DSA_PROBLEMS

18/11/2024

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

Examples :

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

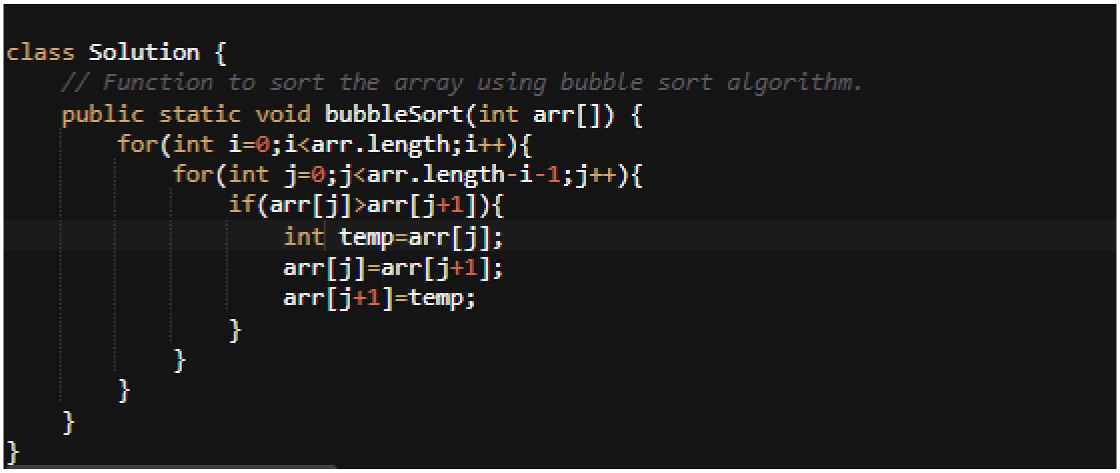
Output: [1, 3, 4, 7, 9]

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

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

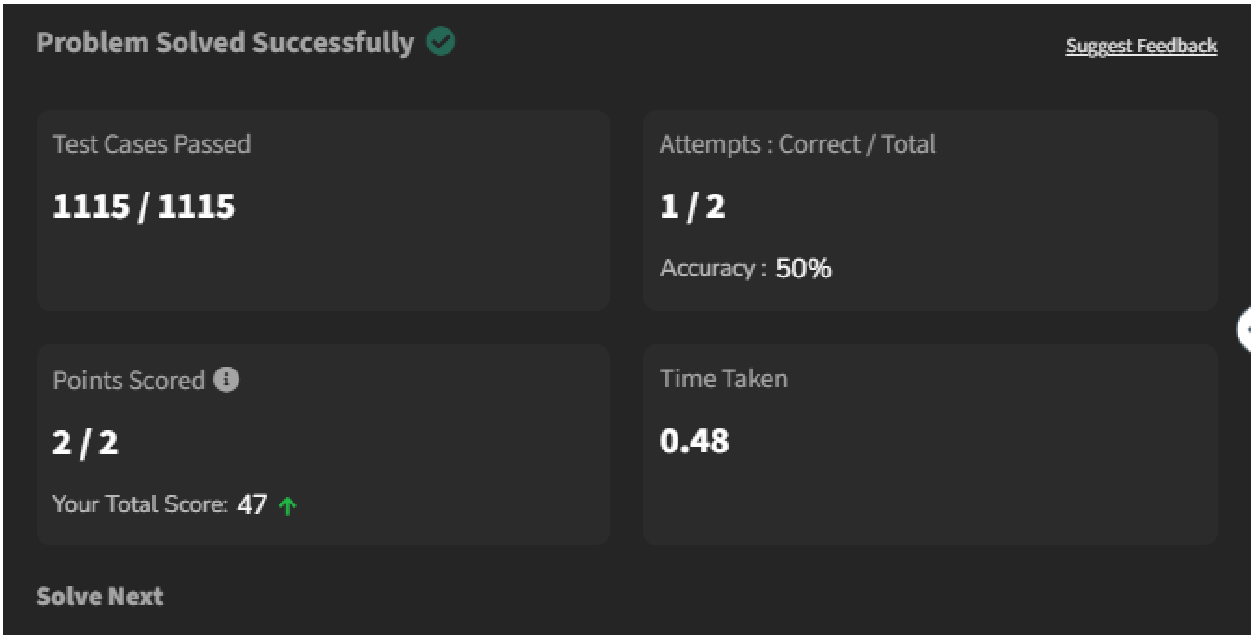
Worst-Case Time Complexity: O(n2)

Code:



```
class Solution {
    // Function to sort the array using bubble sort algorithm.
    public static void bubbleSort(int arr[]) {
        for(int i=0;i<arr.length;i++){
            for(int j=0;j<arr.length-i-1;j++){
                if(arr[j]>arr[j+1]){
                    int temp=arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
                }
            }
        }
    }
}
```

Output:



Problem Solved Successfully ✓

Test Cases Passed

1115 / 1115

Attempts : Correct / Total

1 / 2

Accuracy : 50%

Points Scored

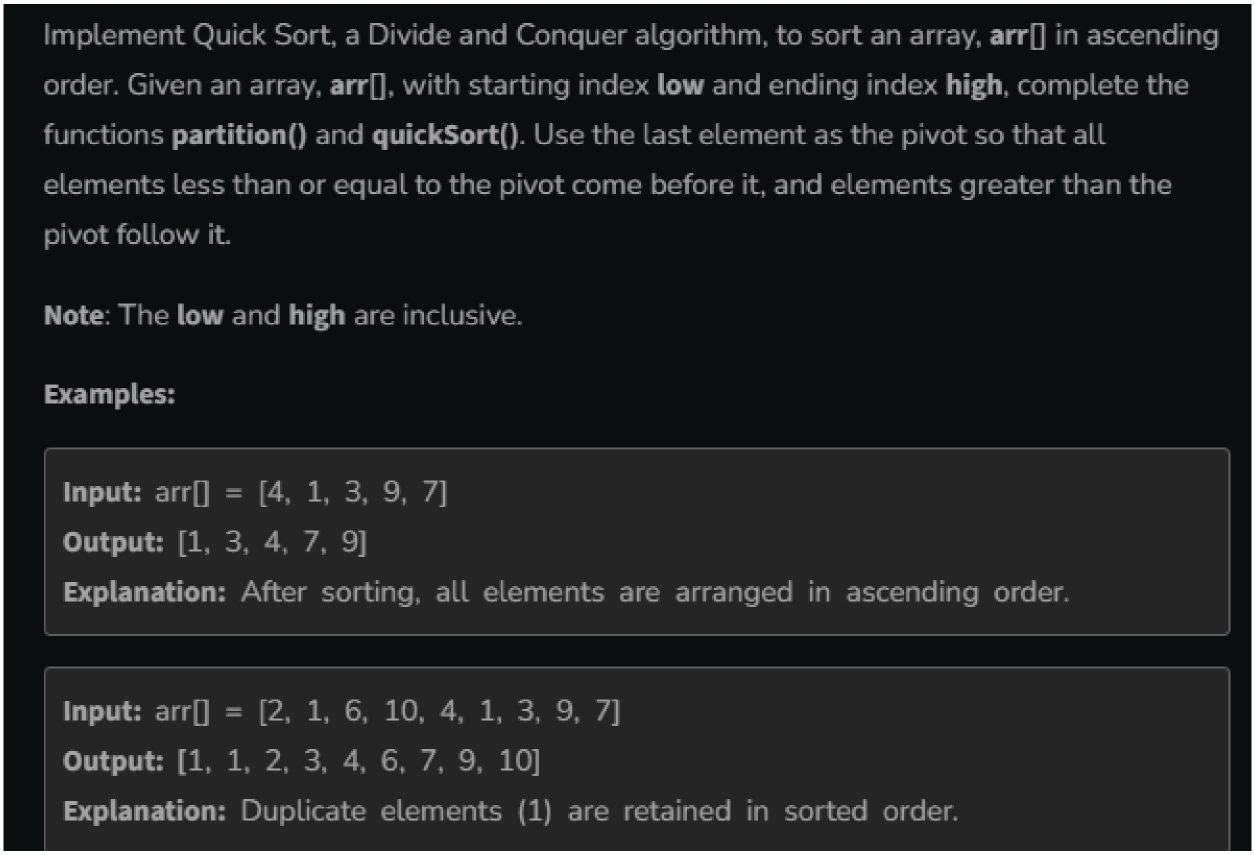
2 / 2

Your Total Score: 47 ↑

Time Taken

0.48

2.



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.

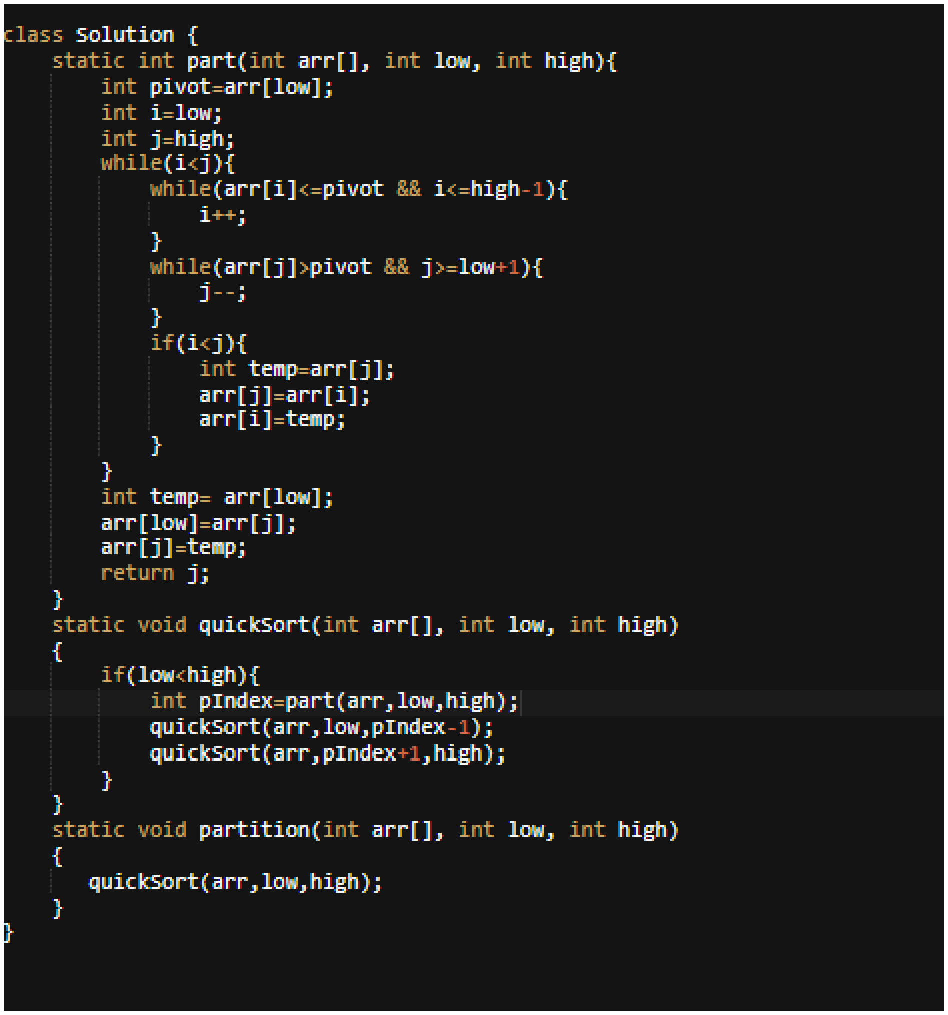
Note: The **low** and **high** are inclusive.

Examples:

```
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.
```

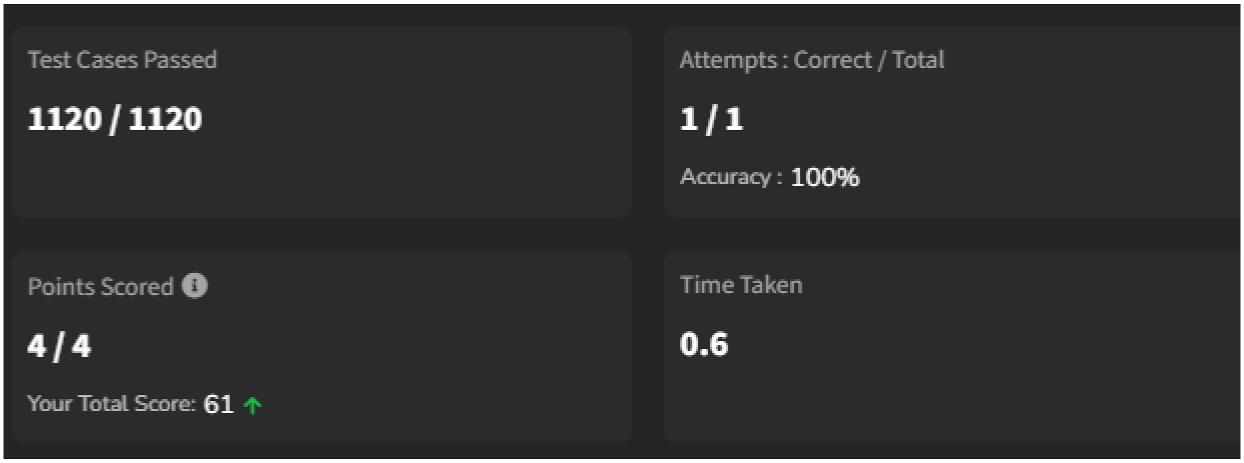
```
Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]
Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.
```

Code:



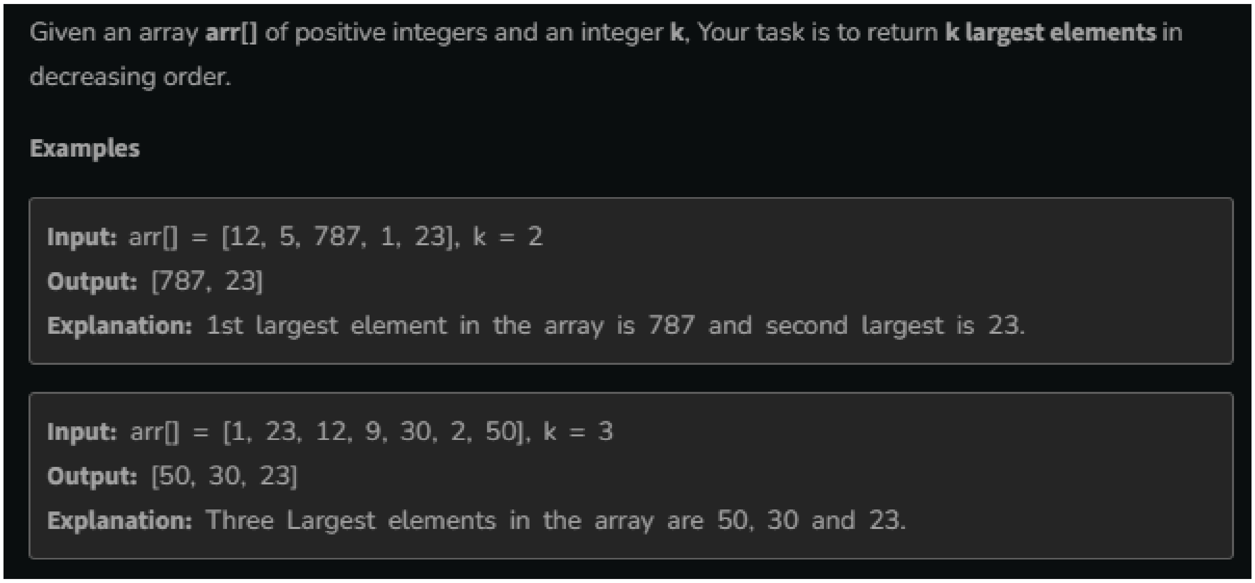
```java
class Solution {
    static int part(int arr[], int low, int high){
        int pivot=arr[low];
        int i=low;
        int j=high;
        while(i<j){
            while(arr[i]<=pivot && i<=high-1){
                i++;
            }
            while(arr[j]>pivot && j>=low+1){
                j--;
            }
            if(i<j){
                int temp=arr[j];
                arr[j]=arr[i];
                arr[i]=temp;
            }
        }
        int temp= arr[low];
        arr[low]=arr[j];
        arr[j]=temp;
        return j;
    }
    static void quickSort(int arr[], int low, int high)
    {
        if(low<high){
            int pIndex=part(arr,low,high);
            quickSort(arr,low,pIndex-1);
            quickSort(arr,pIndex+1,high);
        }
    }
    static void partition(int arr[], int low, int high)
    {
       quickSort(arr,low,high);
    }
}
```

Output:



```
Test Cases Passed                 Attempts : Correct / Total

1120 / 1120                       1 / 1

                                  Accuracy : 100%

Points Scored                     Time Taken

4 / 4                             0.6

Your Total Score: 61 ↑
```

3.



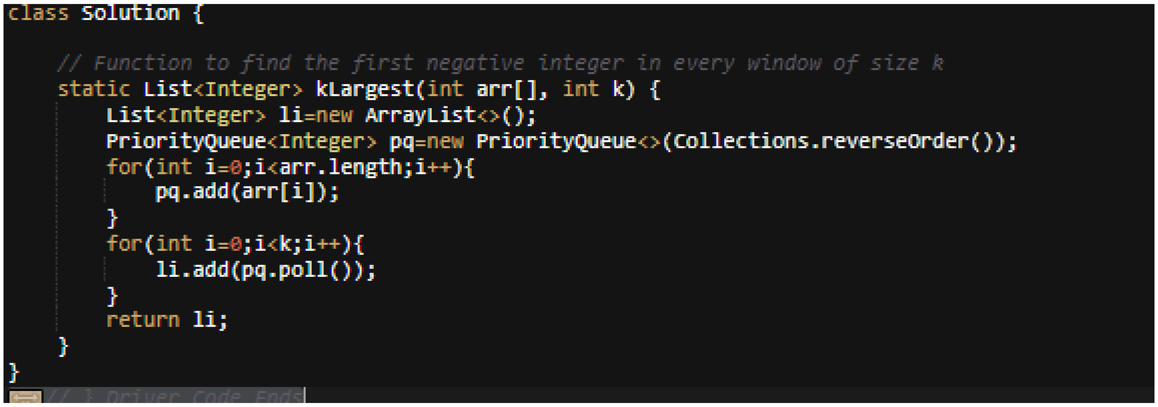
Given an array **arr[]** of positive integers and an integer **k**, Your task is to return **k largest elements** in decreasing order.

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.

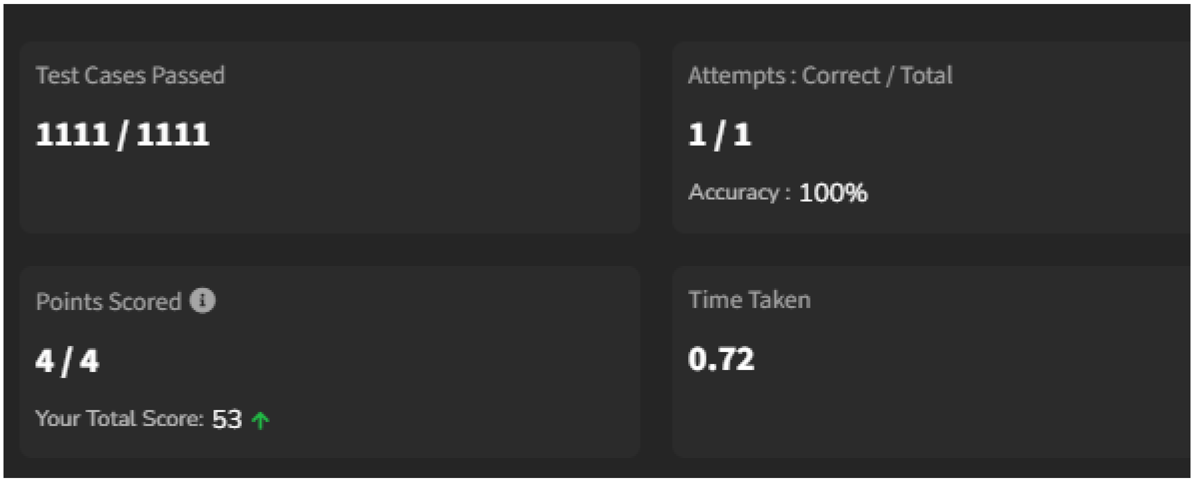
Input: arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3
Output: [50, 30, 23]
Explanation: Three Largest elements in the array are 50, 30 and 23.

Code:



```java
class Solution {

    // Function to find the first negative integer in every window of size k
    static List<Integer> kLargest(int arr[], int k) {
        List<Integer> li=new ArrayList<>();
        PriorityQueue<Integer> pq=new PriorityQueue<>(Collections.reverseOrder());
        for(int i=0;i<arr.length;i++){
            pq.add(arr[i]);
        }
        for(int i=0;i<k;i++){
            li.add(pq.poll());
        }
        return li;
    }
}
// } Driver Code Ends
```

Output:



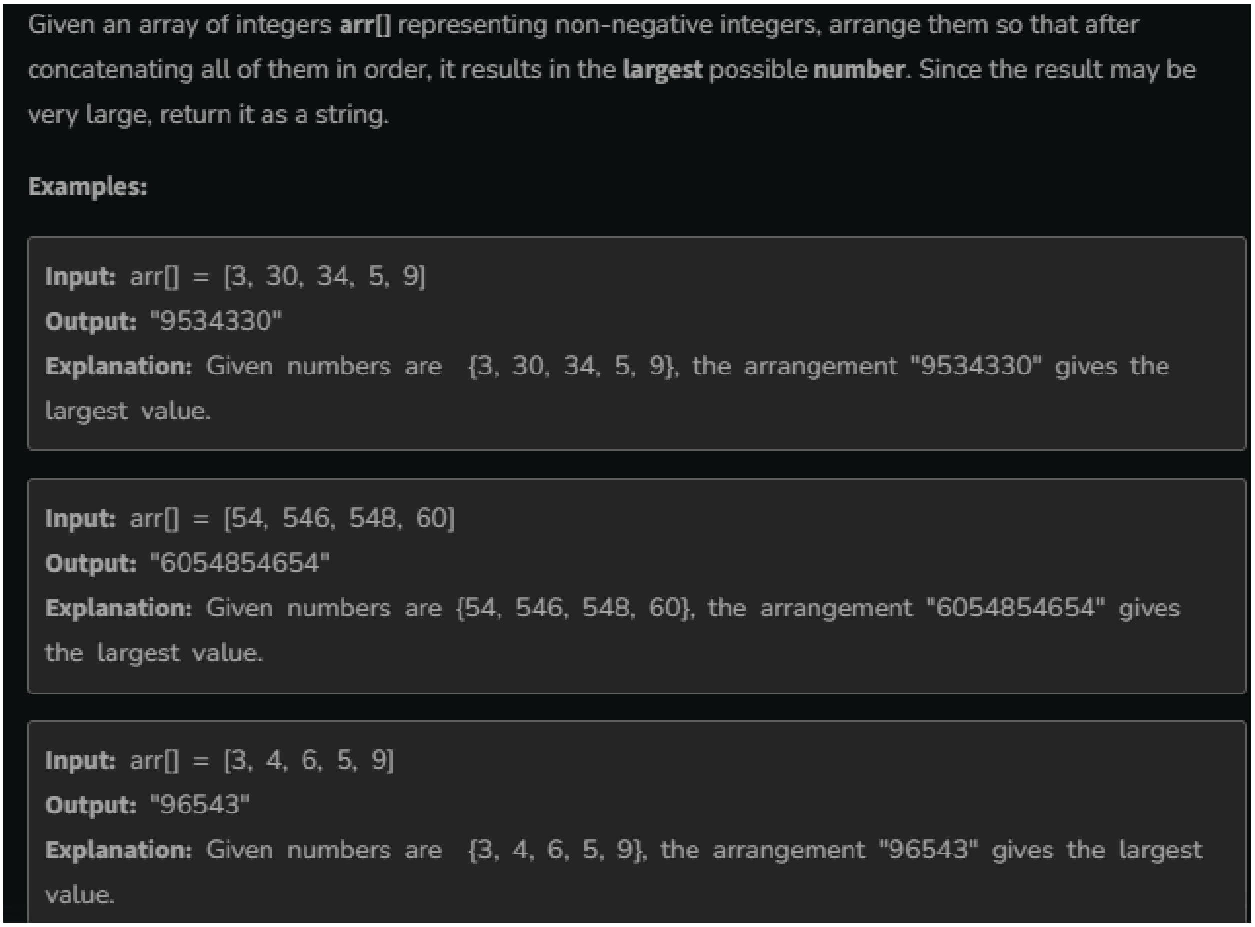
Test Cases Passed
1111 / 1111

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
4 / 4
Your Total Score: 53 ↑

Time Taken
0.72

4.



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.

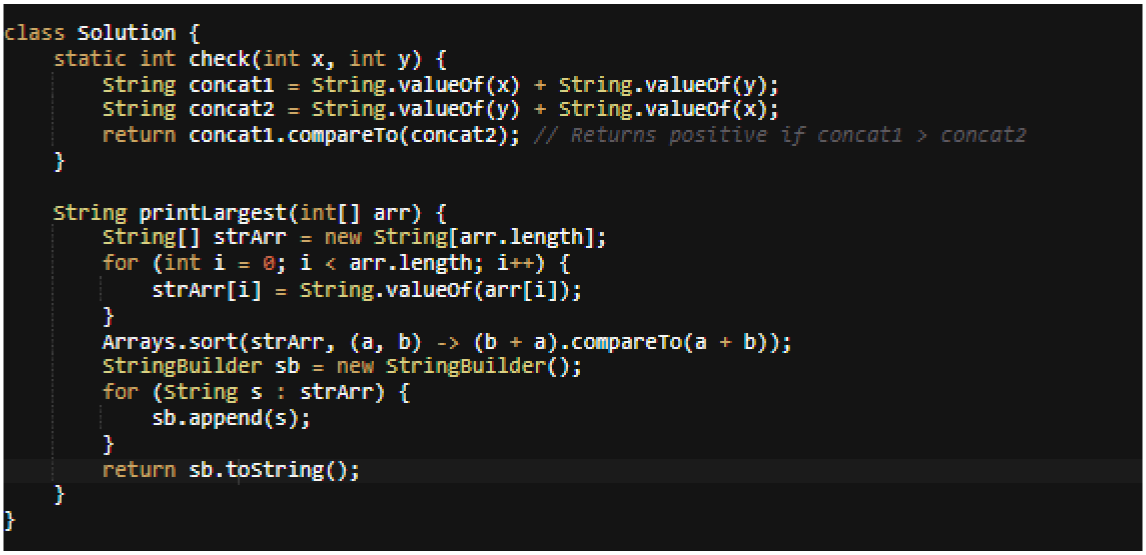
Examples:

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.

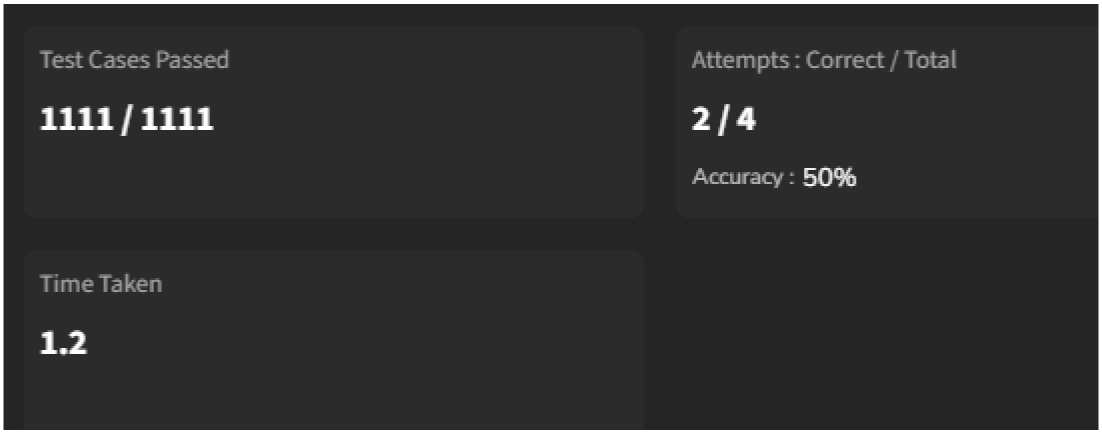
Input: arr[] = [3, 4, 6, 5, 9]
Output: "96543"
Explanation: Given numbers are {3, 4, 6, 5, 9}, the arrangement "96543" gives the largest value.

Code:



```java
class Solution {
    static int check(int x, int y) {
        String concat1 = String.valueOf(x) + String.valueOf(y);
        String concat2 = String.valueOf(y) + String.valueOf(x);
        return concat1.compareTo(concat2); // Returns positive if concat1 > concat2
    }

    String printLargest(int[] arr) {
        String[] strArr = new String[arr.length];
        for (int i = 0; i < arr.length; i++) {
            strArr[i] = String.valueOf(arr[i]);
        }
        Arrays.sort(strArr, (a, b) -> (b + a).compareTo(a + b));
        StringBuilder sb = new StringBuilder();
        for (String s : strArr) {
            sb.append(s);
        }
        return sb.toString();
    }
}
```

Output:


Test Cases Passed

1111 / 1111

Attempts : Correct / Total

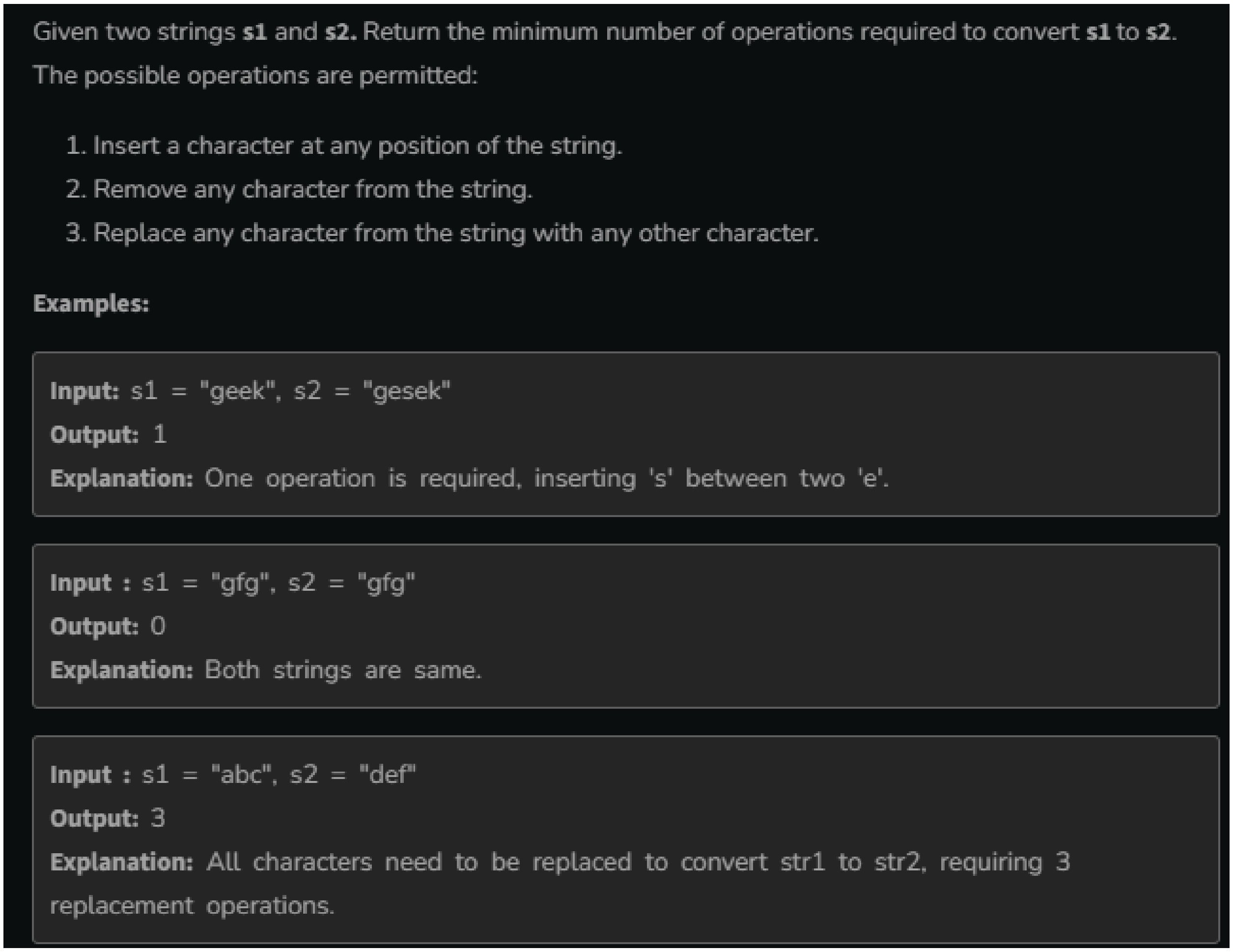
2 / 4

Accuracy : 50%

Time Taken

1.2

5.


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.

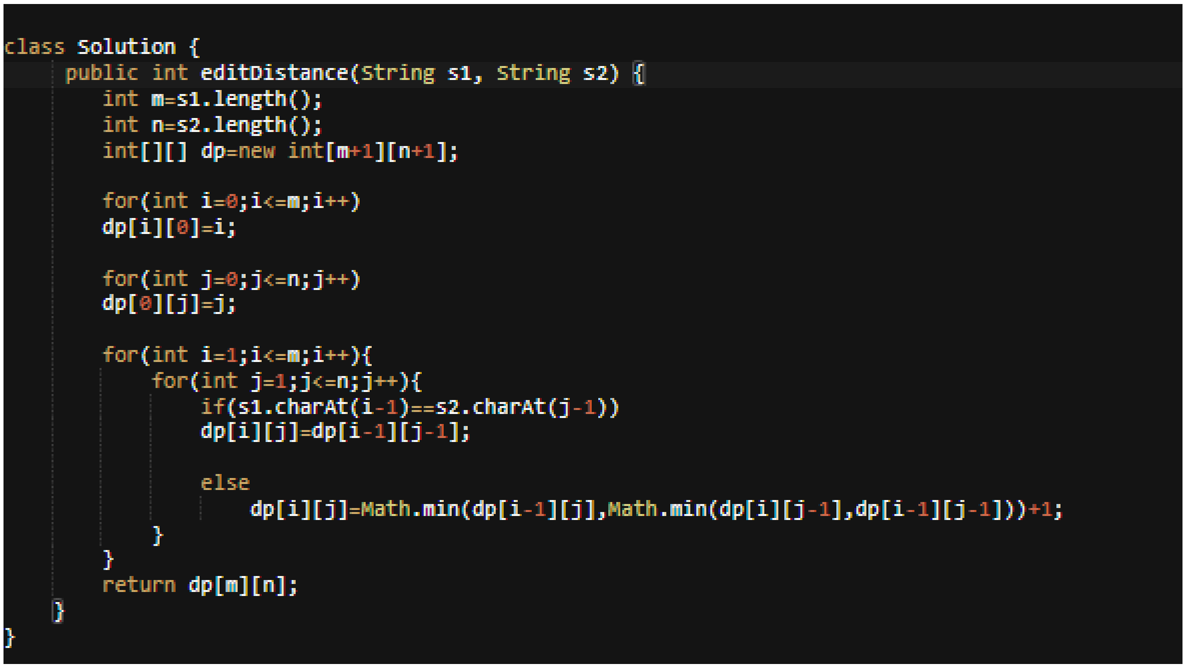
Examples:

```
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.
```

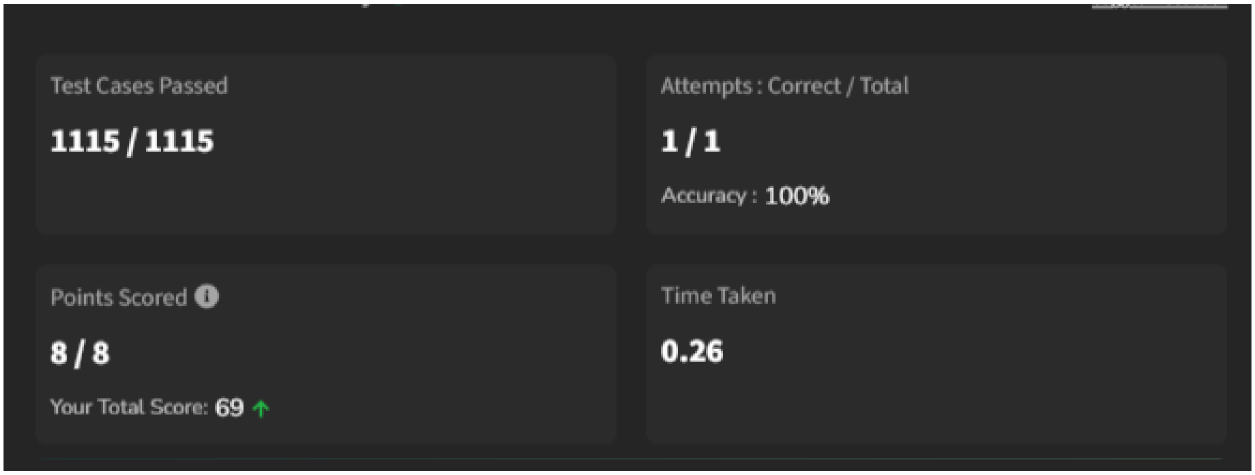
```
Input : s1 = "abc", s2 = "def"
Output: 3
Explanation: All characters need to be replaced to convert str1 to str2, requiring 3
replacement operations.
```

Code:



```
class Solution {
    public int editDistance(String s1, String s2) {
        int m=s1.length();
        int 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(dp[i-1][j],Math.min(dp[i][j-1],dp[i-1][j-1]))+1;
            }
        }
        return dp[m][n];
    }
}
```

Output:



Test Cases Passed

1115 / 1115

Attempts : Correct / Total

1 / 1

Accuracy : 100%

Points Scored

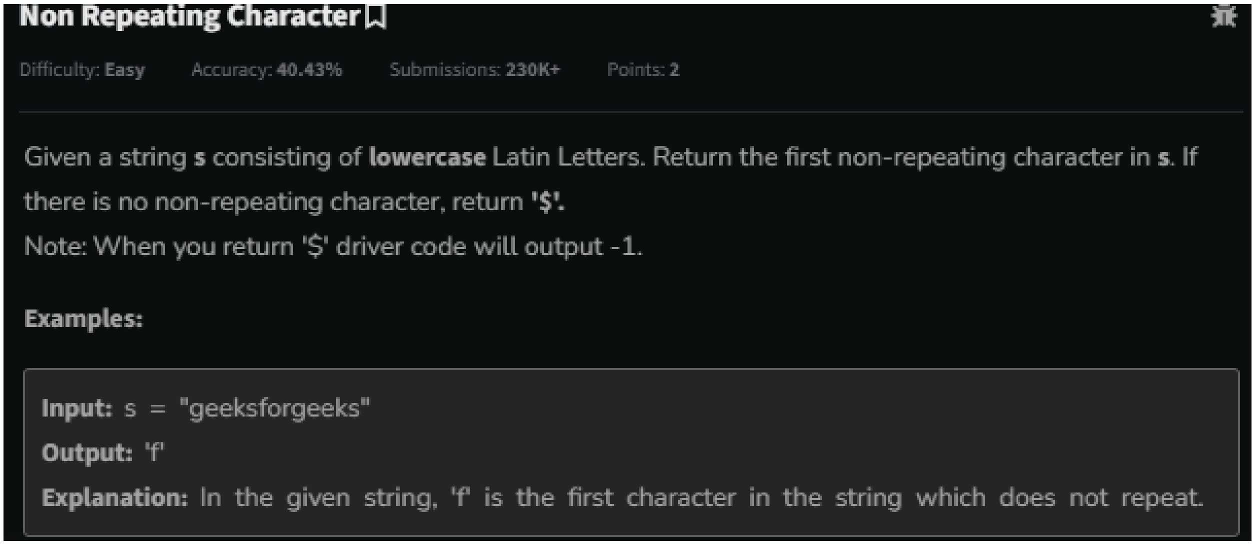
8 / 8

Your Total Score: 69 ↑

Time Taken

0.26

6.



Non Repeating Character

Difficulty: **Easy** Accuracy: **40.43%** Submissions: **230K+** Points: **2**

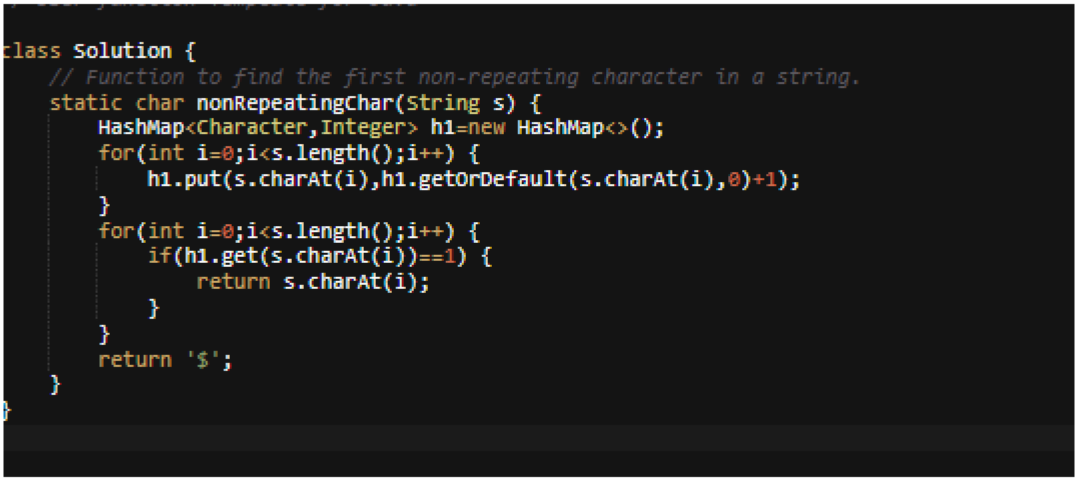
Given a string **s** consisting of **lowercase** Latin Letters. Return the first non-repeating character in **s**. If there is no non-repeating character, return **'$'**.

Note: When you return '$' driver code will output -1.

Examples:

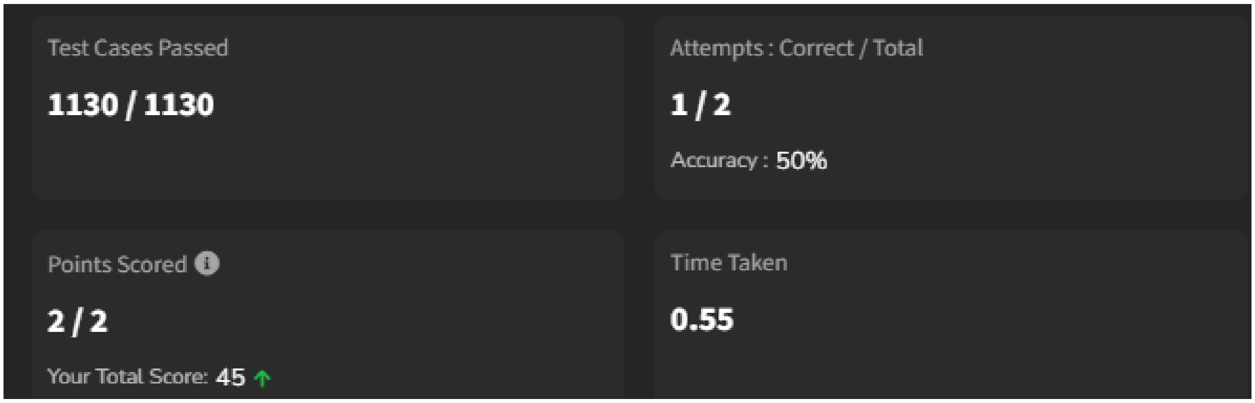
Input: s = "geeksforgeeks"
Output: 'f'
Explanation: In the given string, 'f' is the first character in the string which does not repeat.

Code:



```
class Solution {
    // Function to find the first non-repeating character in a string.
    static char nonRepeatingChar(String s) {
        HashMap<Character,Integer> h1=new HashMap<>();
        for(int i=0;i<s.length();i++) {
            h1.put(s.charAt(i),h1.getOrDefault(s.charAt(i),0)+1);
        }
        for(int i=0;i<s.length();i++) {
            if(h1.get(s.charAt(i))==1) {
                return s.charAt(i);
            }
        }
        return '$';
    }
}
```

Output:



Test Cases Passed

1130 / 1130

Attempts : Correct / Total

1 / 2

Accuracy : 50%

Points Scored

2 / 2

Your Total Score: 45 ↑

Time Taken

0.55