**NON REPEATING CHARACTERS**

**CODE :**

class Solution {

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

static char nonRepeatingChar(String s) {

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

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

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

}

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

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

return c;

}

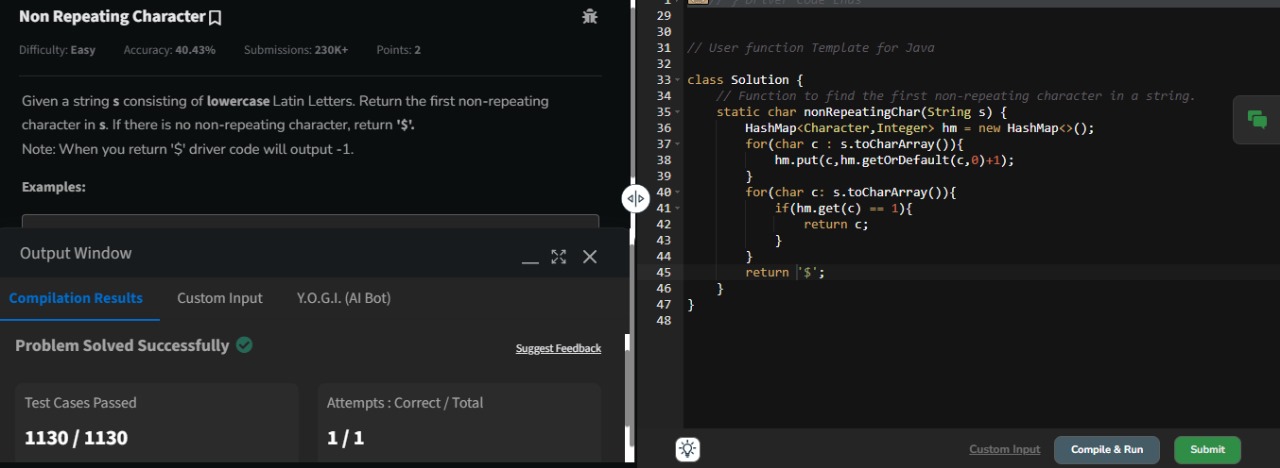
}

return '$';

}

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

**K LARGEST ELEMENTS**

**CODE :**

class Solution {

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

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

Arrays.sort(arr);

for(int i = arr.length - 1 ; i>arr.length -1 -k ; i--){

l.add(arr[i]);

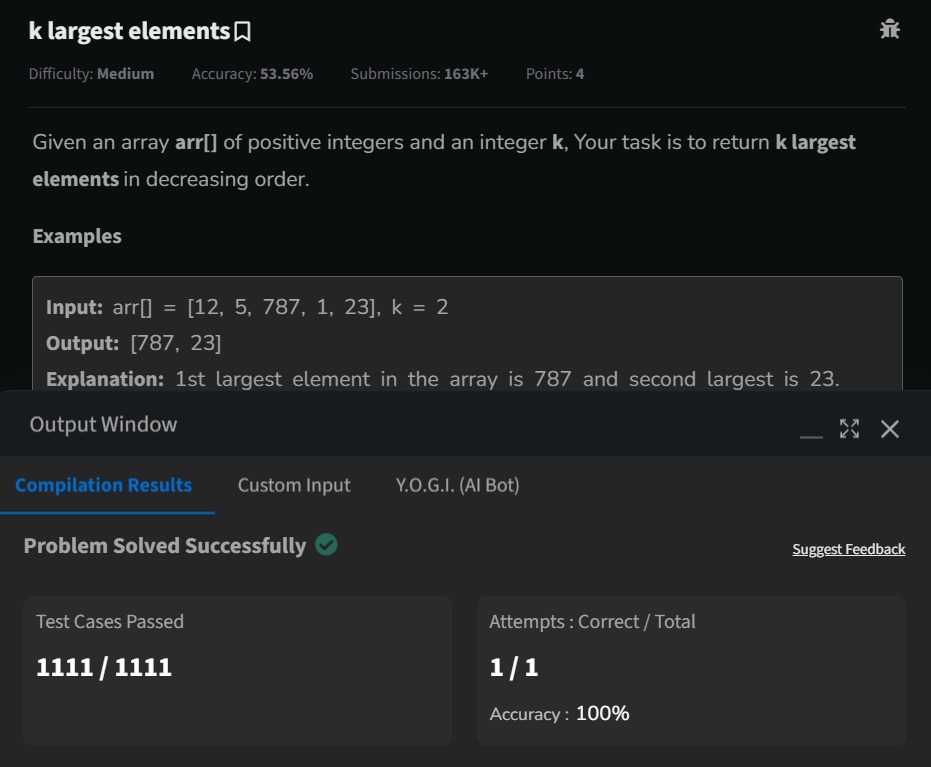
}

return l;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(nlogn)

SPACE COMPLEXITY : O(n)

**BUBBLE SORT :**

**CODE :**

class Solution {

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-1 ; j++){

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

int temp = arr[j];

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

arr[j+1] = temp;

swap = true;

}

}

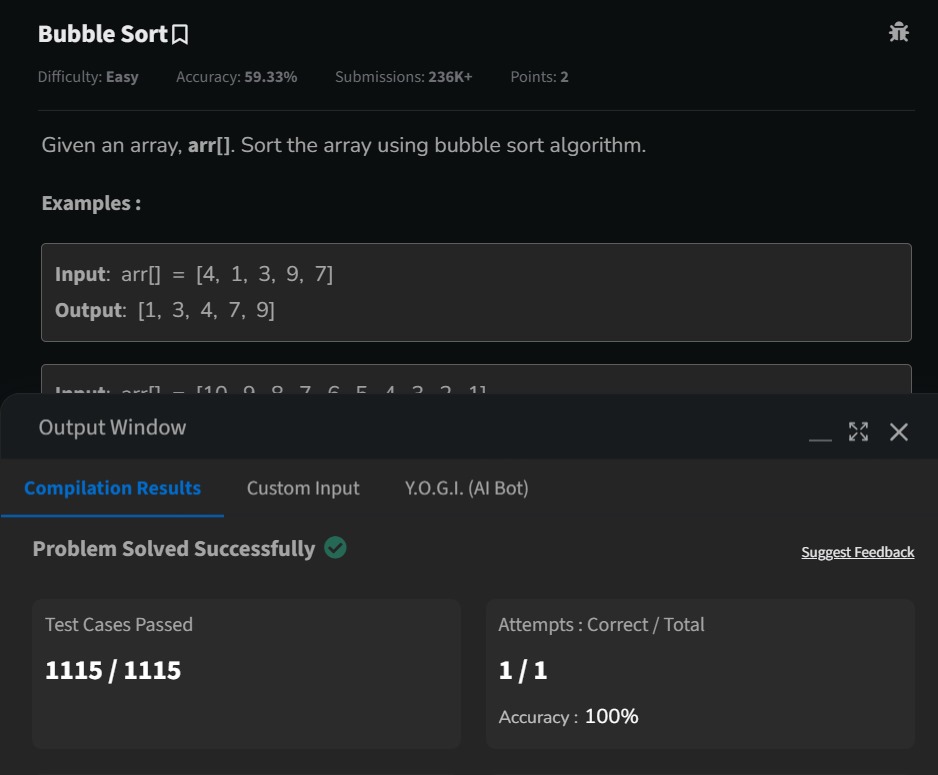
if(!swap) break;

}

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n^2)

SPACE COMPLEXITY : O(1)

**QUICK SORT :**

**CODE :**

class Solution {

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 pivot = arr[high];

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

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];

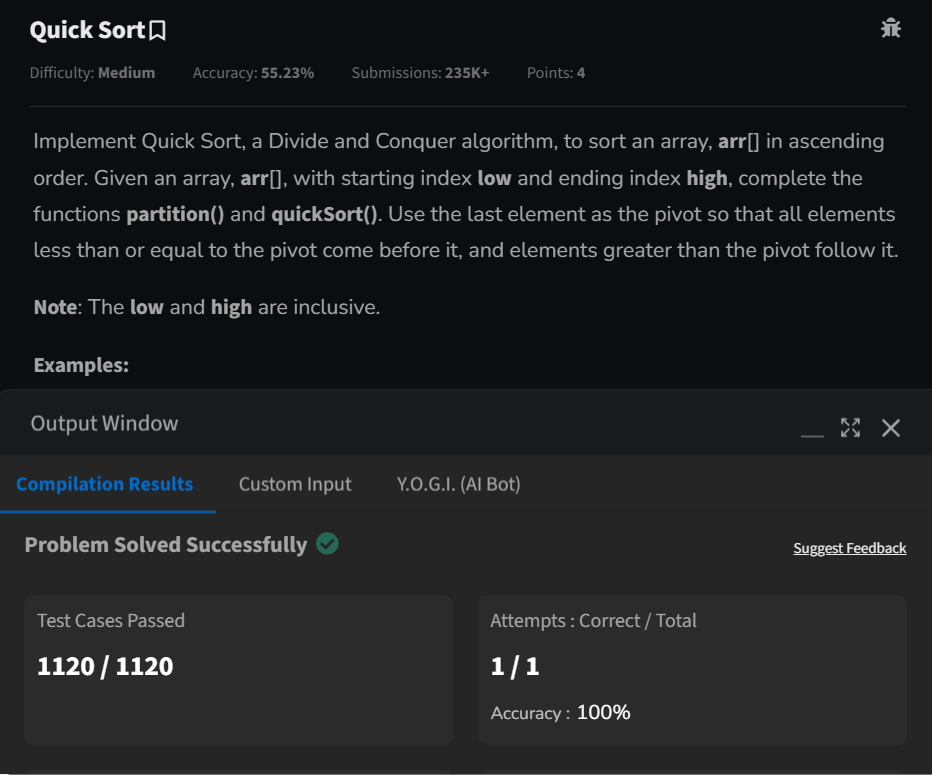
arr[i] = arr[j];

arr[j] = temp;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(nlogn)

SPACE COMPLEXITY : O(logn)

**EDIT DISTANCE :**

**CODE :**

class Solution {

public int editDistance(String str1, String str2) {

int m = str1.length();

int n = str2.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(str1.charAt(i-1)==str2.charAt(j-1))

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

else

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

}

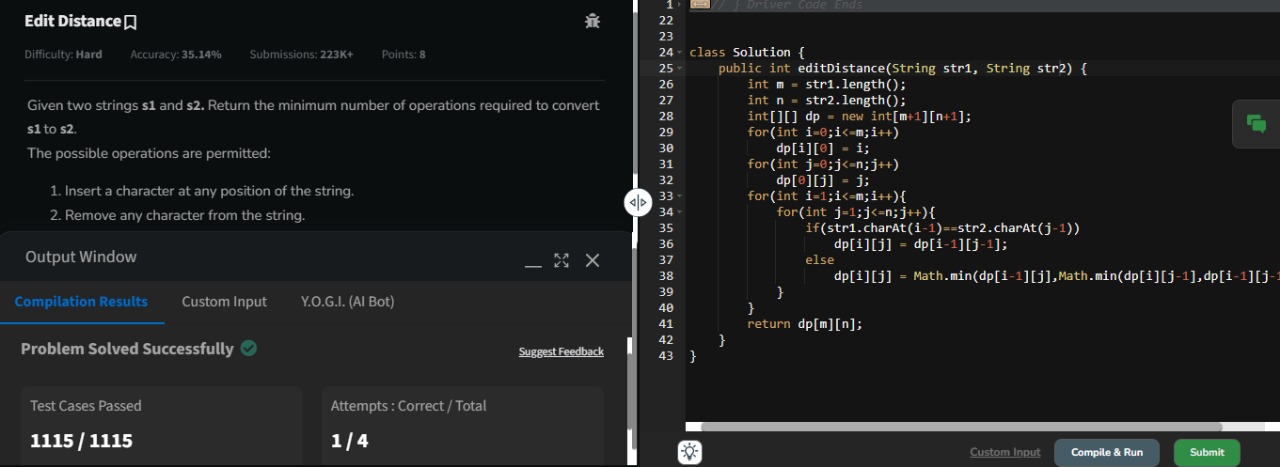
}

return dp[m][n];

}

}

**OUTPUT :**



TIME COMPLEXITY : O(m\*n)

SPACE COMPLEXITY : O(m\*n)

**FORM THE LARGEST NUMBER :**

**CODE :**

class Solution {

String printLargest(int[] array) {

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

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

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

}

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

if (arr[0]=="0"){

return "0";

}

StringBuilder result = new StringBuilder();

for (String numStr : arr) {

result.append(numStr);

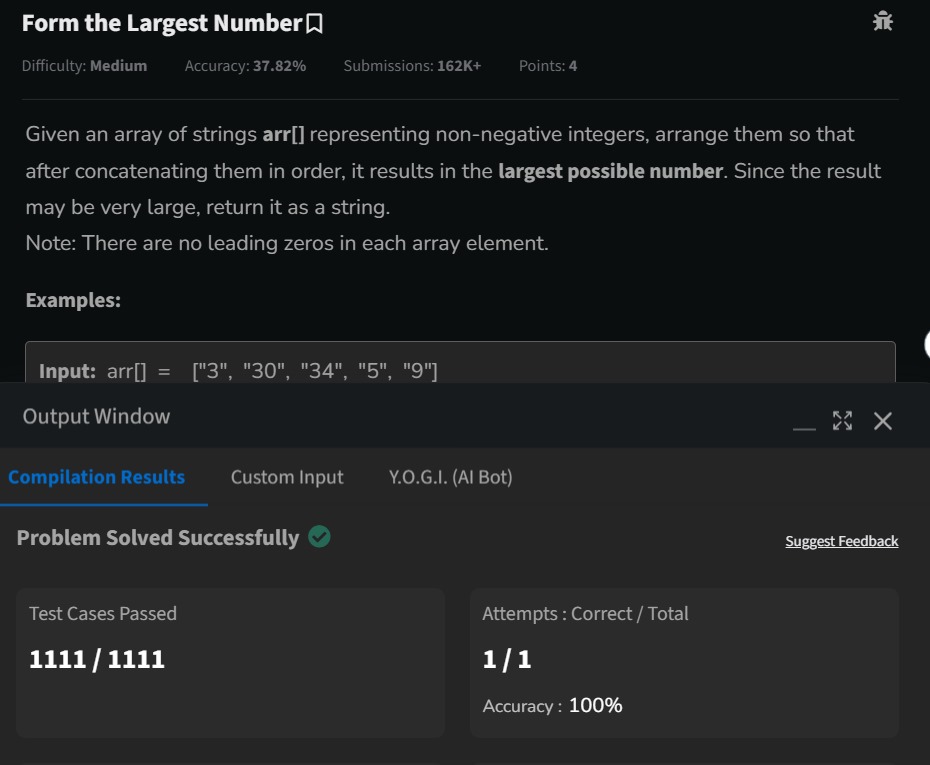
}

return result.toString();

}

}

**OUTPUT :**



TIME COMPLEXITY : O(nlogn \* m)

SPACE COMPLEXITY : O(n)