec*(s1, s2, m - 1, n - 1);

return 1 + Math.*min*(Math.*min*(*editDistRec*(s1, s2, m, n - 1),

*editDistRec*(s1, s2, m - 1, n)),

*editDistRec*(s1, s2, m - 1, n - 1));

}

public static int editDist(String s1, String s2) {

return *editDistRec*(s1, s2, s1.length(), s2.length());

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the first string (s1):");

String s1 = sc.nextLine();

System.***out***.println("Enter the second string (s2):");

String s2 = sc.nextLine();

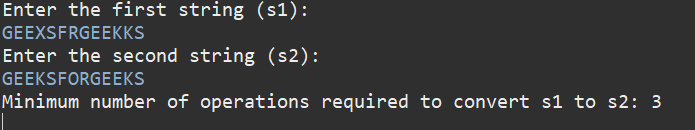
System.***out***.println("Minimum number of operations required to convert s1 to s2: " + *editDist*(s1, s2));

sc.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(3^max(m, n))**

1. **K LARGEST ELEMENTS**

**PROGRAM:**

package dsaPracticeProblems;

import java.util.\*;

class KthLargestElement {

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

Integer[] arrInteger = Arrays.*stream*(arr).boxed().toArray(Integer[]::new);

Arrays.*sort*(arrInteger, Collections.*reverseOrder*());

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

for (int i = 0; i < k; i++)

res.add(arrInteger[i]);

return res;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the number of elements in 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();

}

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

int k = sc.nextInt();

ArrayList<Integer> res = *kLargest*(arr, k);

System.***out***.println("The " + k + " largest elements are:");

for (int ele : res) {

System.***out***.print(ele + " ");

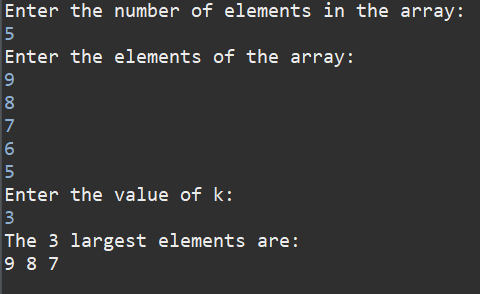
}

sc.close();

}

}

**OUTPUT:**

****

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

1. **FORM LARGEST NUMBERS**

**PROGRAM:**

import java.io.\*;

import java.util.\*;

public class LargestElement {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int t = sc.nextInt();

        sc.nextLine();

        while (t-- > 0) {

            String input = sc.nextLine();

            String[] numbers = input.split(" ");

            int[] arr = new int[numbers.length];

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

                arr[i] = Integer.parseInt(numbers[i]);

            }

            String ans = printLargest(arr);

            System.out.println(ans);

            System.out.println("~");

        }

        sc.close();

    }

    public static String printLargest(int[] arr) {

        String[] strArr = Arrays.stream(arr)

                                .mapToObj(String::valueOf)

                                .toArray(String[]::new);

        Comparator<String> comp = (X, Y) -> (Y + X).compareTo(X + Y);

        Arrays.sort(strArr, comp);

        if (strArr[0].equals("0")) {

            return "0";

        }

        StringBuilder sb = new StringBuilder();

        for (String num : strArr)

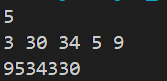
            sb.append(num);

        return sb.toString();

    }

}

**OUTPUT:**

****

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