1 . Anagram Problem

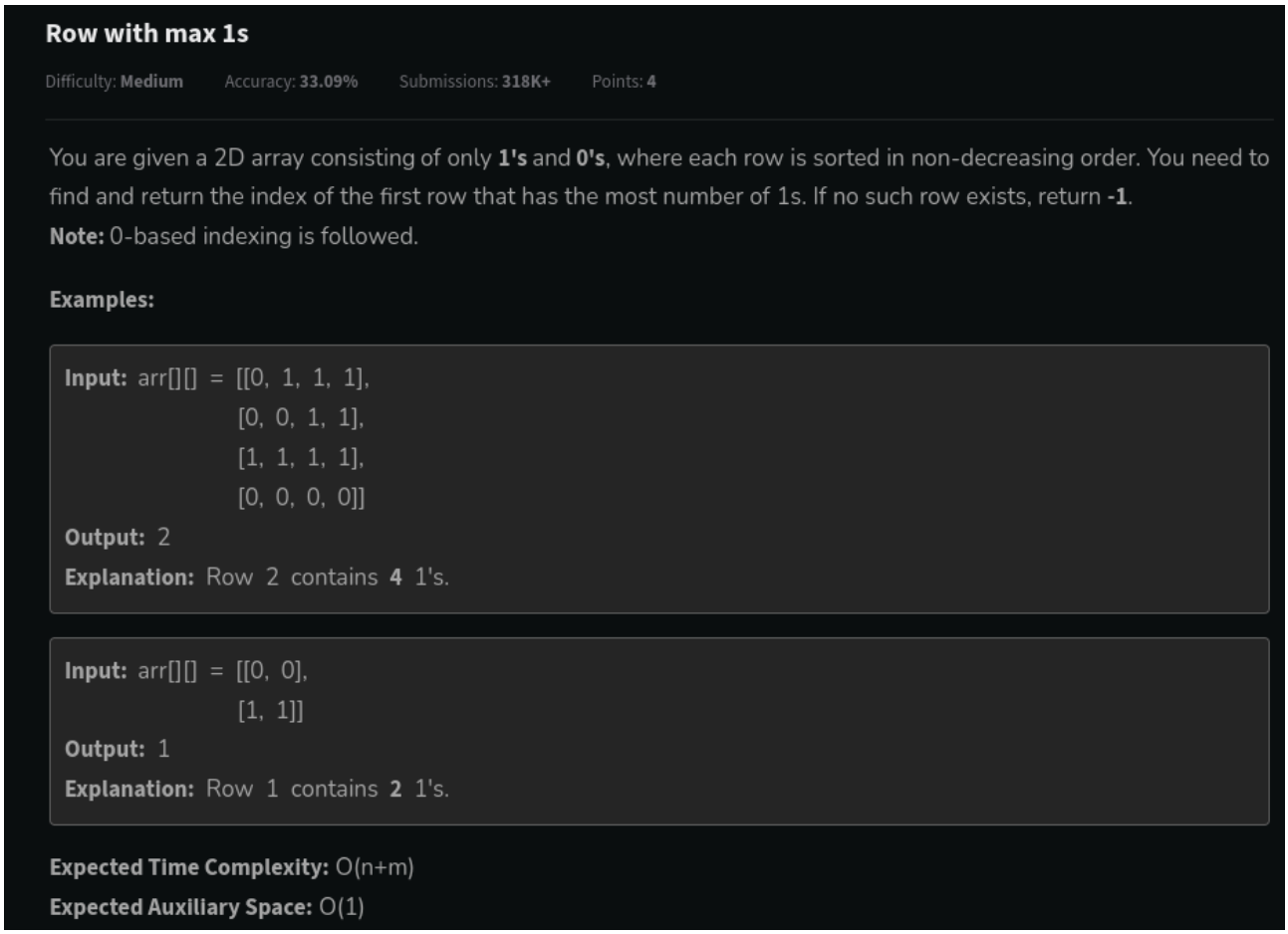
```cpp
#include<bits/stdc++.h>
using namespace std;

bool isAnagram(const string& s, const string& t) {
    if (s.size() != t.size()) return false;
    unordered_map<char, int> charCount;
    for (char c : s) charCount[c]++;
    for (char c : t) {
        if (--charCount[c] < 0) return false;
    }
    return true;
}
int main() {
    int testCases;
    cin >> testCases;
    for (int i = 1; i <= testCases; i++) {
        string s, t;
        cin >> s;
        cin >> t;
        bool result = isAnagram(s, t);
        cout  << (result ? "True" : "False") << endl;
    }

    return 0;
}
```

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

2. Row with Maximum 1’s



Row with max 1s

Difficulty: **Medium** Accuracy: **33.09%** Submissions: **318K+** Points: **4**

You are given a 2D array consisting of only **1's** and **0's**, where each row is sorted in non-decreasing order. You need to find and return the index of the first row that has the most number of 1s. If no such row exists, return **-1**.

Note: 0-based indexing is followed.

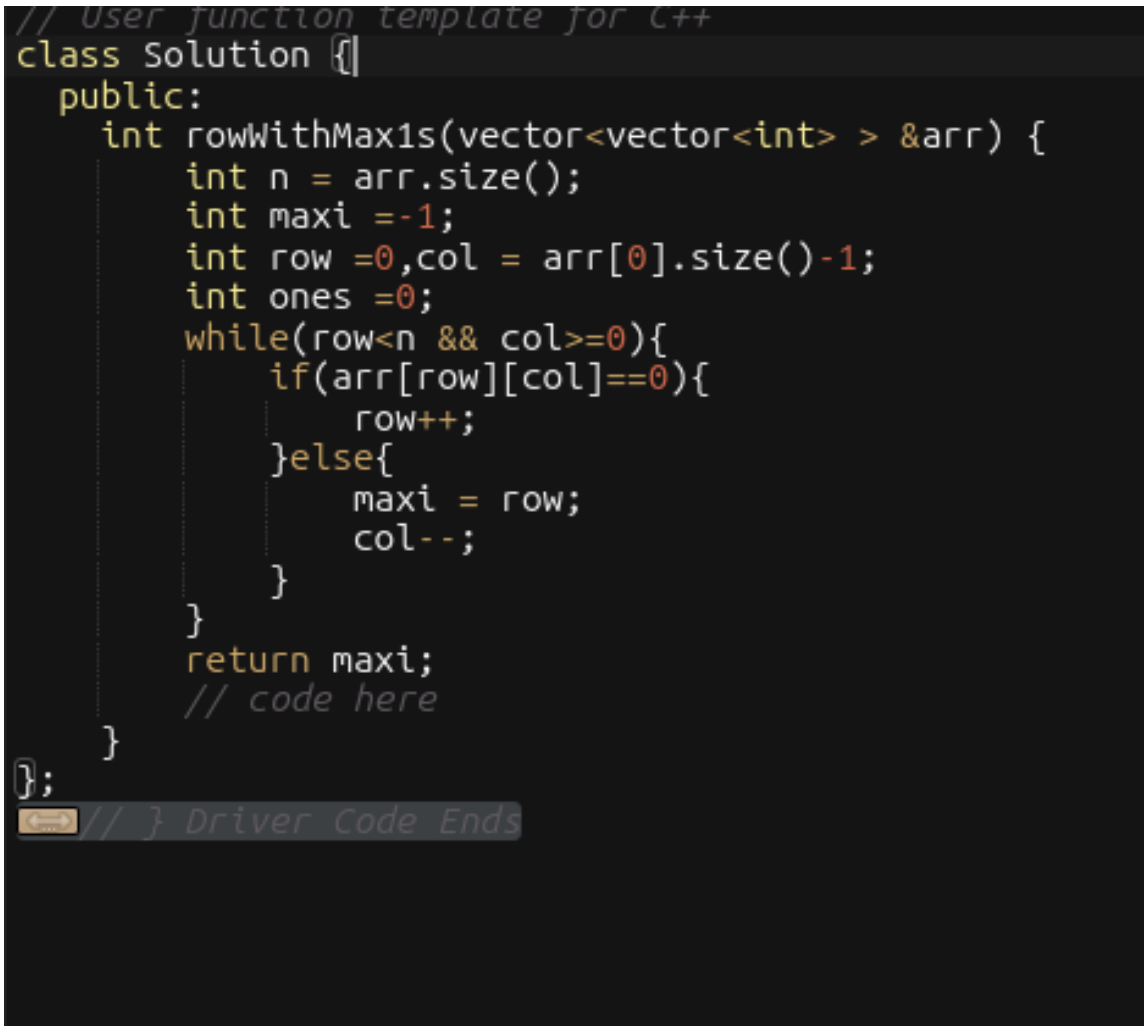
Examples:

Input: arr[][] = [[0, 1, 1, 1],
[0, 0, 1, 1],
[1, 1, 1, 1],
[0, 0, 0, 0]]
Output: 2
Explanation: Row 2 contains **4** 1's.

Input: arr[][] = [[0, 0],
[1, 1]]
Output: 1
Explanation: Row 1 contains **2** 1's.

Expected Time Complexity: O(n+m)
Expected Auxiliary Space: O(1)

Time



```
// User function template for C++
class Solution {
  public:
    int rowWithMax1s(vector<vector<int> > &arr) {
        int n = arr.size();
        int maxi =-1;
        int row =0,col = arr[0].size()-1;
        int ones =0;
        while(row<n && col>=0){
            if(arr[row][col]==0){
                row++;
            }else{
                maxi = row;
                col--;
            }
        }
        return maxi;
        // code here
    }
};
// } Driver Code Ends
```

Complexity:**O(4^(n*m))**
Space Complexity:**O(n*m)**

3. Longest consequtive subsequence

Longest consecutive subsequence

Difficulty: **Medium** Accuracy: **33.0%** Submissions: **309K+** Points: **4**

Given an array **arr** of non-negative integers. Find the **length** of the longest sub-sequence such that elements in the subsequence are consecutive integers, the **consecutive numbers** can be in **any order.**

Examples:

Input: arr[] = [2, 6, 1, 9, 4, 5, 3]
Output: 6
Explanation: The consecutive numbers here are 1, 2, 3, 4, 5, 6. These 6 numbers form the longest consecutive subsquence.

Input: arr[] = [1, 9, 3, 10, 4, 20, 2]
Output: 4
Explanation: 1, 2, 3, 4 is the longest consecutive subsequence.

Input: arr[] = [15, 13, 12, 14, 11, 10, 9]
Output: 7
Explanation: The longest consecutive subsequence is 9, 10, 11, 12, 13, 14, 15, which has a length of 7.

Constraints:
$1 <= arr.size() <= 10^5$
$0 <= arr[i] <= 10^5$

```cpp
#include <bits/stdc++.h>
using namespace std;

int longestConsecutive(vector<int>& arr) {
    int n = arr.size();
    int longseq = 0;
    unordered_set<int> st(arr.begin(), arr.end());
    for (int num : arr) {
        if (!st.count(num - 1)) {
            int curr = num;
            int currstr = 1;
            while (st.count(curr + 1)) {
                curr += 1;
                currstr += 1;
            }
            longseq = max(longseq, currstr);
        }
    }
    return longseq;
}
```

Time Complexity:O(n)

Space Complexity:O(n)

4. longest palindrome in a string

Longest Palindrome in a String

Difficulty: **Medium** Accuracy: **23.2%** Submissions: **306K+** Points: **4**

Given a string **s**, your task is to find the longest palindromic substring within s. A **substring** is a contiguous sequence of characters within a string, defined as s[i...j] where $0 \leq i \leq j < \text{len}(s)$.

A **palindrome** is a string that reads the same forward and backward. More formally, s is a palindrome if reverse(s) == s.

Note: If there are multiple palindromes with the same length, return the **first occurrence** of the longest palindromic substring from left to right.

Examples :

Input: s = "aaaabbaa"
Output: "aabbaa"
Explanation: The longest palindromic substring is "aabbaa".

Input: s = "abc"
Output: "a"
Explanation: "a", "b", and "c" are all palindromes of the same length, but "a" appears first.

Input: s = "abacdfgdcaba"
Output: "aba"
Explanation: The longest palindromic substring is "aba", which occurs twice. The first occurrence is returned.

Constraints:
$1 \leq s.size() \leq 10^3$
The string s consists of **only lowercase English letters** ('a' to 'z').

```cpp
#include <iostream>
#include <string>
using namespace std;

void longestPalindrome(string s) {
    string res = "";
    int resLen = 0;

    for (int i = 0; i < s.size(); i++) {
        int l = i, r = i;
        while (l >= 0 && r < s.size() && s[l] == s[r]) {
            if ((r - l + 1) > resLen) {
                res = s.substr(l, r - l + 1);
                resLen = r - l + 1;
            }
            l--;
            r++;
        }

        l = i, r = i + 1;
        while (l >= 0 && r < s.size() && s[l] == s[r]) {
            if ((r - l + 1) > resLen) {
                res = s.substr(l, r - l + 1);
                resLen = r - l + 1;
            }
            l--;
            r++;
        }
    }

    cout << "Longest Palindromic Substring: " << res << endl;
}
```

Time Complexity: O(n^2)

Space Complexity : O(1)

5. rat in a maze problem

Rat in a Maze Problem - I

Difficulty: **Medium** Accuracy: **35.75%** Submissions: **302K+** Points: **4**

Consider a rat placed at **(0, 0)** in a square matrix **mat** of order **n* n**. It has to reach the destination at **(n - 1, n - 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **'U'(up)**, **'D'(down)**, **'L' (left)**, **'R' (right)**. Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.
Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell. In case of no path, return an empty list. The driver will output **"-1"** automatically.

Examples:

Input: mat[][] = [[1, 0, 0, 0],
[1, 1, 0, 1],
[1, 1, 0, 0],
[0, 1, 1, 1]]
Output: DDRDRR DRDDRR
Explanation: The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

Input: mat[][] = [[1, 0],
[1, 0]]
Output: -1
Explanation: No path exists and destination cell is blocked.

Expected Time Complexity: $O(3^{n^2})$
Expected Auxiliary Space: O(l * x)
Here l = length of the path, x = number of paths.

```cpp
void backTrack(int row, int col, int n, int m, vector<vector<int>>& mat,
               vector<vector<bool>>& vis, vector<string>& ans, string str) {
    if (row == n - 1 && col == m - 1) {
        ans.push_back(str);
        return;
    }
    vis[row][col] = true;
    if (row + 1 < n && !vis[row + 1][col] && mat[row + 1][col] == 1) {
        backTrack(row + 1, col, n, m, mat, vis, ans, str + "D");
    }
    if (col - 1 >= 0 && !vis[row][col - 1] && mat[row][col - 1] == 1) {
        backTrack(row, col - 1, n, m, mat, vis, ans, str + "L");
    }
    if (col + 1 < m && !vis[row][col + 1] && mat[row][col + 1] == 1) {
        backTrack(row, col + 1, n, m, mat, vis, ans, str + "R");
    }
    if (row - 1 >= 0 && !vis[row - 1][col] && mat[row - 1][col] == 1) {
        backTrack(row - 1, col, n, m, mat, vis, ans, str + "U");
    }
    vis[row][col] = false;
}
void findPath(vector<vector<int>>& mat, int n) {
    vector<string> ans;
    if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) {
        cout << -1 << endl;
        return;
    }
    vector<vector<bool>> vis(n, vector<bool>(n, false));
    backTrack(0, 0, n, n, mat, vis, ans, "");
    if (ans.empty(
        cout << -1 << endl;
    } else {
        sort(ans.begin(), ans.end());
        for (const string& path : ans) {
            cout << path << " ";
        }
        cout << endl;
    }
}
```