**1.Anagram Strings**

Given two strings S1 and S2 . Return "1" if both strings are anagrams otherwise return "0" .

**Note**:An anagram of a string is another string with exactly the same quantity of each character in it, in any order.  
  
**Example 1:**

**Input**: S1 = "cdbkdub" , S2 = "dsbkcsdn"

**Output:** 0

**Explanation**: Length of S1 is not same

as length of S2.

**Example 2:**

**Input:** S1 = "geeks" , S2 = "skgee"

**Output:**1

**Explanation**: S1 has the same quantity

of each character in it as S2.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **areAnagram()** which takes S1 and S2 as input and returns "1" if both strings are anagrams otherwise returns "0".  
  
**Expected Time Complexity:** O(n)  
**Expected Auxiliary Space:** O(K) ,Where K= Contstant  
  
**Constraints:**  
1 <= |S1| <= 1000  
1 <= |S2| <= 1000

Program:  
class Solution {

// Function is to check whether two strings are anagram of each other or not.

public static boolean areAnagrams(String s, String t) {

char[] a = s.toCharArray(), b = t.toCharArray();

Arrays.sort(a);

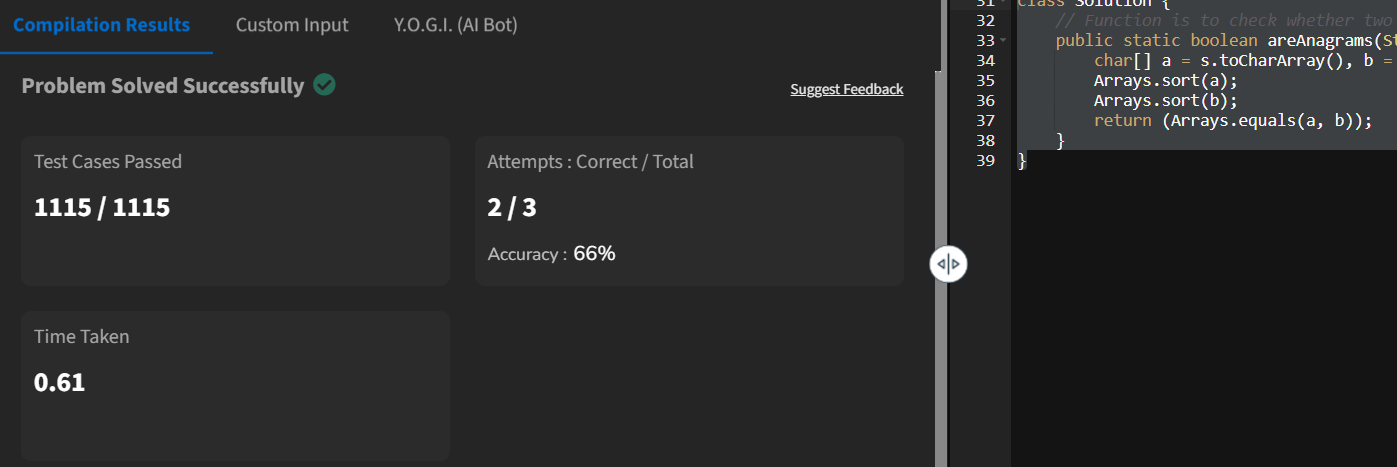
Arrays.sort(b);

return (Arrays.equals(a, b));

}

}

Output:



**2.maximum number of 1’s row**

Given a boolean 2D array, where each row is sorted. Find the row with the maximum number of 1s.

**Example 1:**

**Input:**

N = 3, M = 4

Mat[] = {{0 1 1 1},

{0 0 1 1},

  {0 0 1 1}}

**Output:** 0

**Explanation**: Row 0 has 3 ones whereas

rows 1 and 2 have just 2 ones.

**Example 2:**

**Input**:

N = 2, M = 2

Mat[] = {{0 1},

  {1 1}}

**Output:** 1

**Explanation**: Row 1 has 2 ones whereas

row 0 has just a single one.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **maxOnes ()**which takes a 2D array Mat[][] and its dimensions N and M as inputs and returns the row index with the maximum number of 1s (0-based index). If there are multiple rows with the maximum number of ones, then return the row with the smaller index.

**Expected Time Complexity:**O(NLogM).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= N, M <= 40

**Program:**

class Sol

{

public static int maxOnes (int a[][], int N, int M)

{

int r=0,m=0,z=0;

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

z=0;

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

if (a[i][j]==1){

z+=1;

}}

if (z>m){

m=z;

r=i;

}

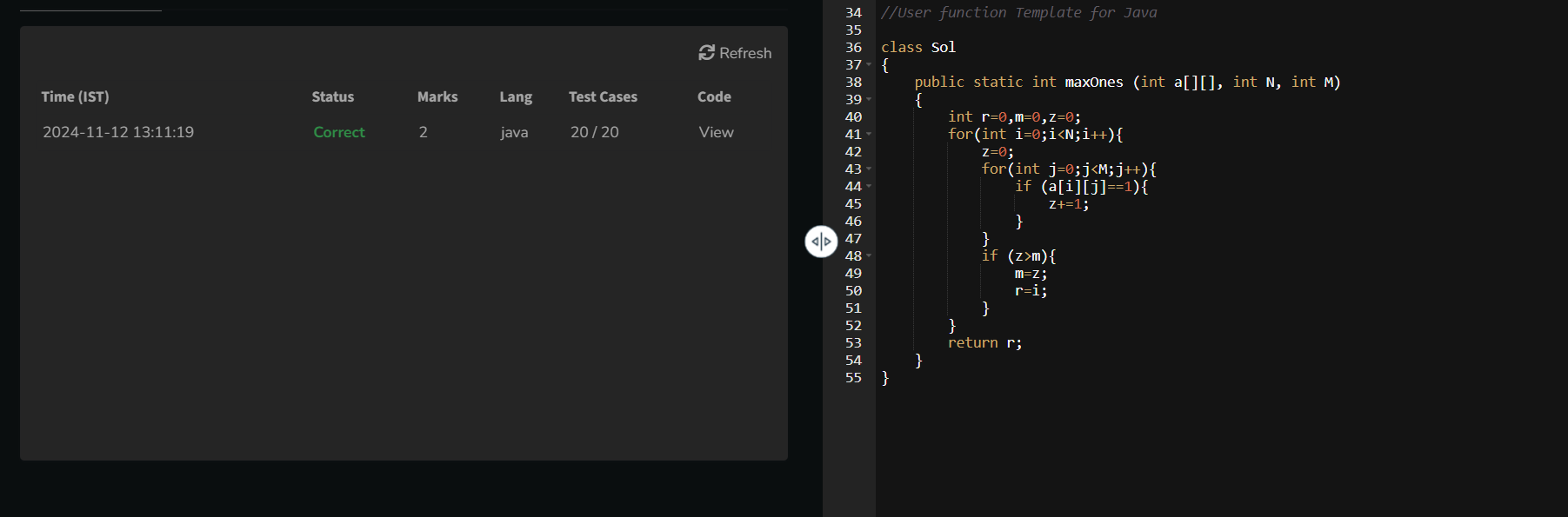
}

return r;

}

}

**OutPut:**

****

3. **Longest consecutive subsequence**

Given an array **arr** of non-negative integers. Find the **length** of the longest sub-sequence such that elements in the subsequence are consecutive integers, the**consecutive numbers** can be in **any order.**

**Examples:**

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

**Output:** 6

**Explanation:** The consecutive numbers here are 1, 2, 3, 4, 5, 6. These 6 numbers form the longest consecutive subsquence.

**Input:** arr[] = [1, 9, 3, 10, 4, 20, 2]

**Output:** 4

**Explanation:** 1, 2, 3, 4 is the longest consecutive subsequence.

**Input**: arr[] = [15, 13, 12, 14, 11, 10, 9]

**Output**: 7

**Explanation**: The longest consecutive subsequence is 9, 10, 11, 12, 13, 14, 15, which has a length of 7.

**Constraints:**  
1 <= arr.size() <= 105  
0 <= arr[i] <= 105

**Program:**

class Solution {

// Function to return length of longest subsequence of consecutive integers.

public int findLongestConseqSubseq(int[] a) {

int m=0,n=a.length,c=1;

Arrays.sort(a);

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

if (a[i]+1 == a[i+1]){

c+=1;

}

else if(a[i]==a[i+1]){continue;}

else{

//m=Math.max(m,c);

c=1;

}

m=Math.max(m,c);

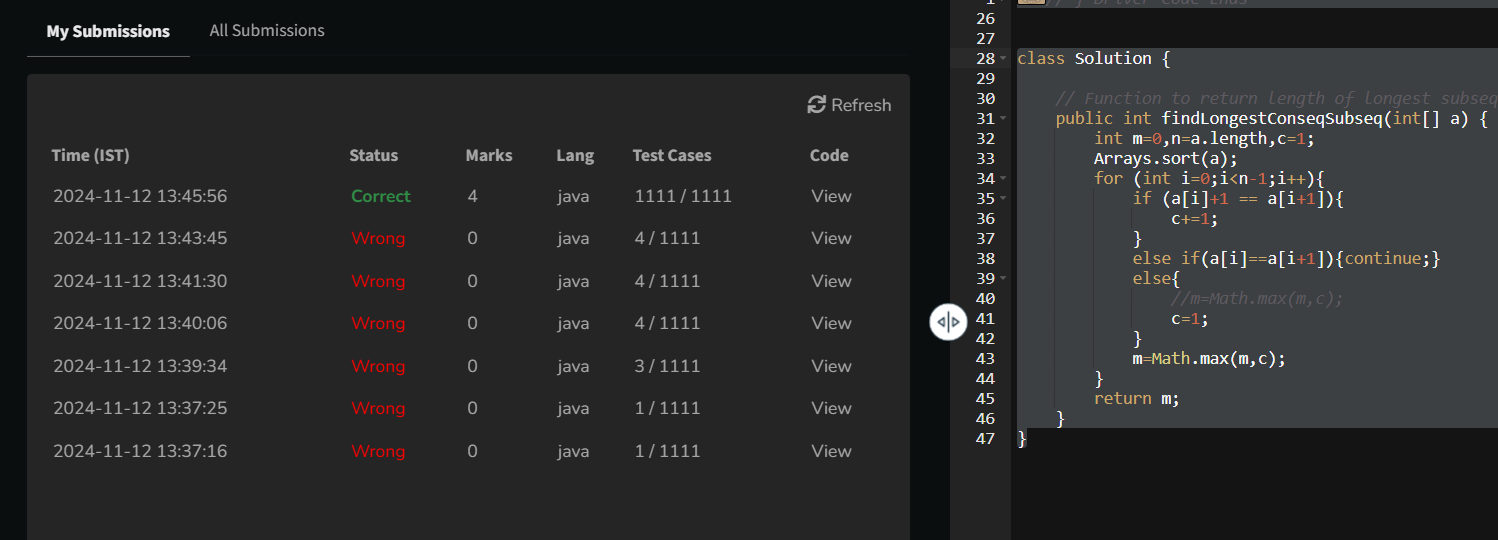
}

return m;

}

}

**Output:**



4. **Longest Palindrome in a String**

Given a string **s**, your task is to find the longest palindromic substring within s. A **substring** is a contiguous sequence of characters within a string, defined as s[i...j] where 0 ≤ i ≤ j < len(s).

A **palindrome** is a string that reads the same forward and backward. More formally, s is a palindrome if reverse(s) == s.

**Note:** If there are multiple palindromes with the same length, return the **first occurrence** of the longest palindromic substring from left to right.

**Examples :**

**Input:** s = "aaaabbaa"

**Output:** "aabbaa"

**Explanation**: The longest palindromic substring is "aabbaa".

**Input**: s = "abc"

**Output:** "a"

**Explanation**: "a", "b", and "c" are all palindromes of the same length, but "a" appears first.

**Input**: s = "abacdfgdcaba"   
**Output:** "aba"   
**Explanation**: The longest palindromic substring is "aba", which occurs twice. The first occurrence is returned.

**Constraints:**  
1 ≤ s.size() ≤ 103The string s consists of **only lowercase English letters** ('a' to 'z').

Program:  
class Solution {

// Static method to find the longest palindromic substring

static String longestPalindrome(String s) {

// code here

if (s.length() <= 1) {

return s;

}

int maxLen = 1;

int start = 0;

int end = 0;

boolean[][] dp = new boolean[s.length()][s.length()];

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

dp[i][i] = true;

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

if (s.charAt(j) == s.charAt(i) && (i - j <= 2 || dp[j + 1][i - 1])) {

dp[j][i] = true;

if (i - j + 1 > maxLen) {

maxLen = i - j + 1;

start = j;

end = i;

}

}

}

}

return s.substring(start, end + 1);

}

}

Output:

