(M)aths : Video - 44

Leetcode - 1895

Medium

∞ → codestorywithmik

𝕏 → CSwithMIK

→ codestorywithMIK

WeekendWithMIK

WeekendWithMIK

@WeekendWithMIK [redacted]

Welcome to WeekendWithMIK 🎥✨ >

[redacted]

MiK.

Try this channel to

see "Life behind the Scenes + Tech News

Motivation -

Consistency → Honest?

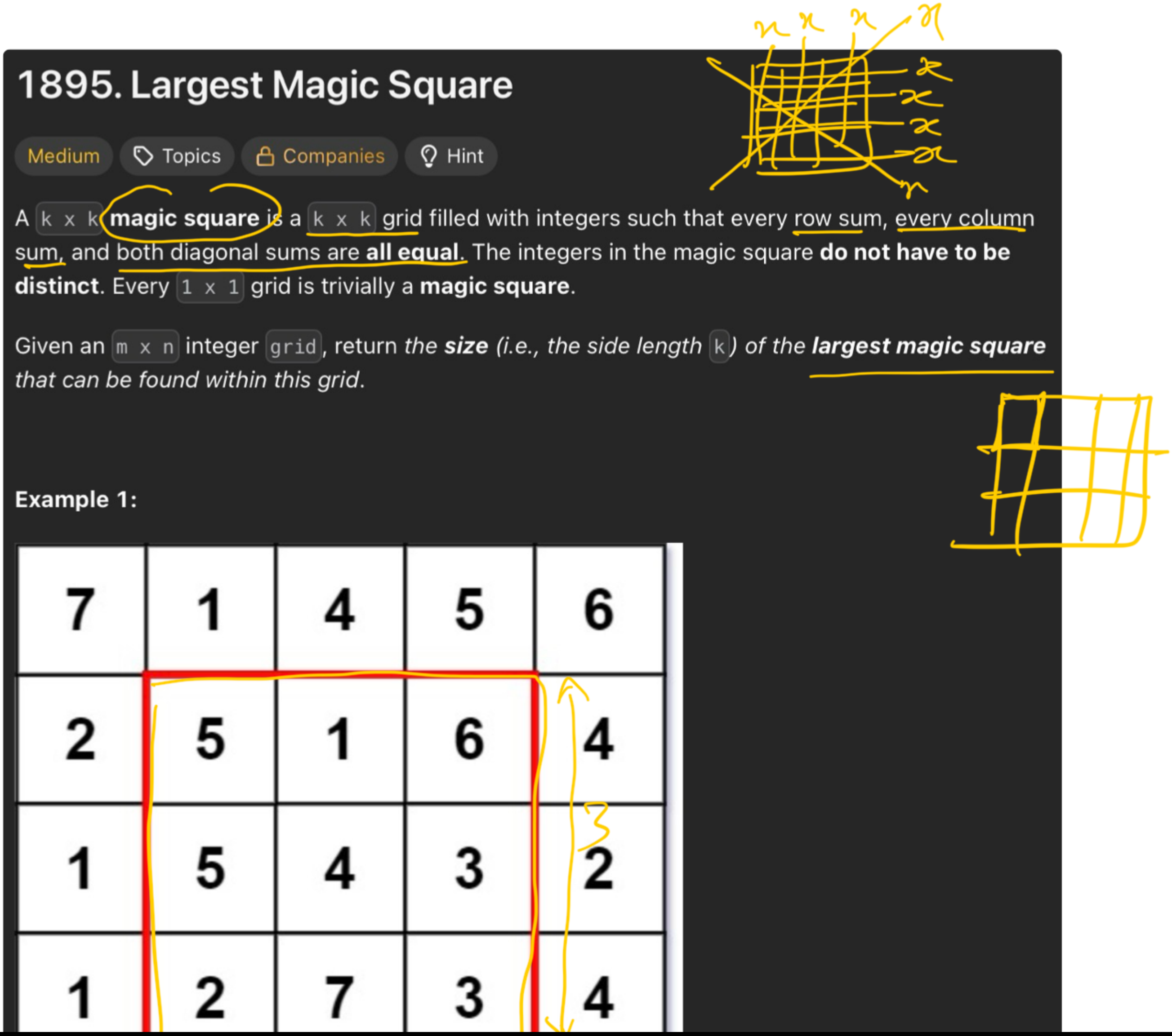
Consistency is what transforms

AVERAGE → EXCELLENCE

MIK...

you want to stay Average ???

or achieve excellence ???



1895. Largest Magic Square

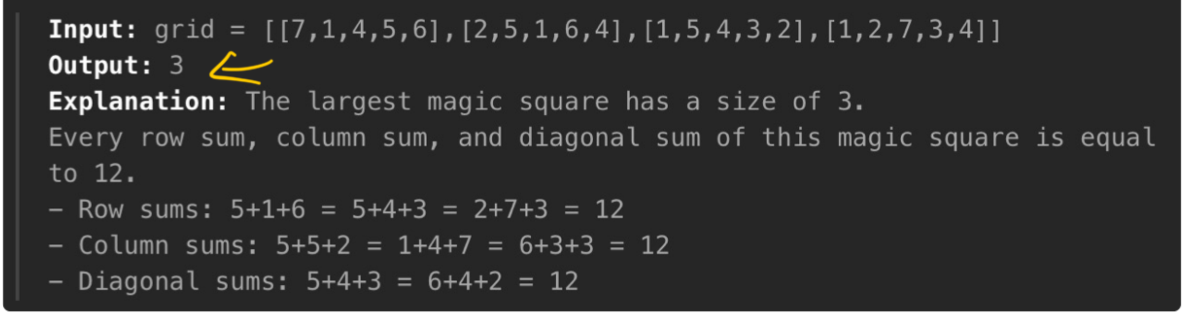
Medium | Topics | Companies | Hint

A `k x k` **magic square** is a `k x k` grid filled with integers such that every row sum, every column sum, and both diagonal sums are **all equal**. The integers in the magic square **do not have to be distinct**. Every `1 x 1` grid is trivially a **magic square**.

Given an `m x n` integer `grid`, return *the* ***size*** *(i.e., the side length* `k`*) of the* ***largest magic square*** *that can be found within this grid.*

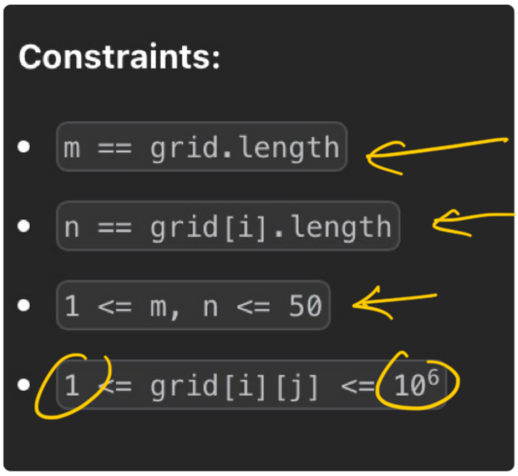
Example 1:

7	1	4	5	6
2	5	1	6	4
1	5	4	3	2
1	2	7	3	4



Input: grid = [[7,1,4,5,6],[2,5,1,6,4],[1,5,4,3,2],[1,2,7,3,4]]
Output: 3
Explanation: The largest magic square has a size of 3.
Every row sum, column sum, and diagonal sum of this magic square is equal to 12.
- Row sums: 5+1+6 = 5+4+3 = 2+7+3 = 12
- Column sums: 5+5+2 = 1+4+7 = 6+3+3 = 12
- Diagonal sums: 5+4+3 = 6+4+2 = 12

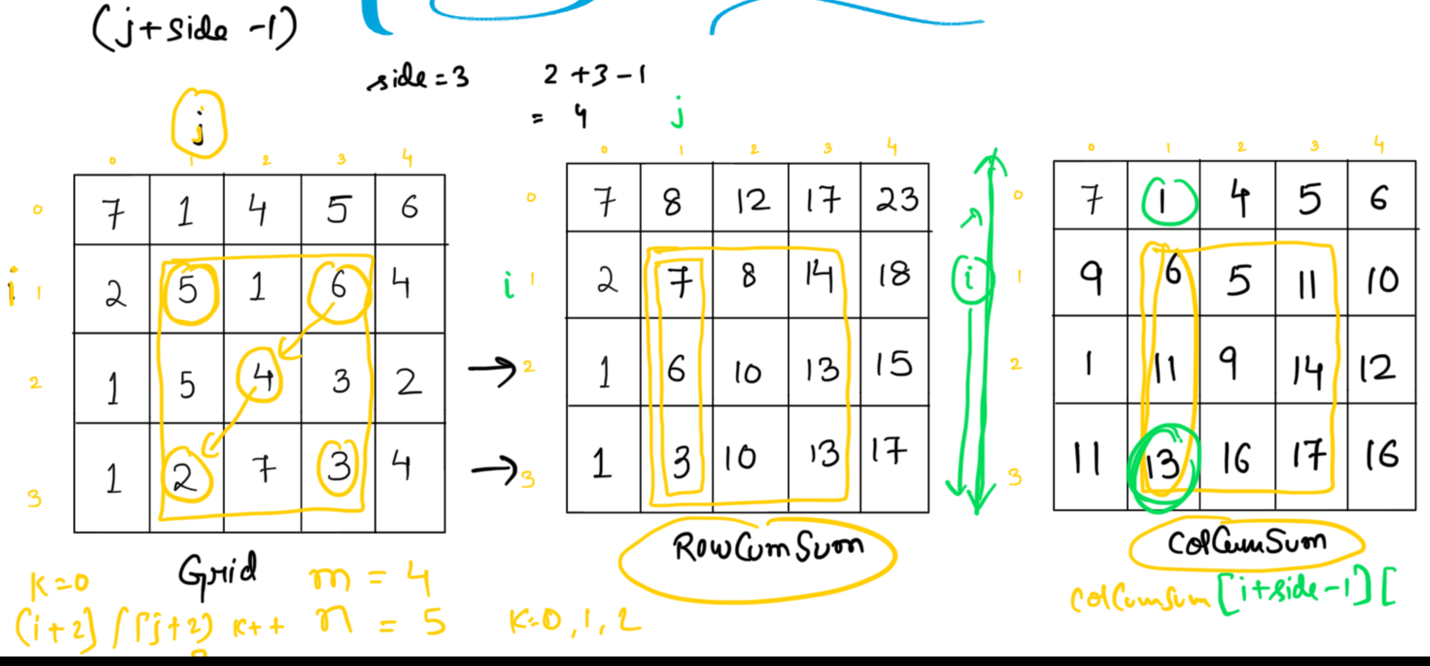
Constraint Analysis :-



Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 50`
- `1 <= grid[i][j] <= 10^6`

Thought Process



(j+side -1)

side=3

2 +3 −1
= 4

	0	1	2	3	4
0	7	1	4	5	6
1	2	5	1	6	4
2	1	5	4	3	2
3	1	2	7	3	4

Grid

K=0
(i+2] / [j+2) K++

m = 4
n = 5

K=0, 1, 2

	0	1	2	3	4
0	7	8	12	17	23
1	2	7	8	14	18
2	1	6	10	13	15
3	1	3	10	13	17

RowCumSum

	0	1	2	3	4
0	7	1	4	5	6
1	9	6	5	11	10
2	1	11	9	14	12
3	11	13	16	17	16

ColCumSum

colCumSum [i+side-1] [

5 + 4 + 3 =

min(m,n)

Check every possible square.

K = 1

grid[i+K] [j+side-1-K]

grid[2][1+3-1-1]

[2] [2]

Square = side = 1×1

2*2

3*3 ←

4*4

5*5

. . .

ans = 1

```
Side = min(m,n) ;    side >= 2
```

→ Slight improvement

```
for ( side = min(m,n) ;  side >= 2 ;  side --)  {

    for ( i = 0  ;  i+side-1<m ;     )  {

        for (j = 0 ; j+side-1 < n ; j++

            // (i,j)
            // side

            int targetSum = rowCumSum [i] [j+side-1] — (j >0 ?
                                 rowCumSum [i] [j-1]  : 0)
O(1)

            bool allSame = True,

            // check all remai rows

            for (int r = i+1 ; r < i+side ; r++) {
                int Sum = rowCumSum[r] [j+side-1] —(j>0?
                               rowCu[r] [j -1] : 0) ;
                if (Sum != targetSum) {
```

```
        allSame = False;
        break;
    }
}
if (allSame == False)
    continue;
```

// Check all columns.

```
for (c = j ; c < j+side ; c++) {
    int sum = colCumSum[i+side-1][c]
              - colCumSum[i-1][c] ;
                   i > 0     : 0
    if (sum != target)
        allSa = Fa
        br- .
}
if (!allSame) continue;
diag = 0 ; antidi = 0;
for (int K = 0 ; K < side ; K++) {
    diag += grid[i+K][j+K] ;
    antidig += grid[i+K][j+side-1-K];
}
if (diag == targetSum && antidi == ta)
    return side;
}
)
}
}
```

Time complexity:

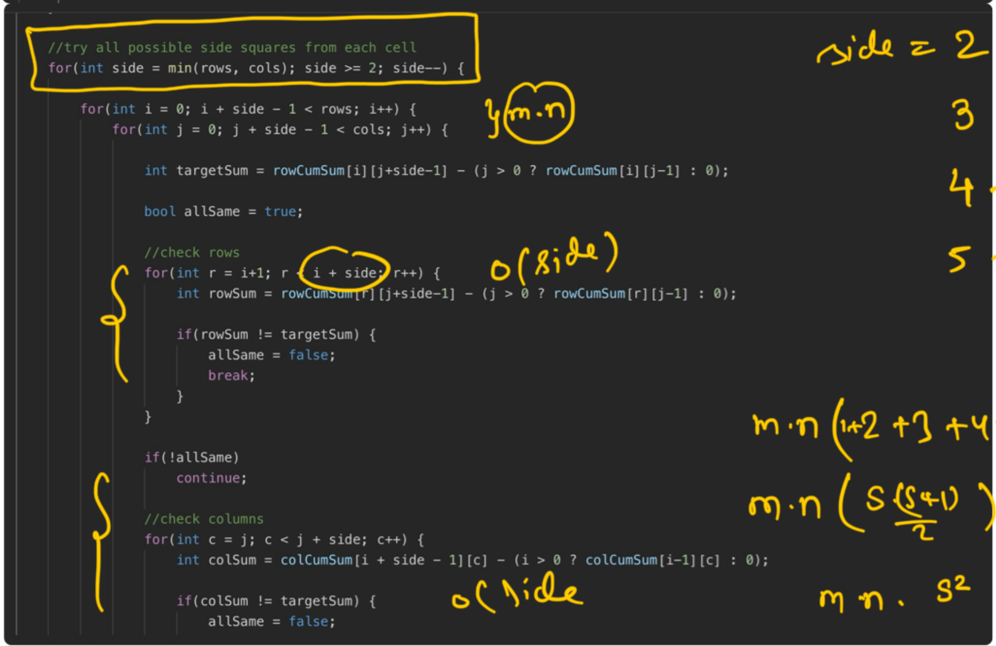
```
class Solution {
public:
    int largestMagicSquare(vector<vector<int>>& grid) {
        int rows = grid.size(); //m
        int cols = grid[0].size(); //n

        //Row wise prefix sum
        vector<vector<int>> rowCumSum(rows, vector<int>(cols));
        for(int i = 0; i < rows; i++) {
            rowCumSum[i][0] = grid[i][0];
            for(int j = 1; j < cols; j++) {
                rowCumSum[i][j] = rowCumSum[i][j-1] + grid[i][j];
            }
        }

        //Column wise prefix sum
        vector<vector<int>> colCumSum(rows, vector<int>(cols));
        for(int j = 0; j < cols; j++) {
            colCumSum[0][j] = grid[0][j];
            for(int i = 1; i < rows; i++) {
                colCumSum[i][j] = colCumSum[i-1][j] + grid[i][j];
            }
        }
```

$O(m \times n)$

$O(m \times n)$



```
        //try all possible side squares from each cell
        for(int side = min(rows, cols); side >= 2; side--) {

            for(int i = 0; i + side - 1 < rows; i++) {
                for(int j = 0; j + side - 1 < cols; j++) {

                    int targetSum = rowCumSum[i][j+side-1] - (j > 0 ? rowCumSum[i][j-1] : 0);

                    bool allSame = true;

                    //check rows
                    for(int r = i+1; r < i + side; r++) {
                        int rowSum = rowCumSum[r][j+side-1] - (j > 0 ? rowCumSum[r][j-1] : 0);

                        if(rowSum != targetSum) {
                            allSame = false;
                            break;
                        }
                    }

                    if(!allSame)
                        continue;

                    //check columns
                    for(int c = j; c < j + side; c++) {
                        int colSum = colCumSum[i + side - 1][c] - (i > 0 ? colCumSum[i-1][c] : 0);

                        if(colSum != targetSum) {
                            allSame = false;
```

$m \cdot n$

$O(side)$

$O(side$

side = 2 → $m \cdot n \cdot 2$

3 → $m \cdot n \cdot 3$

4 → $m \cdot n \cdot 4$

5 → $m \cdot n \cdot 5$

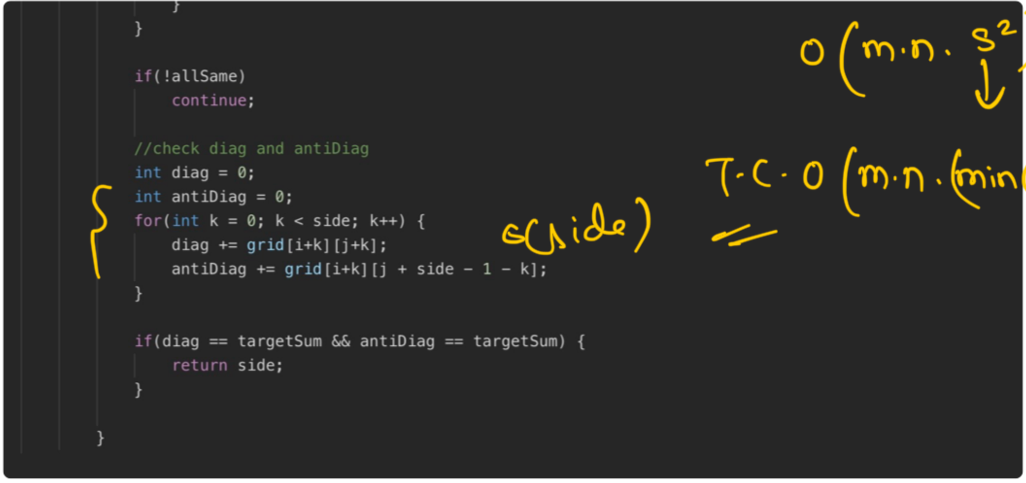
.

:

$m \cdot n \left(1+2+3+4+5+\ldots\right)$

$m \cdot n \left(\frac{S(S+1)}{2}\right)$

$m \cdot n \cdot S^2$



```
                        }
                    }

                    if(!allSame)
                        continue;

                    //check diag and antiDiag
                    int diag = 0;
                    int antiDiag = 0;
                    for(int k = 0; k < side; k++) {
                        diag += grid[i+k][j+k];
                        antiDiag += grid[i+k][j + side - 1 - k];
                    }

                    if(diag == targetSum && antiDiag == targetSum) {
                        return side;
                    }

                }
```

$O(m \cdot n \cdot \underset{\downarrow}{S^2})$

T.C. $O(m \cdot n \cdot (min(m,n))^2)$.

$O(side)$