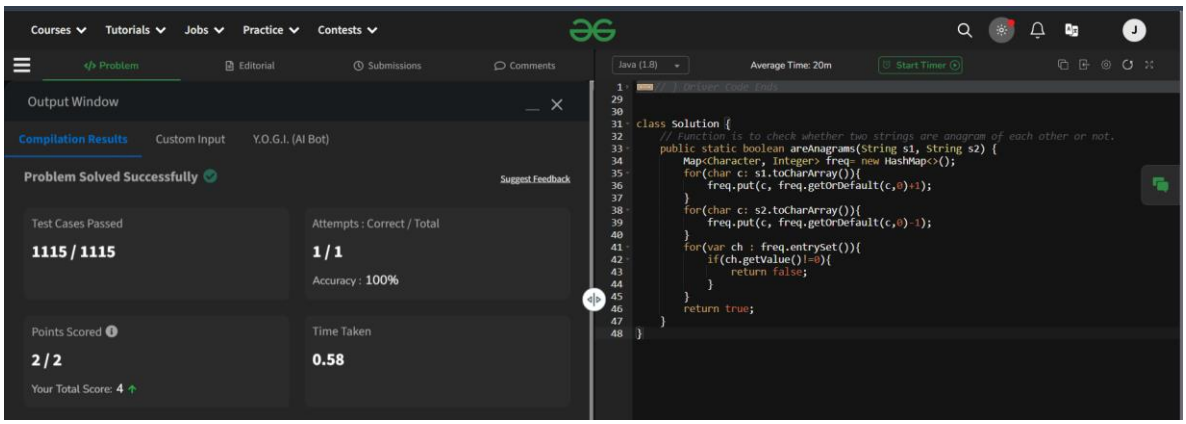
DSA PRACTICE QUESTIONS- DAY 3

DATE: 12/11/2024
NAME: Jeevitha R
REG NO: 22IT040

1. Anagrams

```
class Solution {
        // Function is to check whether two strings are anagram of each other or not.
        public static boolean areAnagrams(String s1, String s2) {
                Map<Character, Integer> freq= new HashMap<>();
                for(char c: s1.toCharArray() ){
                        freq.put(c, freq.getOrDefault(c,0)+1);
                        }
                for(char c: s2.tocharArray()){
                        freq.put(c, freq.getOrDefault(c,0)-1);
                        }
                for(var ch : freq.entrySet() ){
                        if(ch.getValue() != 0){
                                return false;
                        }
                }
                return true;
                }
        }
```

Output

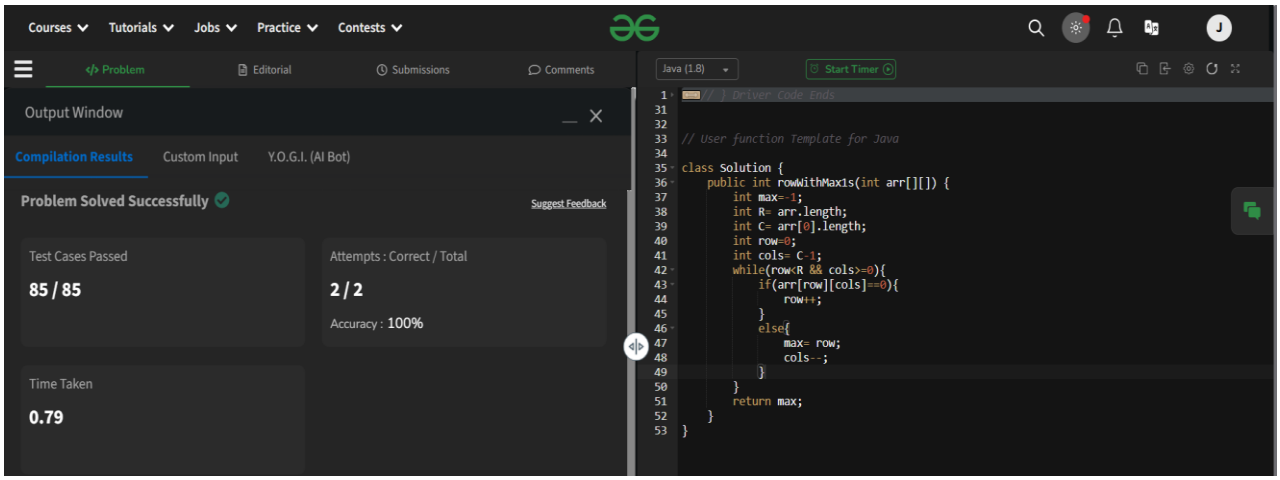


Time complexity: O(n)
Space complexity: O(1)

2.Row with max 1’s

```
class Solution {
        public int rowWithMax1s(int arr[][]) {
                int max =- 1;
                int R= arr.length;
                int C= arr[0].length;
                int row=0;
                int cols= C-1;
                while(row<R && cols>=0){
                        if(arr[row] [cols] == 0){
                                row++;
                        }else{
                                max= row;
                                cols --;
                        }
                }
                return max;
        }
}
```

Output

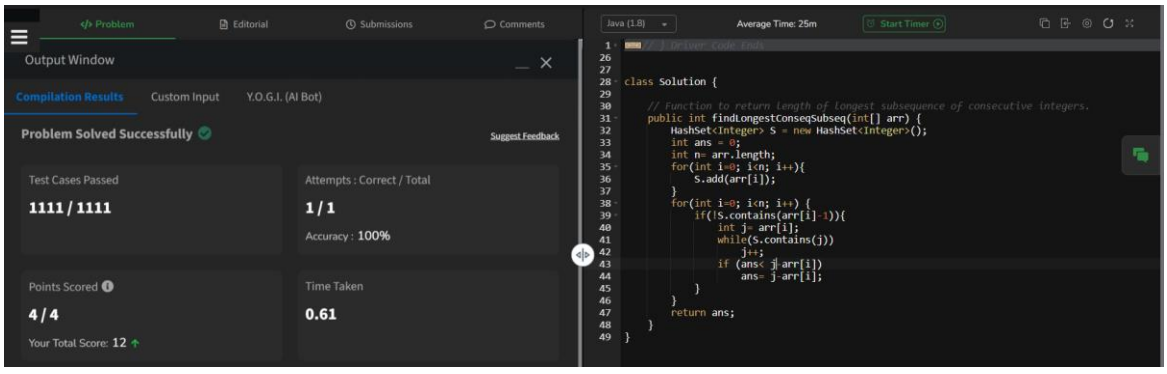


Time complexity: O(n+m)
Space complexity: O(1)

3. Longest consecutive subsequence

```
class Solution {
// Function to return length of longest subsequence of consecutive integers.
        public int findLongestConseqSubseq(int[] arr) {
                HashSet<Integer> S = new HashSet<Integer>();
                int ans = 0;
                int n= arr.length;
                for(int i=0; i<n; i++){
                        S.add(arr[i]);
                }
                for(int i=0; i<n; i++) {
                        if(!S.contains(arr[i]-1)){
                                int j= arr[i];
                                while(S.contains(j)){
                                        j++;
                                }
                        if (ans< j-arr[i]){
                                ans= j-arr[i];
                                }
                        }
                }
                return ans;
        }
}
```

Output:

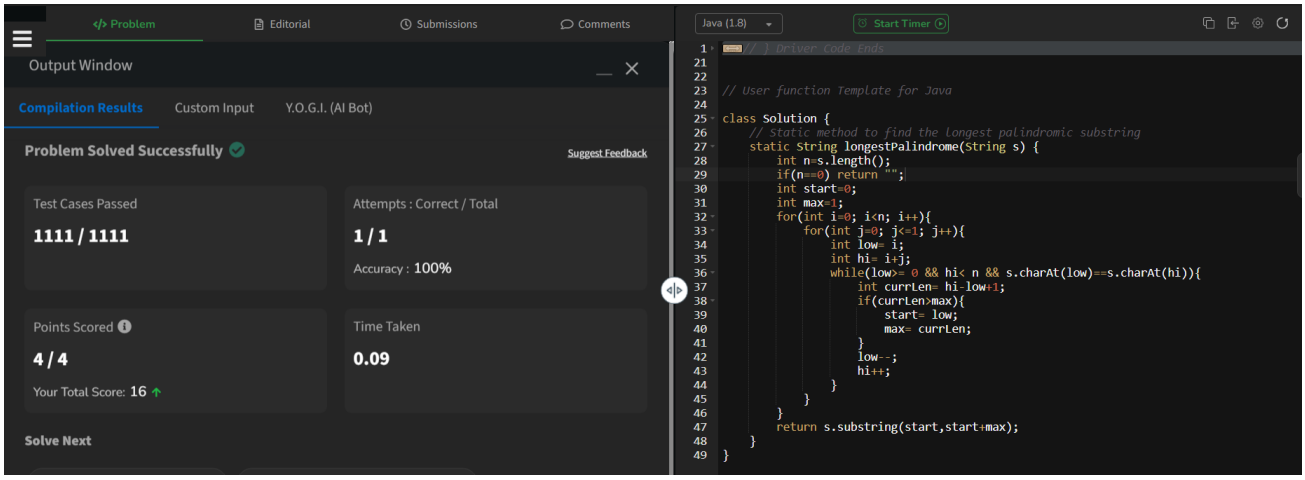


Time complexity: O(n^2)
Space complexity: O(n)

4.Longest palindrome in a string

```
class Solution{
    static String longestPalindrome(String s){
        int n=s.length();
        if(n==0)return "";
        int start=0;
        int max=1;
        for(int i=0;i<n;i++){
            for(int j=0;j<=1;j++){
                int low=i;
                int hi=i+j;
                while(low>=0 && hi<n && s.charAt(low) == s.charAt(hi)){
                    int currLen=hi-low+1;
                    if(currLen>max){
                        start=low;
                        max=currLen;
                    }
                    low--;
                    hi++;
                }
            }
        }
        return s.substring(start,start+max);
    }
}
```

Output



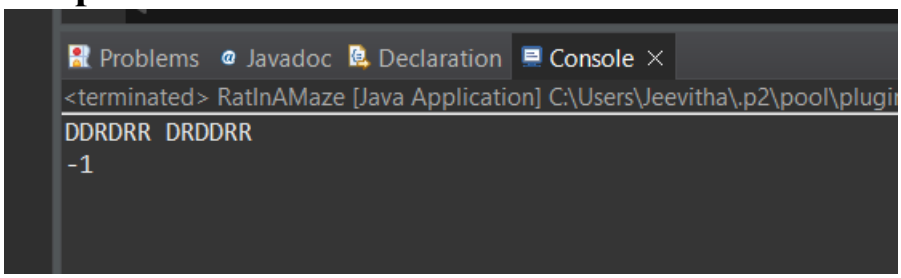
Time complexity: O(n^2)
Space complexity: O(1)

5. Rat in a maze problem

```
package practiceset3;
import java.util.*;
public class RatInAMaze {
   static boolean isSafe(int x,int y,int n,int[][] mat,boolean[][] visited){
      if(x>=0&&x<n&&y>=0&&y<n&&mat[x][y]==1&&!visited[x][y]){
         return true;
      }
      return false;
   }
   static void findPaths(int x,int y,int n,int[][] mat,boolean[][] visited,String path,List<String>
paths){
      if(x==n-1&&y==n-1){
         paths.add(path);
         return;
      }
      visited[x][y]=true;
      if(isSafe(x+1,y,n,mat,visited)){
         findPaths(x+1,y,n,mat,visited,path+"D",paths);
      }
      if(isSafe(x,y-1,n,mat,visited)){
         findPaths(x,y-1,n,mat,visited,path+"L",paths);
      }
      if(isSafe(x,y+1,n,mat,visited)){
         findPaths(x,y+1,n,mat,visited,path+"R",paths);
      }
      if(isSafe(x-1,y,n,mat,visited)){
         findPaths(x-1,y,n,mat,visited,path+"U",paths);
      }
      visited[x][y]=false;
   }
   static List<String> solveMaze(int n,int[][] mat){
      List<String> paths=new ArrayList<>();
      if(mat[0][0]==0){
         return paths;
      }
      boolean[][] visited=new boolean[n][n];
      findPaths(0,0,n,mat,visited,"",paths);
      Collections.sort(paths);
      return paths;
   }
         public static void main(String[] args) {
               int[][] mat1={
                     {1,0,0,0},
                     {1,1,0,1},
                     {1,1,0,0},
                     {0,1,1,1}
                  };
                  List<String> result=solveMaze(4,mat1);
                  if(result.isEmpty()){
                     System.out.println("-1");
                  }
                  else{
                     for(String path:result){
                        System.out.print(path+" ");
```

```
            }
        }
        System.out.println();
        int[][] mat2={
            {1,0},
            {1,0}
        };
        result=solveMaze(2,mat2);
        if(result.isEmpty()){
            System.out.println("-1");
        }
        else{
            for(String path:result){
                System.out.print(path+" ");
            }
        }
    }
}
```

Output



Time complexity: O(4^(n^2))
Space complexity: O(n^2)