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]

**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 {

public static void bubbleSort(int arr[]) {

int n = arr.length;

int[] k = Arrays.copyOf(arr, n);

Arrays.sort(k);

if (Arrays.equals(k, arr)) return;

for (int i = 0; i < n; 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;

}

}

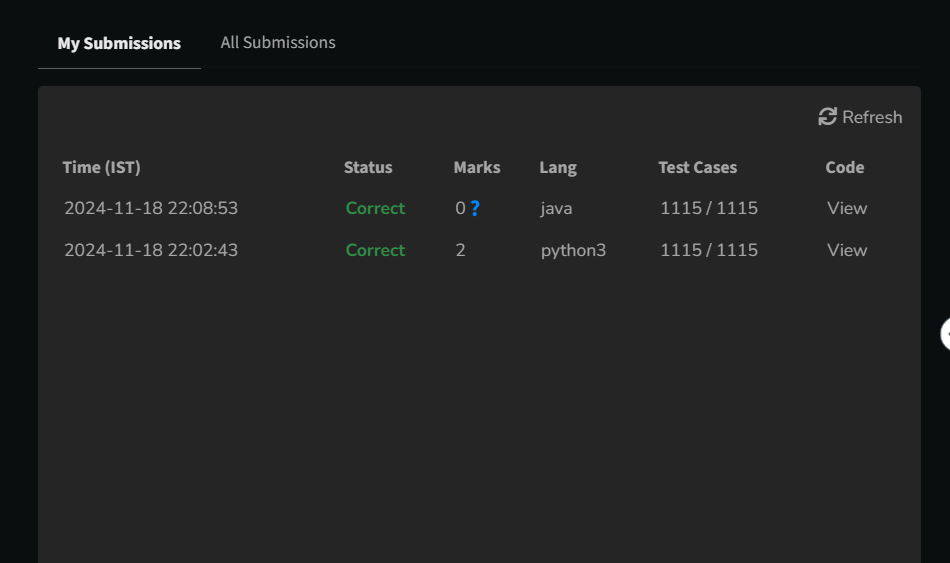
}

return;

}

}

Output:



Time Complexity: O(N2)

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.

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

**Output: [**1, 1, 2, 3, 4, 6, 7, 9, 10]  
**Explanation:** Duplicate elements (1) are retained in sorted order.

**Input:** arr[] = [5, 5, 5, 5]

**Output:** [5, 5, 5, 5]  
**Explanation:** All elements are identical, so the array remains unchanged.

**Constraints:**  
1 <= arr.size() <= 103  
1 <= arr[i] <= 104

Program:

class Solution {

static void quickSort(int arr[], int l, int h) {

if (l < h) {

int p = partition(arr, l, h);

quickSort(arr, l, p - 1);

quickSort(arr, p + 1, h);

}

}

static int partition(int arr[], int l, int h) {

int p = arr[h];

int i = l - 1;

for (int j = l; j < h; j++) {

if (arr[j] <= p) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

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

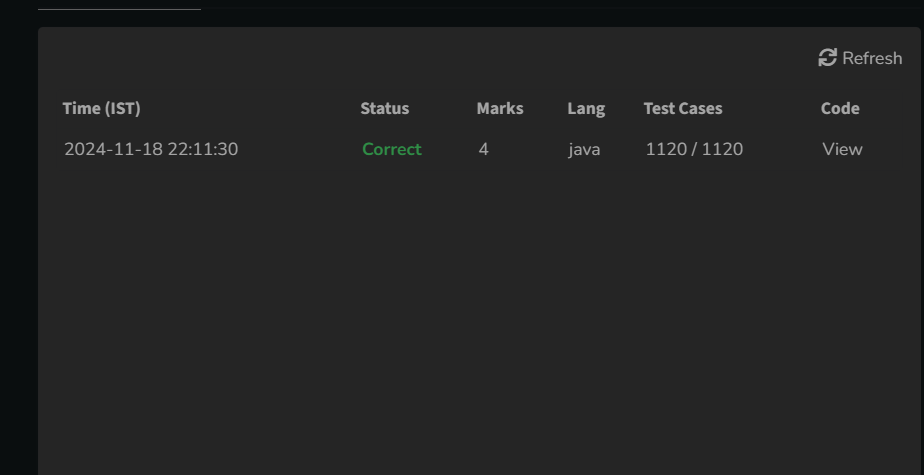
arr[h] = temp;

return i + 1;

}

}

Output:



Time Complexity : O(N)- average case

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.

**Constraints:**  
1 <= s.size() <= 105

Program:

class Solution {

static char nonRepeatingChar(String s) {

int[] c = new int[26];

for (char ch : s.toCharArray()) c[ch - 'a']++;

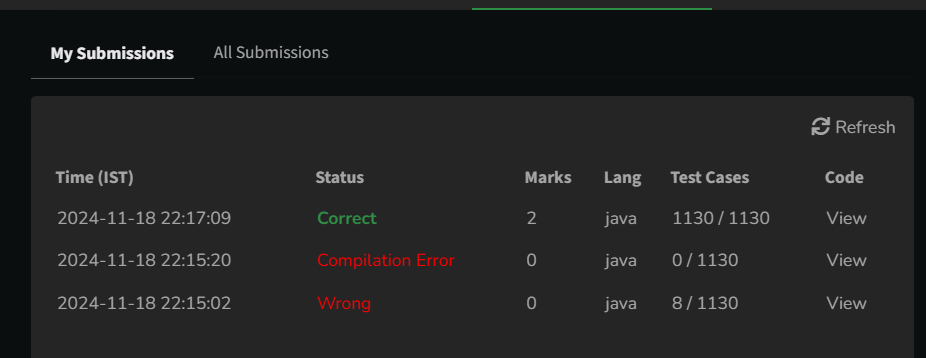
for (char ch : s.toCharArray()) if (c[ch - 'a'] == 1) return ch;

return '$';

}

}

Output:



Time Complexity: O(N)

4.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:

1. Insert a character at any position of the string.
2. Remove any character from the string.
3. 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.

**Constraints:**  
1 ≤ s1.length(), s2.length() ≤ 500  
both the strings are in lowercase.

Program:

class Solution {

public int editDistance(String s1, String s2) {

int m = s1.length(), n = s2.length();

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

for (int i = 0; i <= m; i++) dp[i][0] = i;

for (int j = 0; j <= n; j++) dp[0][j] = j;

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

for (int j = 1; j <= n; 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], dp[i][j - 1]), dp[i - 1][j - 1]) + 1;

}

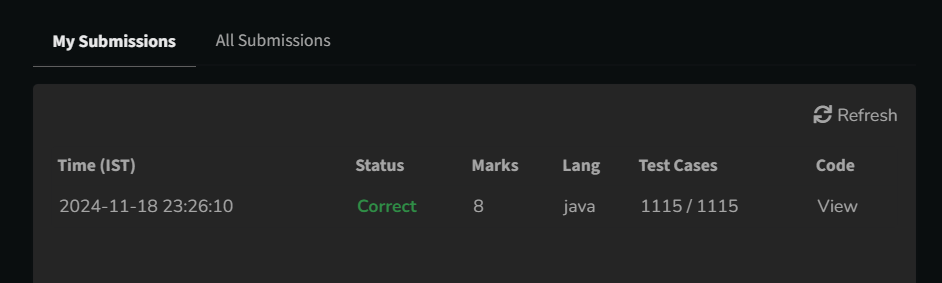
}

}

return dp[m][n];

}

}  
Output:



Time Complexity: O(M\*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.

**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:

class Solution {

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

Arrays.sort(arr);

int n= arr.length;

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

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

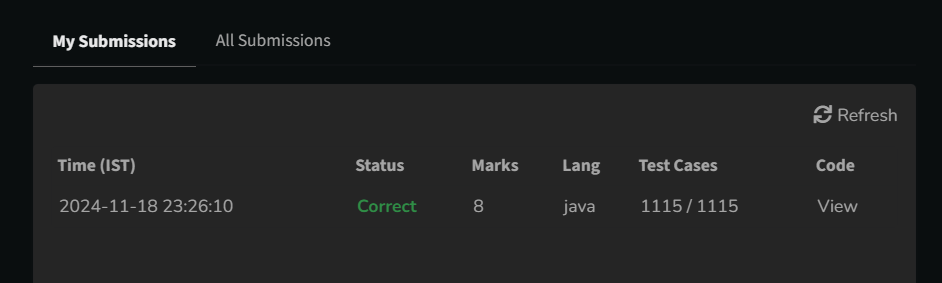
result.add(arr[i]);

}

return result;

}

}  
Output:



Time Complexity: O(N log N)

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.

**Input:** arr[] = [54, 546, 548, 60]

**Output:** "6054854654"

**Explanation:** Given numbers are {54, 546, 548, 60}, the arrangement "6054854654" gives the largest value.

**Input:** arr[] = [3, 4, 6, 5, 9]

**Output:** "96543"

**Explanation:** Given numbers are {3, 4, 6, 5, 9}, the arrangement "96543" gives the largest value.

**Constraints:**  
1 ≤ arr.size() ≤ 105  
0 ≤ arr[i] ≤ 105  
The sum of all the elements of the array is greater than 0.

Try more examples

Program:

class Solution {

public String printLargest(int[] arr) {

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

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

nums[i] = String.valueOf(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) {

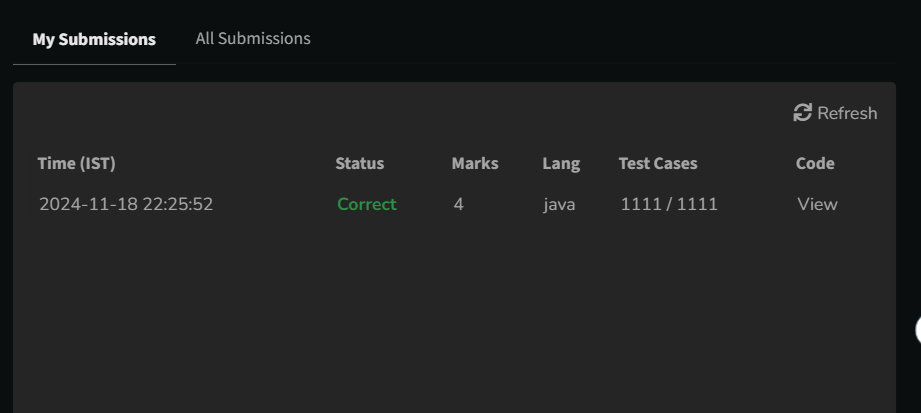
result.append(num);

}

return result.toString();

}

}  
Output:



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