Naresh M\_AI&DS\_DSA\_Day-6

**1.Kth 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.

**Input:** arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3

**Output:** [50, 30, 23]

**Explanation:** Three Largest elements in the array are 50, 30 and 23.

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

**Output:** [23]

**Explanation:** 1st Largest element in the array is 23.

**Constraints:**  
1 ≤ k ≤ arr.size() ≤ 106  
1 ≤ arr[i] ≤ 106

**Program:**

import java.util.\*;

public class KLargestElements {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the array: ");

        int n = sc.nextInt();

        int[] arr = new int[n];

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

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

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

        int k = sc.nextInt();

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

        for (int num : arr) {

            minHeap.add(num);

            if (minHeap.size() > k) {

                minHeap.poll();

            }

        }

        List<Integer> result = new ArrayList<>(minHeap);

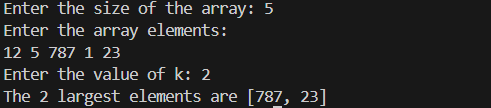
        result.sort(Collections.reverseOrder());

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

        sc.close();

    }

}



**Time complexity : O(n)**

**2.Buble 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]

**Input**: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

**Output**: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**Input**: arr[] = [1, 2, 3, 4, 5]

**Output**: [1, 2, 3, 4, 5]  
**Explanation**: An array that is already sorted should remain unchanged after applying bubble sort.

**Constraints:**  
1 <= arr.size() <= 103  
1 <= arr[i] <= 103

**Program:**

class Solution {

// Function to sort the array using bubble sort algorithm.

public static void bubbleSort(int arr[]) {

int n = arr.length;

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

boolean swaped = 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;

swaped = true;

}

}

if (!swaped) break;

}

}

}



**Time Complexity: O(n2)**

**3.Non Repeating Character**

Given a string **s** consisting of **lowercase**Latin Letters. Return the first non-repeating character in **s**. If there is no non-repeating character, return **'$'.**  
Note:When you return '$' driver code will output -1.

**Examples:**

**Input:** s = "geeksforgeeks"

**Output:** 'f'

**Explanation:** In the given string, 'f' is the first character in the string which does not repeat.

**Input:** s = "racecar"  
**Output:** 'e'  
**Explanation:** In the given string, 'e' is the only character in the string which does not repeat.

**Input:** s = "aabbccc"  
**Output:** '$'  
**Explanation:** All the characters in the given string are repeating.

**Program:**

class Solution {

// Function to find the first non-repeating character in a string.

static char nonRepeatingChar(String s) {

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

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

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

}

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

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

return c;

}

}

return '$';

}

}



**Time Complexity: O(n)**

**4. 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.

public class QuickSort {

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++; // Increment index of the smaller element

// Swap arr[i] and arr[j]

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;

}

static void printArray(int arr[]) {

for (int value : arr) {

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

}

System.out.println();

}

public static void main(String[] args) {

int arr[] = {10, 80, 30, 90, 40, 50, 70};

int n = arr.length;

System.out.println("Original Array:");

printArray(arr);

quickSort(arr, 0, n - 1);

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

printArray(arr);

}

}



**Time Complexity: O(n log n)**

**5. Form the Largest Number**

Difficulty: MediumAccuracy: 37.82%Submissions: 162K+Points: 4

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

Note: There are no leading zeros in each array element.

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.

**Code:**

import java.util.Arrays;

public class LargestNumber {

public static String printLargest(String[] arr) {

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

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

return "0";

}

StringBuilder result = new StringBuilder();

for (String s : arr) {

result.append(s);

}

return result.toString();

}

public static void main(String[] args) {

String[] arr1 = {"3", "30", "34", "5", "9"};

String[] arr2 = {"10", "2"};

String[] arr3 = {"0", "0"};

for(String s : arr1){

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

}

System.out.println();

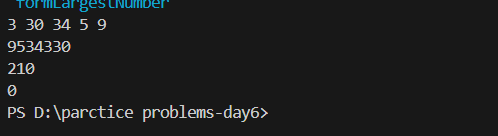
System.out.println(printLargest(arr1)); // Output: 9534330

System.out.println(printLargest(arr2)); // Output: 210

System.out.println(printLargest(arr3)); // Output: 0

}

}



**Time Complexity: O(n log n)**

**6.Edit Distance**

Given two strings s1 and s2. Return the minimum number of operations required to convert s1 to s2.

The possible operations are permitted:

Insert a character at any position of the string.

Remove any character from the string.

Replace any character from the string with any other character.

Examples:

Input: s1 = "geek", s2 = "gesek"

Output: 1

Explanation: One operation is required, inserting 's' between two 'e'.

Input : s1 = "gfg", s2 = "gfg"

Output: 0

Explanation: Both strings are same.

Input : s1 = "abc", s2 = "def"

Output: 3

Explanation: All characters need to be replaced to convert str1 to str2, requiring 3 replacement operations.

**Code:**

public class editDistance {

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

        int m = s1.length(); // Length of string s1

        int n = s2.length(); // Length of string s2

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

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

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

                if (i == 0) {

                    dp[i][j] = j;

                } else if (j == 0) {

                    dp[i][j] = i;

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

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

                } else {

                    dp[i][j] = 1 + Math.min(

                            dp[i - 1][j],      // Deletion

                            Math.min(

                                    dp[i][j - 1],  // Insertion

                                    dp[i - 1][j - 1] // Substitution

                            )

                    );

                }

            }

        }

        return dp[m][n];

    }

    public static void main(String[] args) {

        String s1 = "geek";

        String s2 = "gesek";

        int distance = editDistance(s1, s2);

        System.out.println("Edit Distance between " + s1 + "and " + s2 + ": " + distance);

    }

}

**Time Complexity: O(mxn)**

