**Karthick M \_ AI&DS \_ Day - 6 Practice**

**1. Bubble Sort:**

**Given an array, arr[]. Sort the array using bubble sort algorithm.**

**Examples :**

**Input: arr[] = [4, 1, 3, 9, 7]**

**Output: [1, 3, 4, 7, 9]**

**Program:**

import java.util.Scanner;

public class BubbleSort {

public static void bubbleSort(int[] arr) {

int n = arr.length;

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

boolean swap = false;

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;

swap = true;

}

}

if (!swap) break;

}

}

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();

}

bubbleSort(arr);

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

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

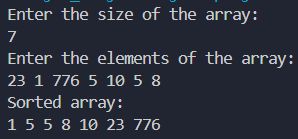
System.out.print(arr[i] + " ");

}

}

}

**Output:**

****

**Time Complexity:** Best Case: O(n), Worst Case: O(n^2)

**2. Quick Sort:**

**Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, arr[] in ascending order. Given an array, arr[], with starting index low and ending index high, complete the functions partition() and quickSort(). Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.**

**Note: The low and high are inclusive.**

**Examples:**

**Input: arr[] = [4, 1, 3, 9, 7]**

**Output: [1, 3, 4, 7, 9]**

**Explanation: After sorting, all elements are arranged in ascending order.**

**Program:**

import java.util.Scanner;

public class QuickSort {

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

int pivot = arr[low];

int i = low + 1;

int j = high;

while (i <= j) {

while (i <= high && arr[i] <= pivot) {

i++;

}

while (j >= low && arr[j] > pivot) {

j--;

}

if (i < j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[low];

arr[low] = arr[j];

arr[j] = temp;

return j;

}

private 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);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements:");

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

arr[i] = scanner.nextInt();

}

quickSort(arr, 0, n - 1);

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

for (int value : arr) {

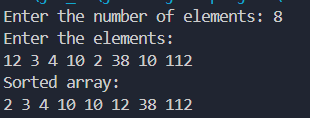
System.out.print(value + " ");

}

}

}

**Output:**

****

**Time Complexity:** O(nlogn)

**3. Non Repeating Character:**

**Program:**

import java.util.\*;

public class NonRepeatingCharacter {

public static char firstNonRepeatingCharacter(String s) {

Map<Character, Integer> frequencyMap = new HashMap<>();

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

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

}

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

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

return c;

}

}

return '$';

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String s = sc.nextLine();

char result = firstNonRepeatingCharacter(s);

System.out.println("First non-repeating character: " + (result == '$' ? "-1" : result));

sc.close();

}

}

**Output:**

****

**Time Complexity:** O(n)

**5. K Largest Elements:**

**Given an array arr[] of positive integers and an integer k, Your task is to return k largest elements in decreasing order.**

**Examples**

**Input: arr[] = [12, 5, 787, 1, 23], k = 2**

**Output: [787, 23]**

**Explanation: 1st largest element in the array is 787 and second largest is 23.**

**Program:**

import java.util.\*;

public class KLargestElement {

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

PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

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

maxHeap.add(arr[i]);

}

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

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

list.add(maxHeap.poll());

}

return list;

}

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();

}

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

int k = sc.nextInt();

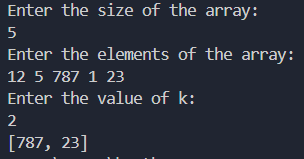
List<Integer> result = kLargest(arr, k);

System.out.println(result);

}

}

**Output:**

****

**Time Complexity:** O(nlogn)

**6. Form the largest number:**

**Given an array of integers arr[] representing non-negative integers, arrange them so that after concatenating all of them in order, it results in the largest possible number. Since the result may be very large, return it as a string.**

**Examples:**

**Input: arr[] = [3, 30, 34, 5, 9]**

**Output: "9534330"**

**Explanation: Given numbers are {3, 30, 34, 5, 9}, the arrangement "9534330" gives the largest value.**

**Program:**

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

public class LargestNumber {

public static String largestNumber(int[] nums) {

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

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

strNums[i] = String.valueOf(nums[i]);

}

Arrays.sort(strNums, new Comparator<String>() {

@Override

public int compare(String x, String y) {

return (y + x).compareTo(x + y);

}

});

if (strNums[0].equals("0")) {

return "0";

}

StringBuilder result = new StringBuilder();

for (String num : strNums) {

result.append(num);

}

return result.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements:");

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

arr[i] = scanner.nextInt();

}

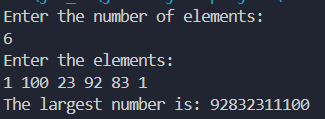
System.out.println("The largest number is: " + largestNumber(arr));

scanner.close();

}

}

**Output:**

****

**Time Complexity:** O(nlogn)