**Anagram**

Difficulty: **Easy**Accuracy: **44.93%**Submissions: **339K+**Points: **2**

Given two strings **s1**and **s2**consisting of lowercase characters. The task is to check whether two given strings are an anagram of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, act and tac are an anagram of each other. Strings **s1**and **s2**can only contain lowercase alphabets.

Note: You can assume both the strings s1 & s2 are **non-empty**.

**Examples :**

**Input:** s1 = "geeks", s2 = "kseeg"

**Output:** true

**Explanation:** Both the string have same characters with same frequency. So, they are anagrams.

**Input:** s1 = "allergy", s2 = "allergic"

**Output:** false

**Explanation:** Characters in both the strings are not same, so they are not anagrams.

**Input:** s1 = "g", s2 = "g"

**Output:** true

**Explanation:** Character in both the strings are same, so they are anagrams.

**Constraints:**  
1 ≤ s1.size(), s2.size() ≤ 105

class Solution {

public static boolean areAnagrams(String s1, String s2) {

if(s1.length()!=s2.length()) return false;

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

for(int i=0;i<s1.length();i++){

char c=s1.charAt(i);

char v=s2.charAt(i);

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

if(map.containsKey(v)) map.put(v,map.get(v)-1);

else map.put(v,-1);

}

int sum=0;

for(int val:map.values()){

if(val<0) return false;

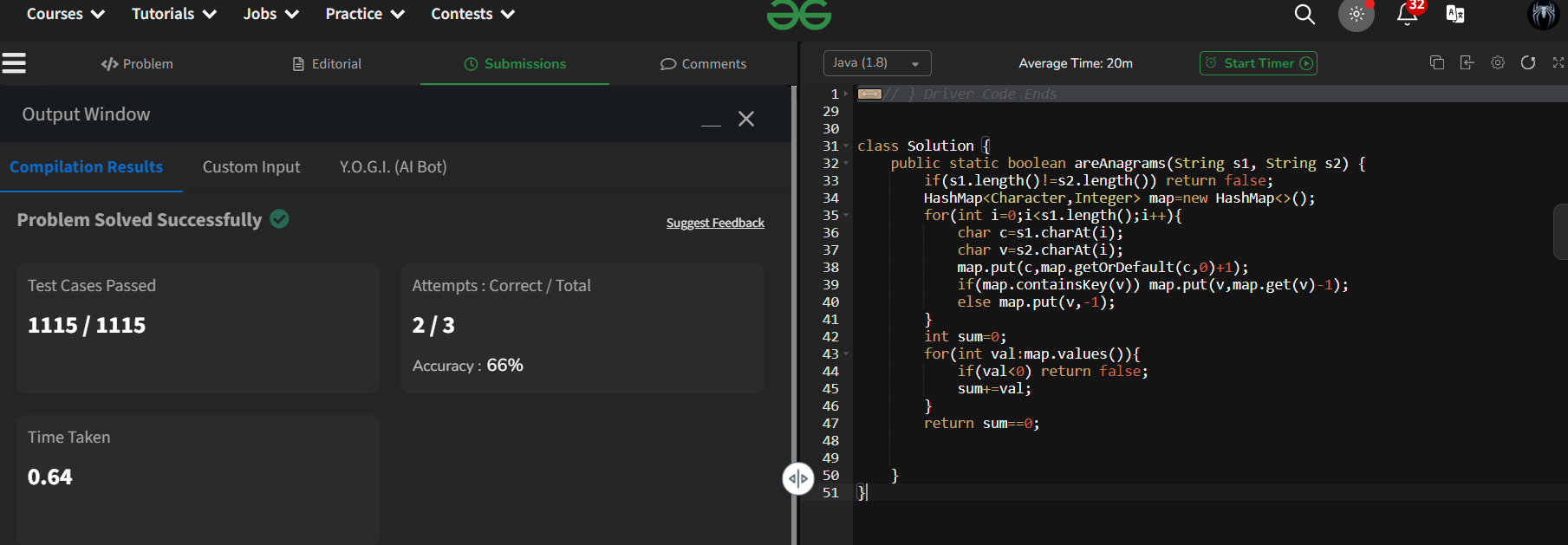
sum+=val;

}

return sum==0;

}

}



**Maximum no of 1's row**

Difficulty: **Easy**Accuracy: **53.13%**Submissions: **43K+**Points: **2**

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.

class Sol

{

public static int maxOnes (int mat[][], int n, int m)

{

int max=0,index=0;

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

int f=0;

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

if(mat[i][j]==1) f++;

}

if(f>max) {

max=f;index=i;

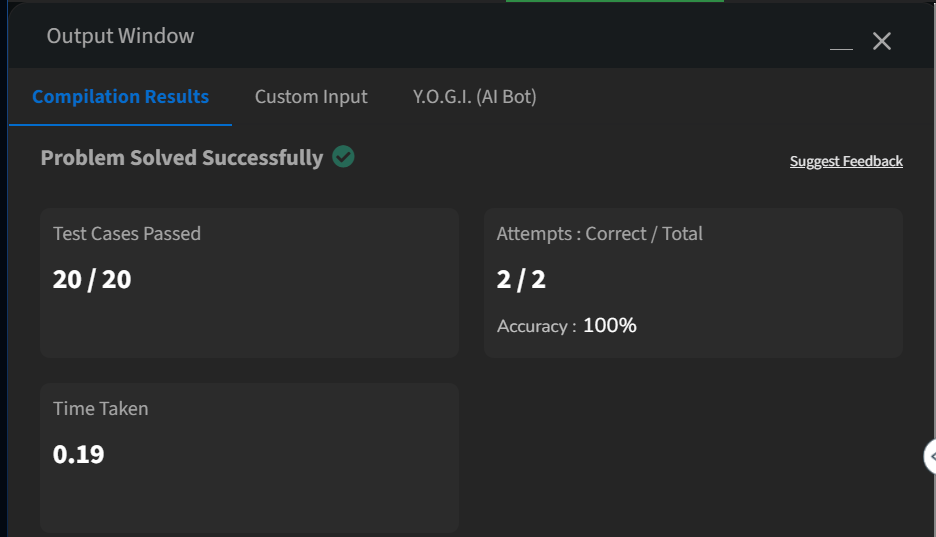
}

}

return index;

}

}



**Longest Consecutive Subsequence**

Difficulty: **Medium**Accuracy: **33.0%**Submissions: **309K+**Points: **4**

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.

class Solution {

public int findLongestConseqSubseq(int[] arr) {

Arrays.sort(arr);

int max=0,curr=1;

for(int i=0;i<arr.length-1;i++){

if (arr[i] == arr[i + 1]) continue;

//System.out.println(max+" "+curr);

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

curr++;

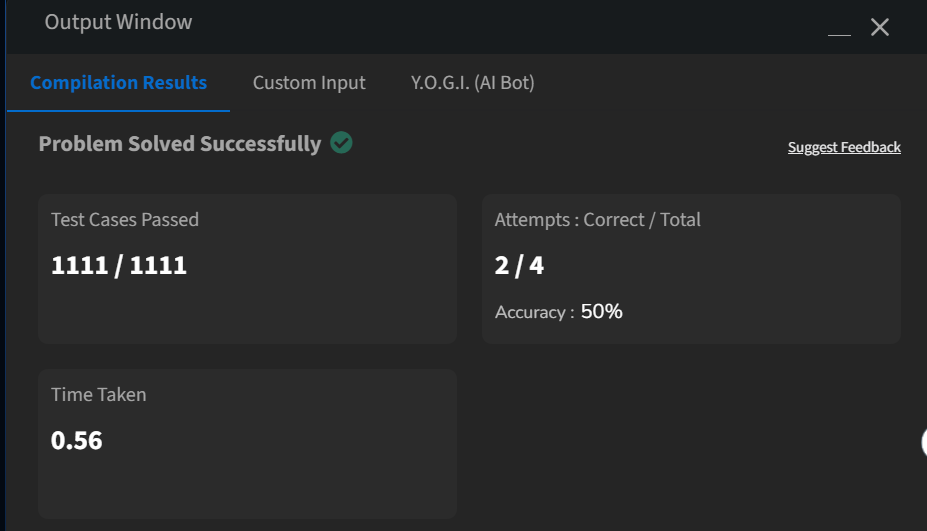
}else{

max=Math.max(curr,max);curr=1;

}

}return Math.max(curr,max);

}

}

**Longest Palindrome Substring**

Difficulty: **Medium**Accuracy: **23.2%**Submissions: **306K+**Points: **4**

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.

CODE:

**package** util;

**import** java.util.Scanner;

**public** **class** Palindrome {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter the string:");

String s = sc.nextLine();

Palindrome obj = **new** Palindrome();

String longestPalindromicSubstring = obj.longestPalindrome(s);

System.***out***.println("Longest Palindromic string: " + longestPalindromicSubstring);

}

**public** String longestPalindrome(String s) {

**int** start = 0;

**int** end =0;

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

**int** len1 = palindrome(s,i,i);

**int** len2 = palindrome(s,i,i+1);

**int** len = Math.*max*(len1,len2);

**if**(len>end-start){

start = i-(len-1)/2;

end = i+len/2;

}

}

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

}

**private** **int** palindrome(String s,**int** left,**int** right){

**while**(left>=0 && right<s.length() && s.charAt(left)==s.charAt(right)){

left--;

right++;

}

**return** right-left-1;

}

}

**Longest Palindromic Subsequence**

Difficulty: **Medium**Accuracy: **56.57%**Submissions: **91K+**Points: **4**

Given a String, find the longest palindromic subsequence.

**NOTE:**Subsequence of a given sequence is a sequence that can be derived from the given sequence by deleting some or no elements without changing the order of the remaining elements

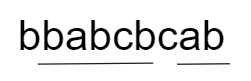
**Example 1:**

**Input:**

S = "bbabcbcab"

**Output:** 7

**Explanation**: Subsequence "babcbab" is the

longest subsequence which is also a palindrome.  


**Example 2:**

**Input**:

S = "abcd"

**Output:** 1

**Explanation**: "a", "b", "c" and "d" are

palindromic and all have a length 1.

class Solution {

public int longestPalinSubseq(String S) {

int n = S.length();

String revS = new StringBuilder(S).reverse().toString();

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

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

for (int j = 1; j <= n; j++) {

if (S.charAt(i - 1) == revS.charAt(j - 1)) {

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

} else {

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

}

}

}

return dp[n][n];

}

}

