**DSA – DAY 3 CODING 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']--;

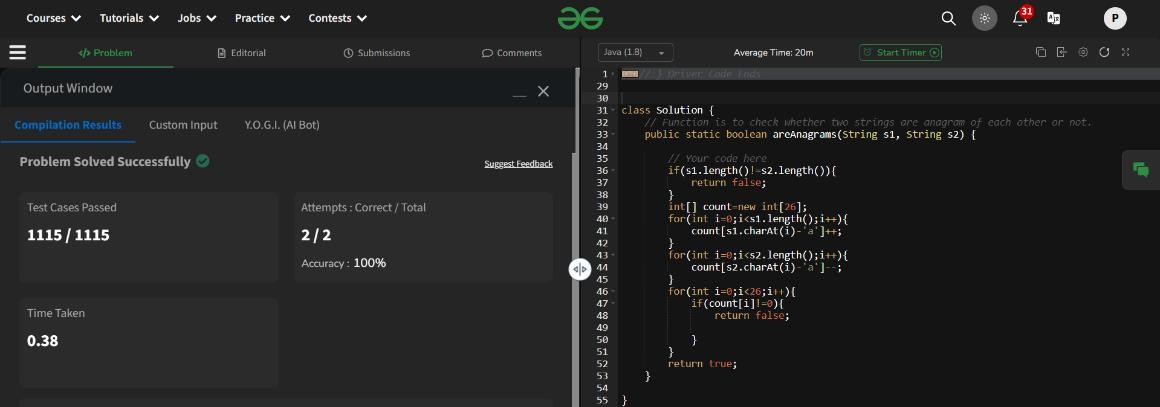
}

for(int i=0;i<26;i++){ if(count[i]!=0){ return false;

} } return true;

}

}



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--;

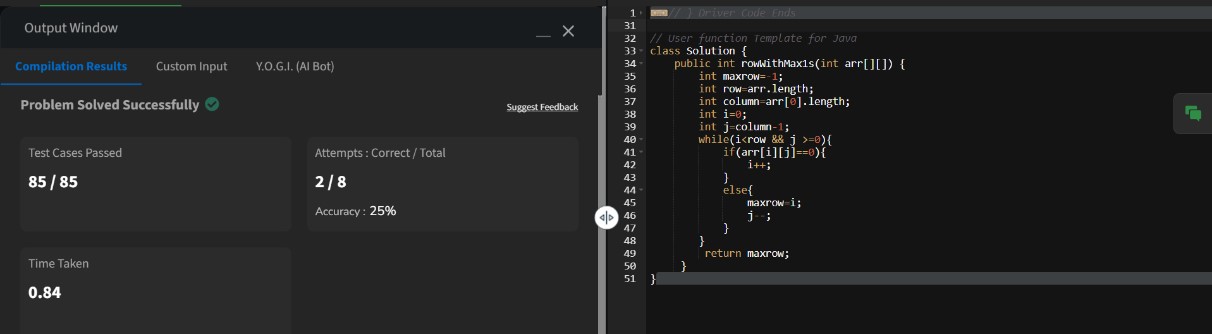
}

}

return maxrow;

}

}



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);

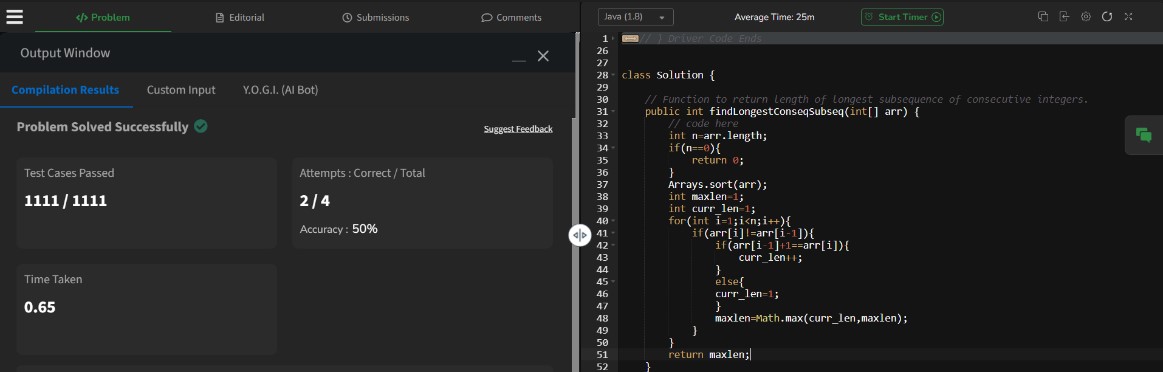
}

}

return maxlen;

}

}



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++;

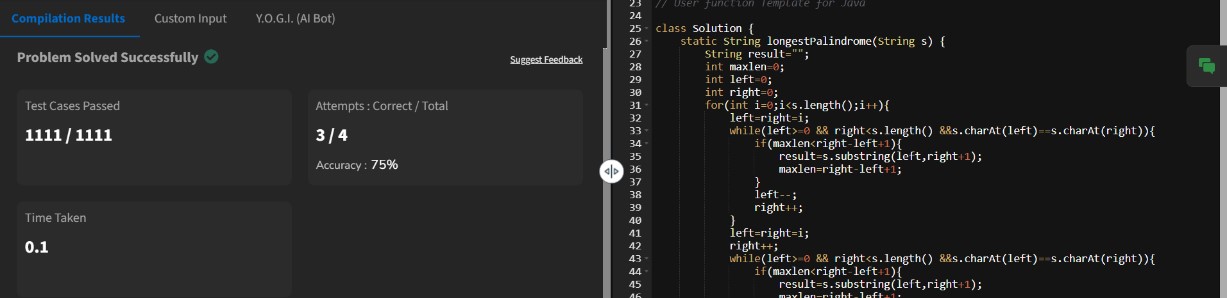
} left=right=i; right++; while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){ if(maxlen<right-left+1){ result=s.substring(left,right+1); maxlen=right-left+1;

} left--; right++;

} } return result;

}

}



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];

}

}

