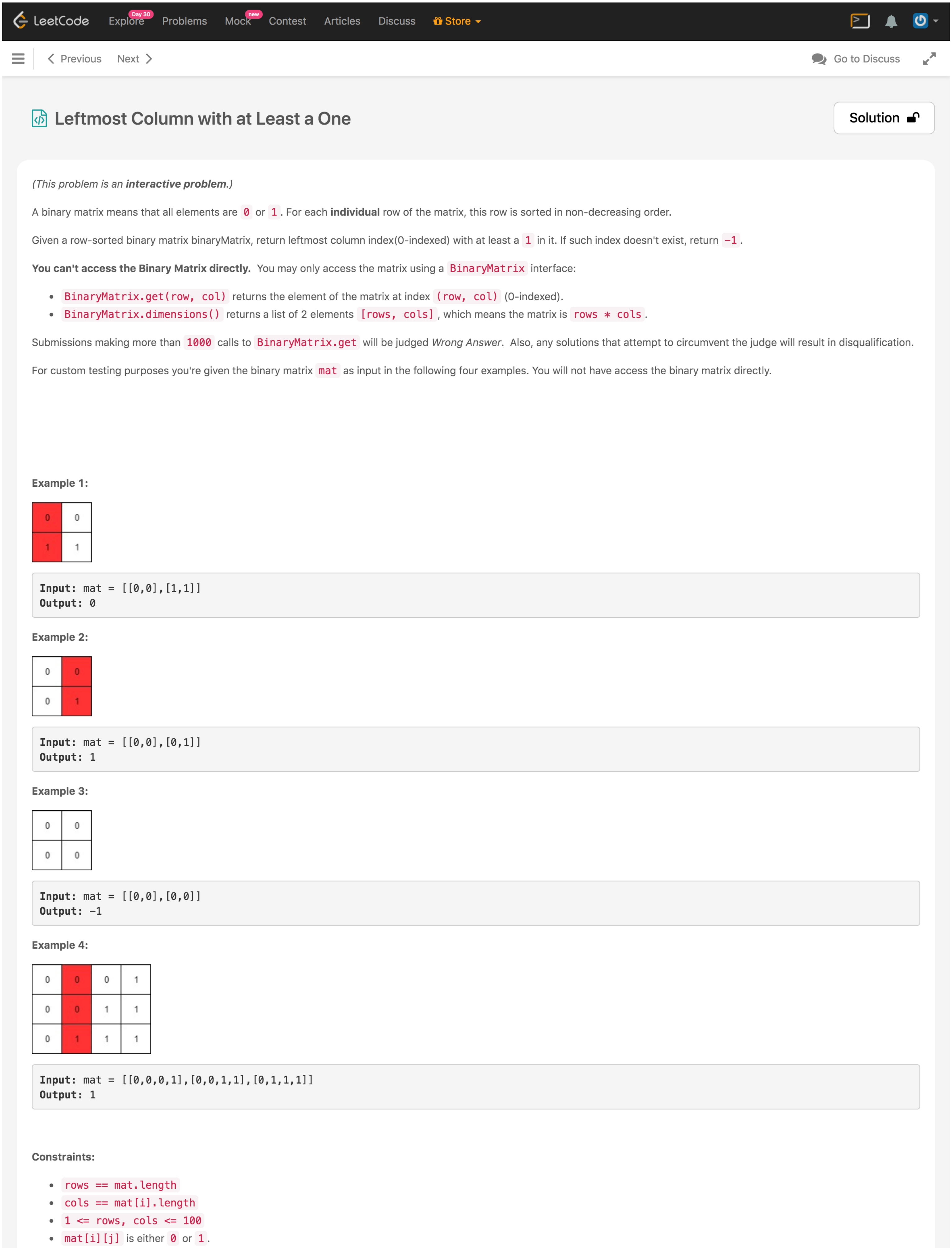


Leftmost Column with at Least a One

(This problem is an ***interactive problem****.)*

A binary matrix means that all elements are `0` or `1`. For each **individual** row of the matrix, this row is sorted in non-decreasing order.

Given a row-sorted binary matrix binaryMatrix, return leftmost column index(0-indexed) with at least a `1` in it. If such index doesn't exist, return `-1`.

You can't access the Binary Matrix directly. You may only access the matrix using a `BinaryMatrix` interface:

- `BinaryMatrix.get(row, col)` returns the element of the matrix at index `(row, col)` (0-indexed).
- `BinaryMatrix.dimensions()` returns a list of 2 elements `[rows, cols]`, which means the matrix is `rows * cols`.

Submissions making more than `1000` calls to `BinaryMatrix.get` will be judged *Wrong Answer*. Also, any solutions that attempt to circumvent the judge will result in disqualification.

For custom testing purposes you're given the binary matrix `mat` as input in the following four examples. You will not have access the binary matrix directly.

Example 1:

0	0
1	1

```
Input: mat = [[0,0],[1,1]]
Output: 0
```

Example 2:

0	0
0	1

```
Input: mat = [[0,0],[0,1]]
Output: 1
```

Example 3:

0	0
0	0

```
Input: mat = [[0,0],[0,0]]
Output: -1
```

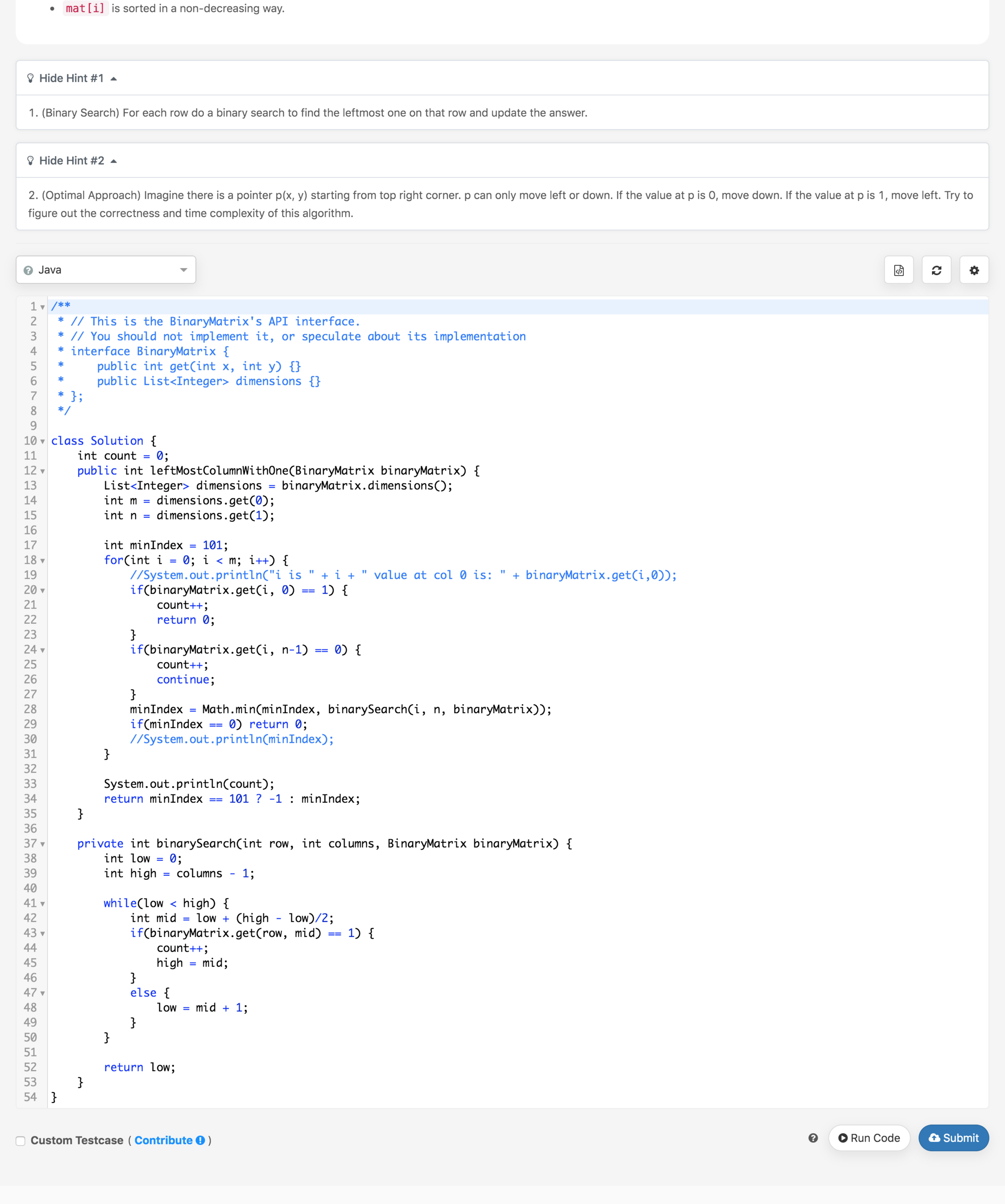
Example 4:

0	0	0	1
0	0	1	1
0	1	1	1

```
Input: mat = [[0,0,0,1],[0,0,1,1],[0,1,1,1]]
Output: 1
```

Constraints:

- `rows == mat.length`
- `cols == mat[i].length`
- `1 <= rows, cols <= 100`
- `mat[i][j]` is either `0` or `1`.



- `mat[i]` is sorted in a non-decreasing way.

Hide Hint #1

1. (Binary Search) For each row do a binary search to find the leftmost one on that row and update the answer.

Hide Hint #2

2. (Optimal Approach) Imagine there is a pointer p(x, y) starting from top right corner. p can only move left or down. If the value at p is 0, move down. If the value at p is 1, move left. Try to figure out the correctness and time complexity of this algorithm.

```java
/**
 * // This is the BinaryMatrix's API interface.
 * // You should not implement it, or speculate about its implementation
 * interface BinaryMatrix {
 *     public int get(int x, int y) {}
 *     public List<Integer> dimensions {}
 * };
 */

class Solution {
    int count = 0;
    public int leftMostColumnWithOne(BinaryMatrix binaryMatrix) {
        List<Integer> dimensions = binaryMatrix.dimensions();
        int m = dimensions.get(0);
        int n = dimensions.get(1);

        int minIndex = 101;
        for(int i = 0; i < m; i++) {
            //System.out.println("i is " + i + " value at col 0 is: " + binaryMatrix.get(i,0));
            if(binaryMatrix.get(i, 0) == 1) {
                count++;
                return 0;
            }
            if(binaryMatrix.get(i, n-1) == 0) {
                count++;
                continue;
            }
            minIndex = Math.min(minIndex, binarySearch(i, n, binaryMatrix));
            if(minIndex == 0) return 0;
            //System.out.println(minIndex);
        }

        System.out.println(count);
        return minIndex == 101 ? -1 : minIndex;
    }

    private int binarySearch(int row, int columns, BinaryMatrix binaryMatrix) {
        int low = 0;
        int high = columns - 1;

        while(low < high) {
            int mid = low + (high - low)/2;
            if(binaryMatrix.get(row, mid) == 1) {
                count++;
                high = mid;
            }
            else {
                low = mid + 1;
            }
        }

        return low;
    }
}
```