**DSA PRACTICE – 3 – 12/11/24**

1. **ANAGRAM**

class Solution {

static int areAnagram(String S1, String S2) {

// code here

if(S1.length()!=S2.length()){

return 0;

}

int[] arr=new int[256];

for(char ch:S1.toCharArray()){

arr[ch]++;

}

for(char ch:S2.toCharArray()){

arr[ch]--;

}

for(int i=0;i<256;i++){

if(arr[i]!=0){

return 0;

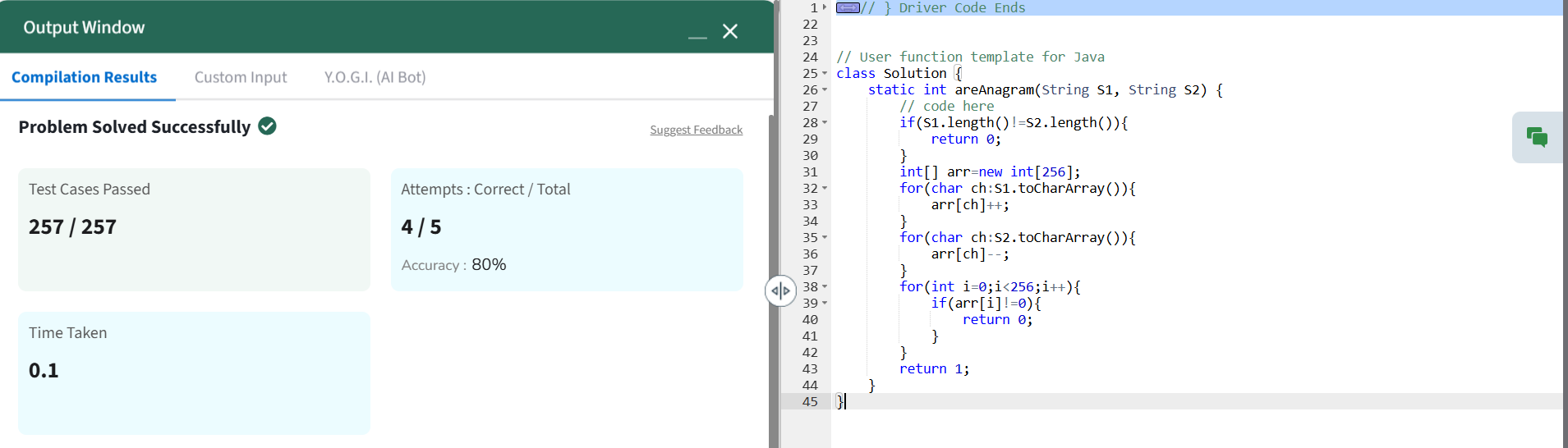
}

}

return 1;

}

}



Time Complexity : O(n)

Space Complexity : O(1)

**2.ROW WITH MAX 1’S**

class Solution {

public int rowWithMax1s(int arr[][]) {

// code here

int n=arr.length;

int m=arr[0].length;

int maxr = -1;

int j=m-1;

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

while(j>=0 && arr[i][j]==1){

j--;

maxr=i;

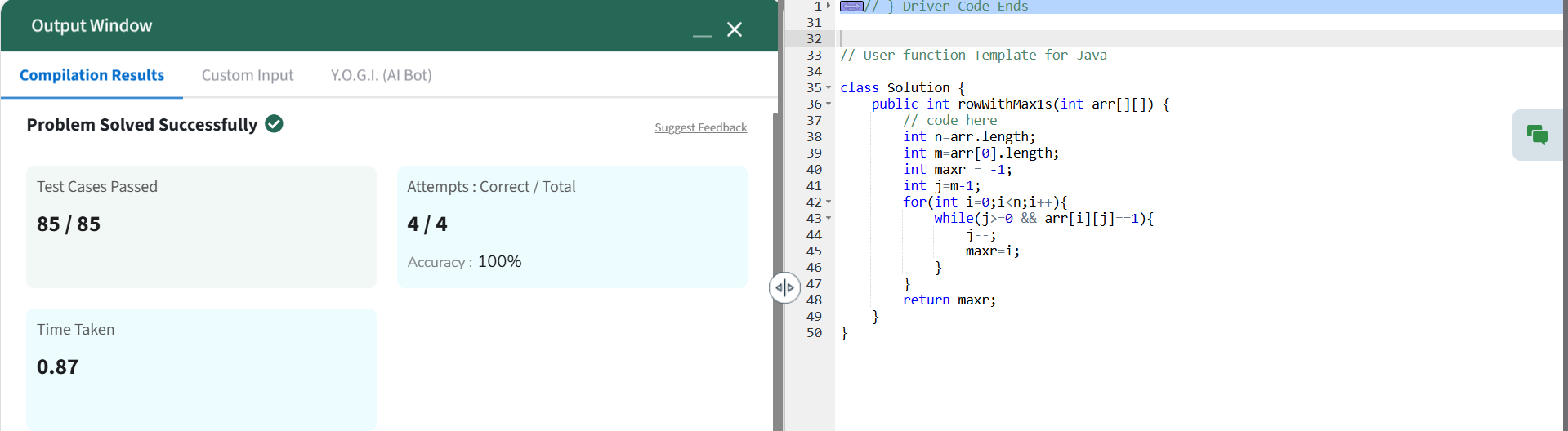
}

}

return maxr;

}

}



Time Complexity : O(n)

Space Complexity : O(1)

**3.LONGEST CONSECUTIVE SUBSEQUENCE**

class Solution {

public int findLongestConseqSubseq(int[] arr) {

int n = arr.length;

if (n == 0) {

return 0;

}

Arrays.sort(arr);

int maxLength = 1;

int currentLength = 1;

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

if (arr[i] != arr[i - 1]) {

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

currentLength++;

} else {

currentLength = 1;

}

maxLength = Math.max(maxLength, currentLength);

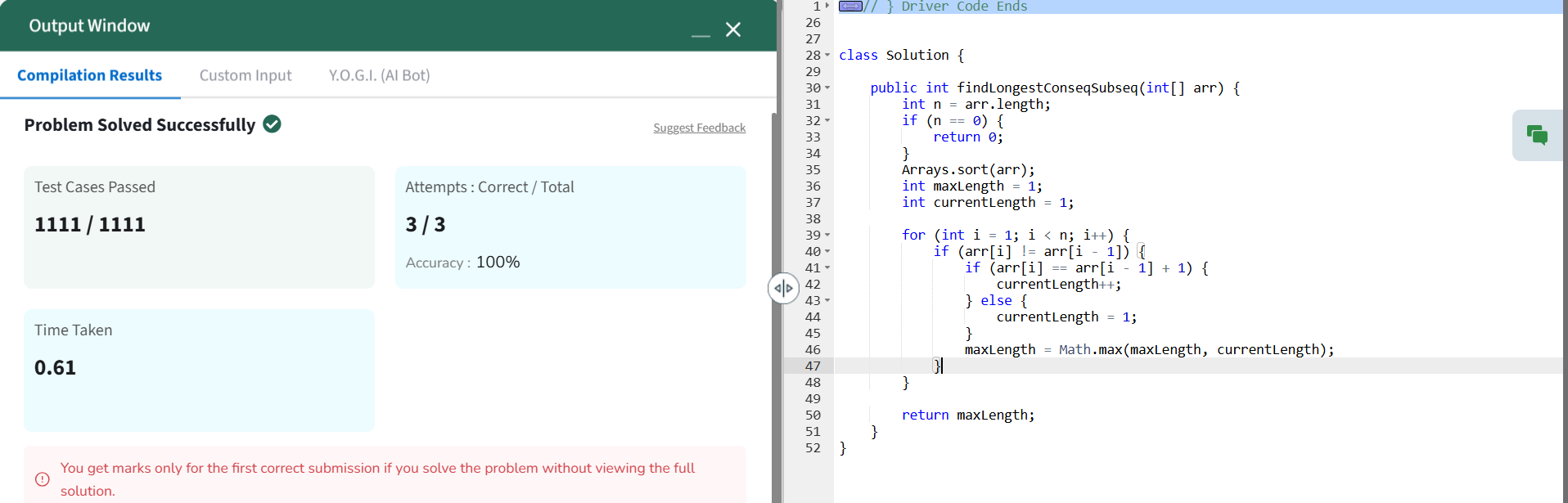
}

}

return maxLength;

}

}



Time Complexity : O(nlogn)

Space Complexity : O(1)

**4.LONGEST PALINDROMIC SUBSTRING**

class Solution {

public String longestPalindrome(String s) {

int n = s.length();

if (n <= 1) {

return s;

}

String longest = "";

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

String odd = expandAroundCenter(s, i, i);

if (odd.length() > longest.length()) {

longest = odd;

}

String even = expandAroundCenter(s, i, i + 1);

if (even.length() > longest.length()) {

longest = even;

}

}

return longest;

}

private String expandAroundCenter(String s, int left, int right) {

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

left--;

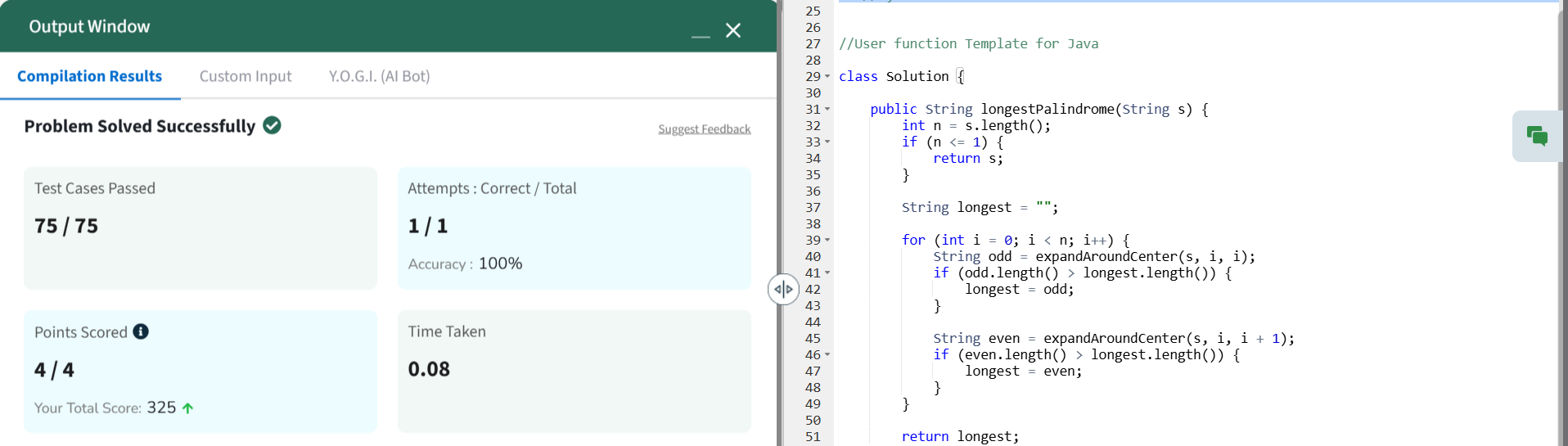
right++;

}

return s.substring(left + 1, right);

}

}



Time Complexity : O(n^2)

Space Complexity : O(1)

**5.RAT IN A MAZE**

class Solution {

private boolean isSafe(int x, int y, int n, int[][] mat, boolean[][] visited) {

return (x >= 0 && x < n && y >= 0 && y < n && mat[x][y] == 1 && !visited[x][y]);

}

private void findPaths(int[][] mat, int x, int y, int n, String path, ArrayList<String> result, boolean[][] visited) {

if (x == n - 1 && y == n - 1) {

result.add(path);

return;

}

visited[x][y] = true;

if (isSafe(x + 1, y, n, mat, visited)) {

findPaths(mat, x + 1, y, n, path + 'D', result, visited);

}

if (isSafe(x, y - 1, n, mat, visited)) {

findPaths(mat, x, y - 1, n, path + 'L', result, visited);

}

if (isSafe(x, y + 1, n, mat, visited)) {

findPaths(mat, x, y + 1, n, path + 'R', result, visited);

}

if (isSafe(x - 1, y, n, mat, visited)) {

findPaths(mat, x - 1, y, n, path + 'U', result, visited);

}

visited[x][y] = false;

}

public ArrayList<String> findPath(int[][] mat) {

int n = mat.length;

ArrayList<String> result = new ArrayList<>();

if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) {

result.add("-1");

return result;

}

boolean[][] visited = new boolean[n][n];

findPaths(mat, 0, 0, n, "", result, visited);

if (result.isEmpty()) {

result.add("-1");

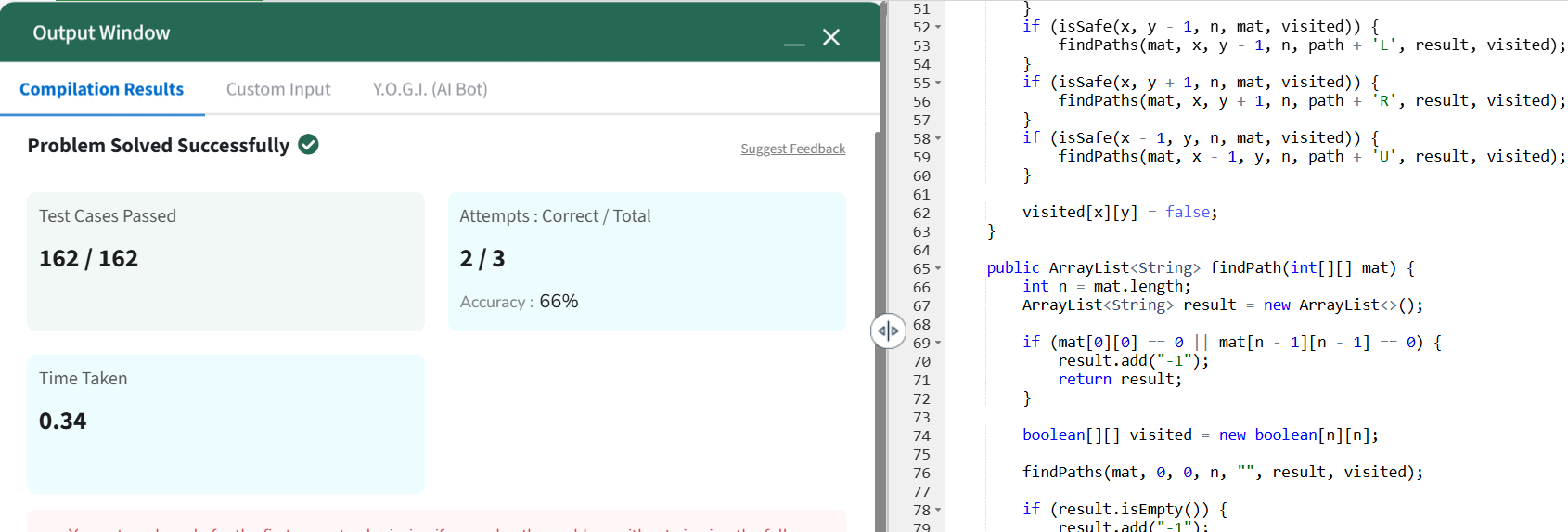
}

result.sort(String::compareTo);

return result;

}

}



Time Complexity : O(n)

Space Complexity : O(1)