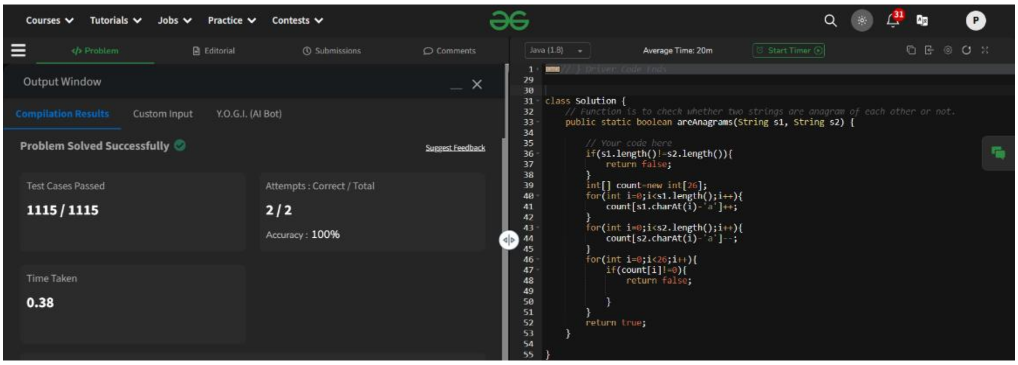
Date: 12/11/24

# DSA Practice Problems

1. **Anagram**

```
class Solution {
    public static boolean areAnagrams(String s1, String s2) {
        if(s1.length()!=s2.length()){
            return false;
        }
        int[] count=new int[26];
        for(int i=0;i<s1.length();i++){
            count[s1.charAt(i)-'a']++;
        }
        for(int i=0;i<s2.length();i++){
            count[s2.charAt(i)-'a']--;
        }
        for(int i=0;i<26;i++){
            if(count[i]!=0){
                return false;

            }
        }
        return true;
    }
}
```



Problem Solved Successfully

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
2 / 2
Accuracy : 100%

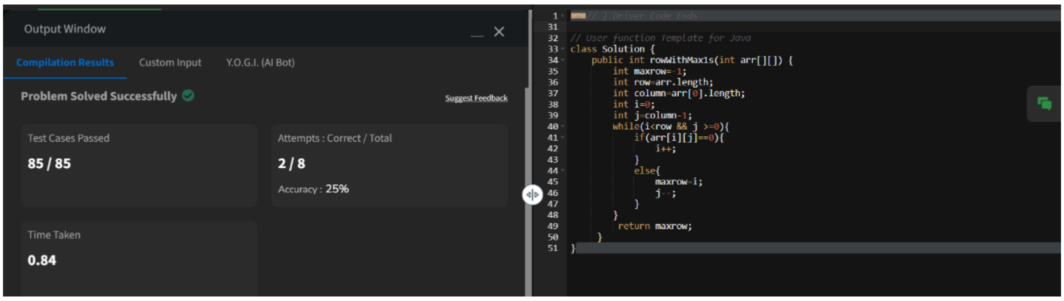
Time Taken
0.38

```
// } Driver Code Ends

class Solution {
    // Function is to check whether two strings are anagram of each other or not.
    public static boolean areAnagrams(String s1, String s2) {

        // Your code here
        if(s1.length()!=s2.length()){
            return false;
        }
        int[] count=new int[26];
        for(int i=0;i<s1.length();i++){
            count[s1.charAt(i)-'a']++;
        }
        for(int i=0;i<s2.length();i++){
            count[s2.charAt(i)-'a']--;
        }
        for(int i=0;i<26;i++){
            if(count[i]!=0){
                return false;

            }
        }
        return true;
    }

}
```

Time Complexity: O(n)

## 2. Row with max 1s'

```
class Solution {
    public int rowWithMax1s(int arr[][]) {
        int maxrow=-1;
        int row=arr.length;
        int column=arr[0].length;
        int i=0;
        int j=column-1;
        while(i<row && j >=0){
            if(arr[i][j]==0){
                i++;
            }
            else{
                maxrow=i;
                j--;
            }
        }
         return maxrow;
    }
}
```



Problem Solved Successfully

Test Cases Passed
85 / 85

Attempts : Correct / Total
2 / 8
Accuracy : 25%

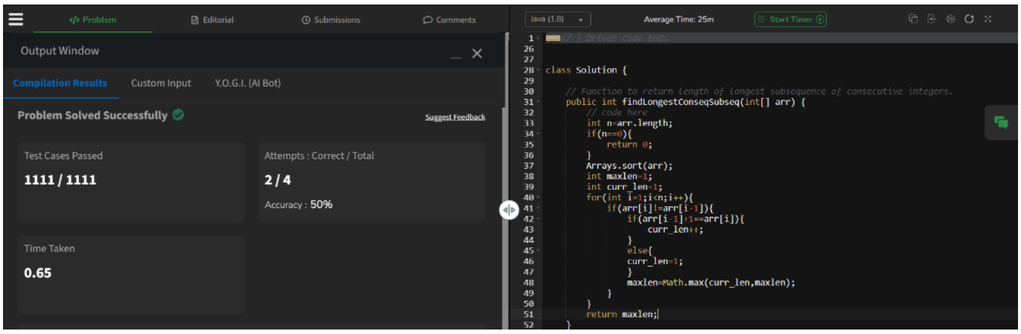
Time Taken
0.84

```
// } Driver Code Ends

// User function Template for Java
class Solution {
    public int rowWithMax1s(int arr[][]) {
        int maxrow=-1;
        int row=arr.length;
        int column=arr[0].length;
        int i=0;
        int j=column-1;
        while(i<row && j >=0){
            if(arr[i][j]==0){
                i++;
            }
            else{
                maxrow=i;
                j--;
            }
        }
         return maxrow;
    }
}
```

Time Complexity: O(M+N)

## 3. Longest consecutive subsequence

```
class Solution {
    public int findLongestConseqSubseq(int[] arr) {
        int n=arr.length;
        if(n==0){
            return 0;
        }
        Arrays.sort(arr);
        int maxlen=1;
        int curr_len=1;
        for(int i=1;i<n;i++){
            if(arr[i]!=arr[i-1]){
                if(arr[i-1]+1==arr[i]){
                    curr_len++;
                }
                else{
                curr_len=1;
                }
                maxlen=Math.max(curr_len,maxlen);
            }
        }
        return maxlen;
    }
}
```



Problem Solved Successfully

Test Cases Passed
1111 / 1111

Attempts : Correct / Total
2 / 4
Accuracy : 50%

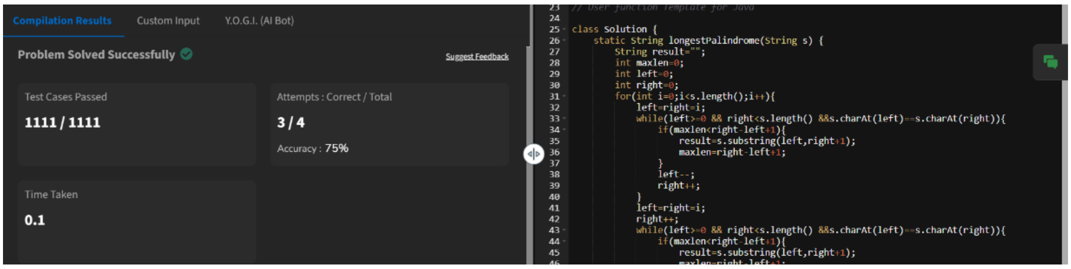
Time Taken
0.65

```
// } Driver Code Ends


class Solution {

    // Function to return length of longest subsequence of consecutive integers.
    public int findLongestConseqSubseq(int[] arr) {
        // code here
        int n=arr.length;
        if(n==0){
            return 0;
        }
        Arrays.sort(arr);
        int maxlen=1;
        int curr_len=1;
        for(int i=1;i<n;i++){
            if(arr[i]!=arr[i-1]){
                if(arr[i-1]+1==arr[i]){
                    curr_len++;
                }
                else{
                curr_len=1;
                }
                maxlen=Math.max(curr_len,maxlen);
            }
        }
        return maxlen;
    }
```

Time Complexity: O(n log n)

4. **Longest palindrome in a string**

```
class Solution {
  static String longestPalindrome(String s) {
    String result="";
    int maxlen=0;
    int left=0;
    int right=0;
    for(int i=0;i<s.length();i++){
      left=right=i;
      while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){
        if(maxlen<right-left+1){
          result=s.substring(left,right+1);
          maxlen=right-left+1;
        }
        left--;
        right++;
      }
      left=right=i;
      right++;
      while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){
        if(maxlen<right-left+1){
          result=s.substring(left,right+1);
          maxlen=right-left+1;
        }
        left--;
        right++;
      }
    }
    return result;
  }
}
```



Compilation Results

Custom Input

Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓

Suggest Feedback

Test Cases Passed

1111 / 1111

Attempts : Correct / Total

3 / 4

Accuracy : 75%

Time Taken

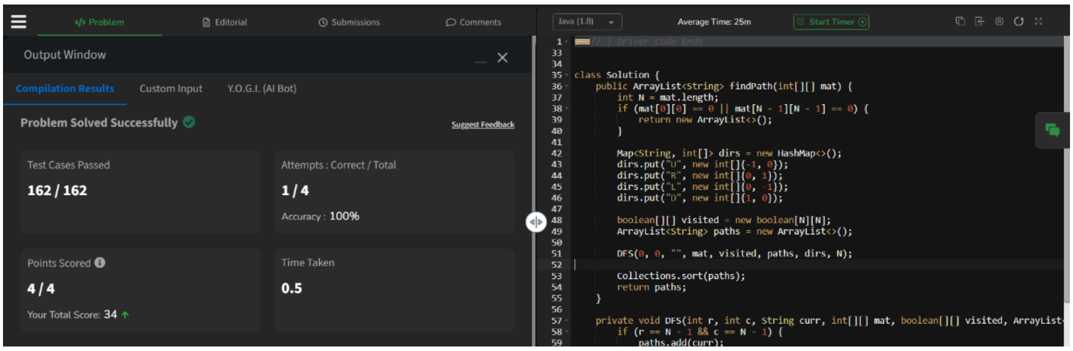
0.1

```
// User function Template for Java

class Solution {
    static String longestPalindrome(String s) {
        String result="";
        int maxlen=0;
        int left=0;
        int right=0;
        for(int i=0;i<s.length();i++){
            left=right=i;
            while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){
                if(maxlen<right-left+1){
                    result=s.substring(left,right+1);
                    maxlen=right-left+1;
                }
                left--;
                right++;
            }
            left=right=i;
            right++;
            while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){
                if(maxlen<right-left+1){
                    result=s.substring(left,right+1);
                    maxlen=right-left+1;
```

Time Complexity: O(n^2)

## 5. Rat in a maze problem

```
class Solution {
    public ArrayList<String> findPath(int[][] mat) {
        int N = mat.length;
        if (mat[0][0] == 0 || mat[N - 1][N - 1] == 0) {
            return new ArrayList<>();
        }

        Map<String, int[]> dirs = new HashMap<>();
        dirs.put("U", new int[]{-1, 0});
        dirs.put("R", new int[]{0, 1});
        dirs.put("L", new int[]{0, -1});
        dirs.put("D", new int[]{1, 0});

        boolean[][] visited = new boolean[N][N];
        ArrayList<String> paths = new ArrayList<>();

        DFS(0, 0, "", mat, visited, paths, dirs, N);

        Collections.sort(paths);
        return paths;
    }
    private void DFS(int r, int c, String curr, int[][] mat, boolean[][] visited, ArrayList<String> paths, Map<String, int[]> dirs, int N) {
        if (r == N - 1 && c == N - 1) {
            paths.add(curr);
            return;
        }

        visited[r][c] = true;

        for (Map.Entry<String, int[]> entry : dirs.entrySet()) {
            int nr = r + entry.getValue()[0];
            int nc = c + entry.getValue()[1];
            if (isValid(nr, nc, mat, visited, N)) {
                DFS(nr, nc, curr + entry.getKey(), mat, visited, paths, dirs, N);
            }
        }

        visited[r][c] = false;
    }

    private boolean isValid(int r, int c, int[][] mat, boolean[][] visited, int N) {
        return r >= 0 && r < N && c >= 0 && c < N && mat[r][c] == 1 && !visited[r][c];
    }
}
```



Problem Solved Successfully

Test Cases Passed: 162 / 162

Attempts : Correct / Total: 1 / 4

Accuracy : 100%

Points Scored: 4 / 4

Your Total Score: 34

Time Taken: 0.5

```
// } Driver Code Ends


class Solution {
    public ArrayList<String> findPath(int[][] mat) {
        int N = mat.length;
        if (mat[0][0] == 0 || mat[N - 1][N - 1] == 0) {
            return new ArrayList<>();
        }

        Map<String, int[]> dirs = new HashMap<>();
        dirs.put("U", new int[]{-1, 0});
        dirs.put("R", new int[]{0, 1});
        dirs.put("L", new int[]{0, -1});
        dirs.put("D", new int[]{1, 0});

        boolean[][] visited = new boolean[N][N];
        ArrayList<String> paths = new ArrayList<>();

        DFS(0, 0, "", mat, visited, paths, dirs, N);

        Collections.sort(paths);
        return paths;
    }

    private void DFS(int r, int c, String curr, int[][] mat, boolean[][] visited, ArrayList
        if (r == N - 1 && c == N - 1) {
            paths.add(curr);
```