

## DSA Practice – 3

### 1. Anagram program

The screenshot displays a coding platform interface for the 'Anagram' problem. The top navigation bar includes links for Courses, Tutorials, Jobs, Practice, and Contests. The left sidebar shows the 'Output Window' with 'Compilation Results' and 'Custom Input' tabs. The main area shows the problem status: 'Problem Solved Successfully' with a green checkmark. Below this, statistics are provided: 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 3 / 3', 'Accuracy: 100%', and 'Time Taken: 0.29'. The right side shows the Java code for the solution, which uses frequency arrays to compare two strings.

```
1 // Driver Code Ends
29 class Solution {
30     public static boolean areAnagrams(String s1, String s2) {
31         if (s1.length() != s2.length()) {
32             return false;
33         }
34         int[] freq = new int[26];
35         for (int i = 0; i < s1.length(); i++) {
36             freq[s1.charAt(i) - 'a']++;
37             freq[s2.charAt(i) - 'a']--;
38         }
39         for (int count : freq) {
40             if (count != 0) {
41                 return false;
42             }
43         }
44         return true;
45     }
46 }
47
```

**Time Complexity :  $O(n)$**

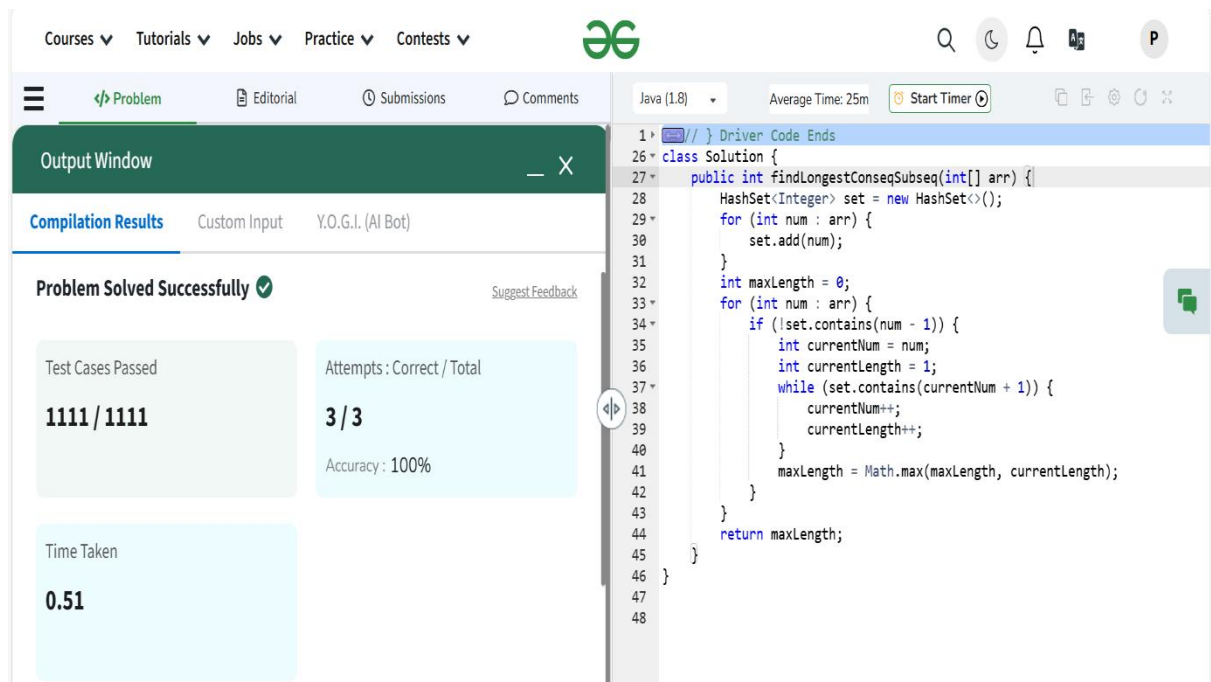
### 2. Row with max 1s'

The screenshot displays a coding platform interface for the 'Row with max 1s' problem. The top navigation bar includes links for Courses, Tutorials, Jobs, Practice, and Contests. The left sidebar shows the 'Output Window' with 'Compilation Results' and 'Custom Input' tabs. The main area shows the problem status: 'Problem Solved Successfully' with a green checkmark. Below this, statistics are provided: 'Test Cases Passed: 85 / 85', 'Attempts: Correct / Total: 2 / 2', 'Accuracy: 100%', and 'Time Taken: 0.89'. The right side shows the Java code for the solution, which iterates through a 2D array to find the row with the maximum number of 1s.

```
1 // Driver Code Ends
31 class Solution {
32     public int rowWithMax1s(int arr[][]) {
33         int n = arr.length;
34         int m = arr[0].length;
35         int maxRowIndex = -1;
36         int j = m - 1;
37         for (int i = 0; i < n; i++) {
38             while (j >= 0 && arr[i][j] == 1) {
39                 j--;
40                 maxRowIndex = i;
41             }
42         }
43         return maxRowIndex;
44     }
45 }
46
```

**Time Complexity :  $O(n + m)$**

### 3. Longest consecutive subsequence



The screenshot shows a coding platform interface. On the left, the 'Output Window' displays 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 3 / 3', and 'Accuracy: 100%'. The 'Time Taken' is 0.51. On the right, the code editor shows a Java solution for the 'Longest consecutive subsequence' problem. The code uses a HashSet to track elements and a loop to find the longest consecutive subsequence.

```
1 // } Driver Code Ends
26 class Solution {
27     public int findLongestConseqSubseq(int[] arr) {
28         HashSet<Integer> set = new HashSet<>();
29         for (int num : arr) {
30             set.add(num);
31         }
32         int maxLength = 0;
33         for (int num : arr) {
34             if (!set.contains(num - 1)) {
35                 int currentNum = num;
36                 int currentLength = 1;
37                 while (set.contains(currentNum + 1)) {
38                     currentNum++;
39                     currentLength++;
40                 }
41                 maxLength = Math.max(maxLength, currentLength);
42             }
43         }
44         return maxLength;
45     }
46 }
47
48
```

**Time Complexity :  $O(n)$**