1.Anagram Strings

Given two strings S1 and S2 . Return "1" if both strings are anagrams otherwise return "0" .

Note: An anagram of a string is another string with exactly the same quantity of each character in it, in any order.

Example 1:

Input: S1 = "cdbkdub" , S2 = "dsbkcsdn"

Output: 0

Explanation: Length of S1 is not same

as length of S2.

Example 2:

Input: S1 = "geeks" , S2 = "skgee"

Output: 1

Explanation: S1 has the same quantity

of each character in it as S2.

Your Task:
You don't need to read input or print anything. Your task is to complete the function **areAnagram()** which takes S1 and S2 as input and returns "1" if both strings are anagrams otherwise returns "0".

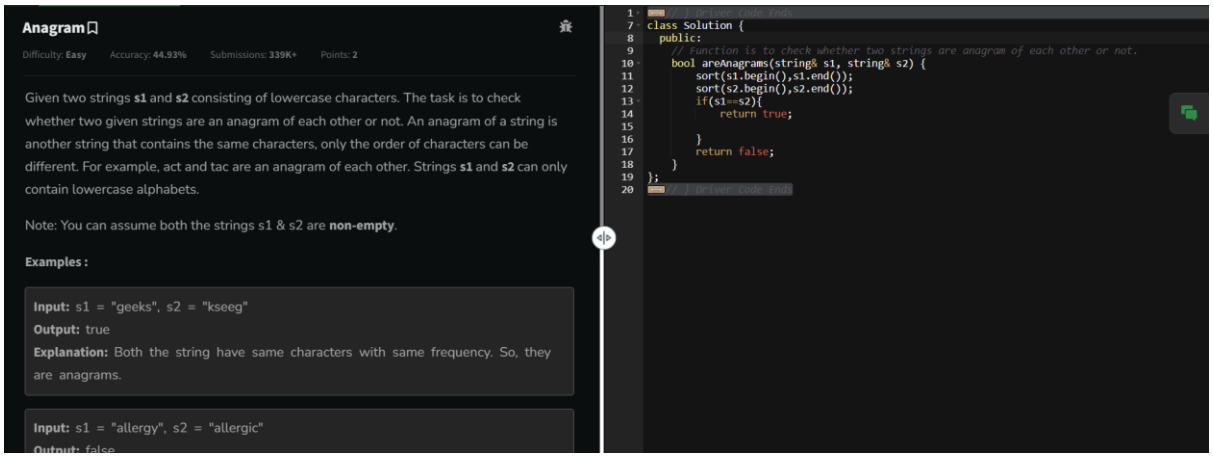
Expected Time Complexity: O(n)
Expected Auxiliary Space: O(K) ,Where K= Contstant

Constraints:
1 <= |S1| <= 1000
1 <= |S2| <= 1000

Program:

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;



// } Driver Code Ends
class Solution {
  public:
```

```
    // Function is to check whether two strings are anagram of each other or not.
    bool areAnagrams(string& s1, string& s2) {
        sort(s1.begin(),s1.end());
        sort(s2.begin(),s2.end());
        if(s1==s2){
            return true;

        }
        return false;
    }
};
```



2.maximum number of 1's row

Given a boolean 2D array, where each row is sorted. Find the row with the maximum number of 1s.

Example 1:

Input:

N = 3, M = 4

Mat[] = {{0 1 1 1},

{0 0 1 1},

{0 0 1 1}}

Output: 0

Explanation: Row 0 has 3 ones whereas

rows 1 and 2 have just 2 ones.

Example 2:

Input:

N = 2, M = 2

Mat[] = {{0 1},

{1 1}}

Output: 1

Explanation: Row 1 has 2 ones whereas

row 0 has just a single one.

Your Task:
You don't need to read input or print anything. Your task is to complete the function **maxOnes ()** which takes a 2D array Mat[][] and its dimensions N and M as inputs and returns the row index with the maximum number of 1s (0-based index). If there are multiple rows with the maximum number of ones, then return the row with the smaller index.

Expected Time Complexity: O(NLogM).
Expected Auxiliary Space: O(1).

Constraints:
1 <= N, M <= 40

Program:

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;



// } Driver Code Ends
class Solution
{
  public:
```

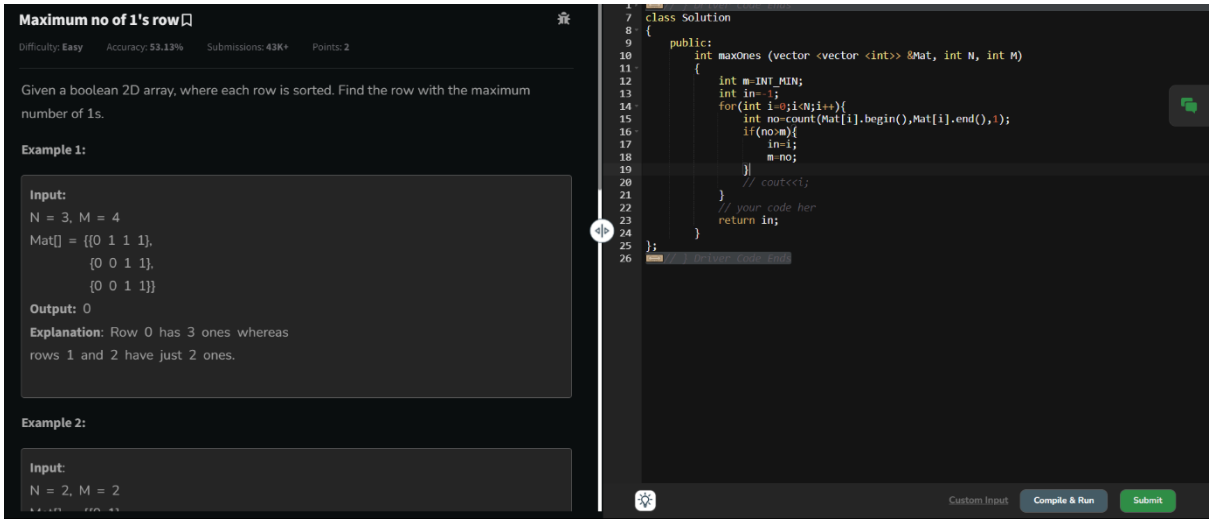
```
    int maxOnes (vector <vector <int>> &Mat, int N, int M)
    {
        int m=INT_MIN;
        int in=-1;
        for(int i=0;i<N;i++){
            int no=count(Mat[i].begin(),Mat[i].end(),1);
            if(no>m){
                in=i;
                m=no;
            }
            // cout<<i;
        }
        // your code her
        return in;
    }
};

//{ Driver Code Starts.

int main(){
    int t; cin >> t;
    while (t--){
        int n, m; cin >> n >> m;
        vector <vector <int>> arr (n, vector <int> (m));
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                cin >> arr[i][j];
        Solution ob;
        cout << ob.maxOnes(arr, n, m) << endl;

cout << "~" << "\n";
```

}

}

// } Driver Code Ends

Output:



3. **Longest consecutive subsequence**

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.

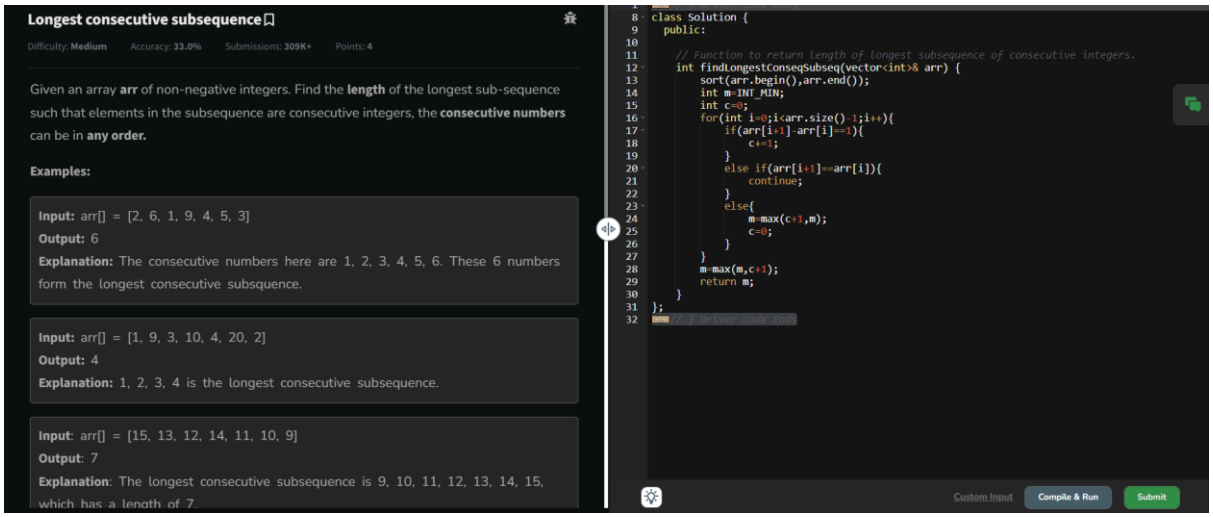
Constraints:
$1 <= arr.size() <= 10^5$
$0 <= arr[i] <= 10^5$

Program:

```
//{ Driver Code Starts
#include <bits/stdc++.h>


using namespace std;



// } Driver Code Ends
class Solution {
  public:

   // Function to return length of longest subsequence of consecutive integers.
   int findLongestConseqSubseq(vector<int>& arr) {
      sort(arr.begin(),arr.end());
      int m=INT_MIN;
      int c=0;
      for(int i=0;i<arr.size()-1;i++){
         if(arr[i+1]-arr[i]==1){
            c+=1;
         }
         else if(arr[i+1]==arr[i]){
            continue;
         }
         else{
            m=max(c+1,m);
            c=0;
         }
      }
      m=max(m,c+1);
```

```
        return m;
    }
};

//{ Driver Code Starts.
int main() {
    int t;
    cin >> t;
    cin.ignore();
    while (t--) {
        vector<int> arr;
        string input;

        // Read first array
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }

        Solution ob;
        int res = ob.findLongestConseqSubseq(arr);

        cout << res << endl << "~" << endl;
    }
    return 0;
}

// } Driver Code Ends
```

Output:



4. **Longest Palindrome in a String**

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 ≤ i ≤ j < 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.

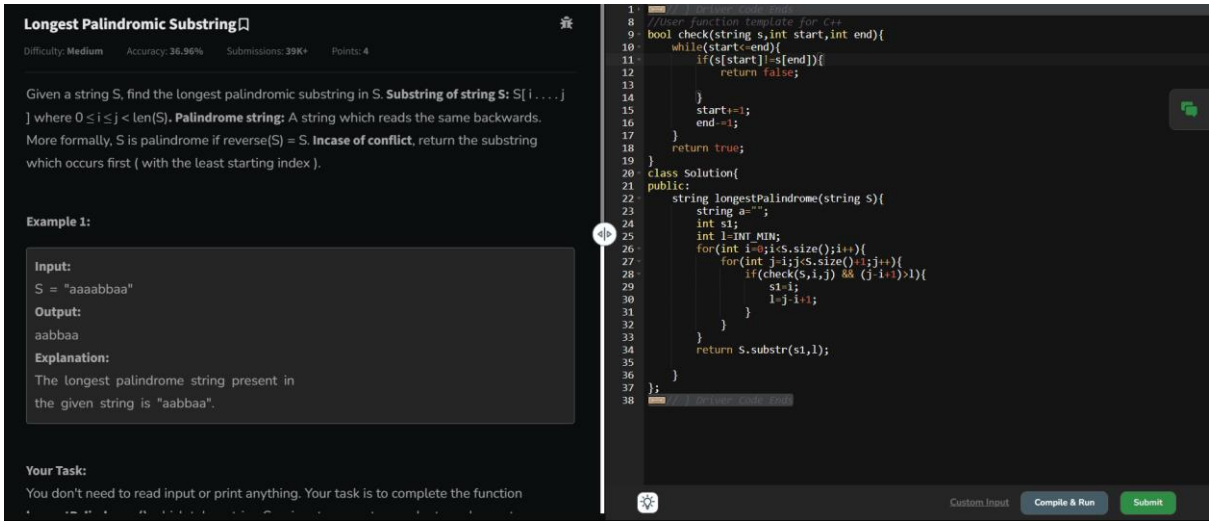
Input: s = "abacdfgdcaba"
Output: "aba"
Explanation: The longest palindromic substring is "aba", which occurs twice. The first occurrence is returned.

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

Program: //User function template for C++

```
bool check(string s,int start,int end){
   while(start<=end){
      if(s[start]!=s[end]){
         return false;


      }
      start+=1;
      end-=1;
   }
   return true;
}
class Solution{
public:
   string longestPalindrome(string S){
      string a="";
      int s1;
      int l=INT_MIN;
      for(int i=0;i<S.size();i++){
         for(int j=i;j<S.size()+1;j++){
            if(check(S,i,j) && (j-i+1)>l){
               s1=i;
               l=j-i+1;
            }
         }
      }
      return S.substr(s1,l);


   }
};
```

Output:



5. **Rat in a Maze Problem - I**

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.

Constraints:
2 ≤ n ≤ 5
0 ≤ mat[i][j] ≤ 1

Program:

```
//{ Driver Code Starts
// Initial template for C++

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



// } Driver Code Ends
// User function template for C++

class Solution {
   void solve(int i,int j,vector<vector<int>> &a,int n,vector<string> &ans,string move,
vector<vector<int>> &vis ,vector<int> &di,vector<int> &dj){
      if(i==n-1 && j==n-1){
         ans.push_back(move);
         return ;
      }
      string d="DLRU";
      for(int ind=0;ind<4;ind++){
         int ni=di[ind]+i;
         int nj=dj[ind]+j;
         if(ni>=0 && nj>=0 && !vis[ni][nj] && ni<n && nj<n &&a[ni][nj]==1 ){
            vis[i][j]=1;
            solve(ni,nj,a,n,ans,move+d[ind],vis,di,dj);
            vis[i][j]=0;
         }
      }
   }
```

```
  public:
    vector<string> findPath(vector<vector<int>> &mat) {
        // Your code goes here
        int n=mat.size();
        vector<vector<int>> vis(n+1,vector<int>(n+1,0));

        vector<int> di={1,0,0,-1};
        vector<int> dj={0,-1,1,0};
        vector<string> ans;
        if(mat[0][0]==1){
            solve(0,0,mat,n,ans,"",vis,di,dj);
        }
        return ans;
    }
};



//{ Driver Code Starts.

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<vector<int>> m(n, vector<int>(n, 0));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cin >> m[i][j];
            }
        }
```

```
    Solution obj;
    vector<string> result = obj.findPath(m);
    sort(result.begin(), result.end());
    if (result.size() == 0)
        cout << -1;
    else
        for (int i = 0; i < result.size(); i++)
            cout << result[i] << " ";
    cout << endl;

    cout << "~"
         << "\n";
  }
  return 0;
}
// } Driver Code Ends
```

Output: