

DSA CODING PRACTICE – 3

1. Anagram

Courses ▾ Tutorials ▾ Jobs ▾ Practice ▾ Contests ▾

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
2 / 2

Accuracy : 100%

Time Taken
0.5

Java (1.8) Average Time: 20m Start Timer

```
1 // Driver Code Ends
29
30
31 class Solution {
32     // function is to check whether two strings are anagram of each other or not.
33     public static boolean areAnagrams(String s1, String s2) {
34
35         // Your code here
36         if(s1.length() != s2.length()){
37             return false;
38         }
39         char[] s = s1.toCharArray();
40         char[] t = s2.toCharArray();
41
42         Arrays.sort(s);
43         Arrays.sort(t);
44
45         for(int i = 0; i < s.length(); i++){
46             if(s[i] != t[i]){
47                 return false;
48             }
49         }
50         return true;
51     }
52 }
```

Time Complexity: $O(n)$, Space Complexity: $O(1)$

2. Row with max 1s'

Courses ▾ Tutorials ▾ Jobs ▾ Practice ▾ Contests ▾

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓

Test Cases Passed
85 / 85

Attempts : Correct / Total
3 / 3

Accuracy : 100%

Time Taken
1.2

Java (1.8) Start Timer

```
1 // Driver Code Ends
31
32 // User function Template for Java
33
34 class Solution {
35     public int rowWithMaxis(int arr[][]){
36         // code here
37         int rows = arr.length;
38         int cols = arr[0].length;
39
40         int maxRowIndex = -1;
41         int maxOneCount = 0;
42
43         int row = 0;
44         int col = cols - 1;
45
46         while(row < rows && col >= 0){
47             if(arr[row][col] == 1){
48                 maxRowIndex = row;
49                 col--;
50             }
51             else{
52                 row++;
53             }
54         }
55         return maxRowIndex;
56     }
57 }
58
59
60 }
```

You get marks only for the first correct submission if you solve the problem without viewing the full solution.

Ctrl + Enter

Time Complexity: $O(m + n)$, Space Complexity: $O(1)$

3. Longest consecutive subsequence

The screenshot displays a coding platform interface for the problem 'Longest consecutive subsequence'. The 'Output Window' on the left shows 'Problem Solved Successfully' with 1111 test cases passed, 2/2 attempts correct, 100% accuracy, and a time taken of 0.61. The code editor on the right shows a Java solution (1.8) with an average time of 25m. The code defines a `Solution` class with a `findLongestConseqSubseq` method that sorts the array and iterates through it to find the longest consecutive subsequence.

```
1 // Driver Code Ends
26
27
28 class Solution {
29
30 // Function to return length of longest subsequence of consecutive integers.
31 public int findLongestConseqSubseq(int[] arr) {
32 // code here
33 int maxCount = 1;
34 int c = 1;
35 Arrays.sort(arr);
36 for(int i=1; i<arr.length;i++){
37 if(arr[i] - arr[i-1] == 1){
38 c++;
39 }else if(arr[i] - arr[i-1]==0){
40 } else{
41 c = 1;
42 }
43 maxCount = Math.max(maxCount, c);
44 }
45 return maxCount;
46 }
47 }
48 }
```

Time Complexity: $O(n)$, Space Complexity: $O(n)$

4. Longest palindrome in a string

The screenshot displays a coding platform interface for the problem 'Longest palindrome in a string'. The 'Output Window' on the left shows 'Problem Solved Successfully' with 1111 test cases passed, 1/2 attempts correct, 50% accuracy, 0/4 points scored, and a time taken of 0.14. The code editor on the right shows a Java solution (1.8) with an average time of 25m. The code defines a `Solution` class with a `longestPalindromicSubstring` method that uses a helper function `expandAroundCenter` to find the longest palindromic substring.

```
22 // User function Template for Java
23
24 class Solution {
25 // Static method to find the longest palindromic substring
26 static String longestPalindromicSubstring(String s) {
27 if (s.isEmpty()) {
28 return "";
29 }
30
31 int st = 0;
32 int e = 0;
33 for (int i = 0; i < s.length(); i++) {
34 int odd = expandAroundCenter(s, i, i);
35 int even = expandAroundCenter(s, i, i + 1);
36 int max_len = Math.max(odd, even);
37 if (max_len > e - st + 1) {
38 st = i - (max_len - 1) / 2;
39 e = i + max_len / 2;
40 }
41 }
42 return s.substring(st, e + 1);
43 }
44
45 private static int expandAroundCenter(String s, int l, int r) {
46 while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
47 l--;
48 r++;
49 }
50 return r - l - 1;
51 }
52 }
53 }
```

Time Complexity: $O(n^2)$, Space Complexity: $O(1)$

5.Rat in a Maze

The screenshot displays a coding platform interface with the following components:

- Top Navigation Bar:** Includes links for Courses, Tutorials, Jobs, Practice, and Contests. A search icon and a user profile icon are also present.
- Problem Header:** Shows the problem name 'Rat in a Maze' and the difficulty level 'Y.O.G.I. (AI Bot)'.
- Output Window:** Displays the status 'Problem Solved Successfully' with a green checkmark. It also shows 'Test Cases Passed: 162 / 162', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.56'.
- Code Editor:** Contains the Java code for the solution. The code defines a class 'Solution' with a method 'findPath' and a recursive 'find' method. The 'find' method explores four directions (Up, Down, Left, Right) and marks visited cells.
- Bottom Bar:** Includes a 'Submit' button and a 'Custom Input' field.

Time Complexity: $O(2^{(n^2)})$ Space Complexity: $O(n^2)$