Day 6

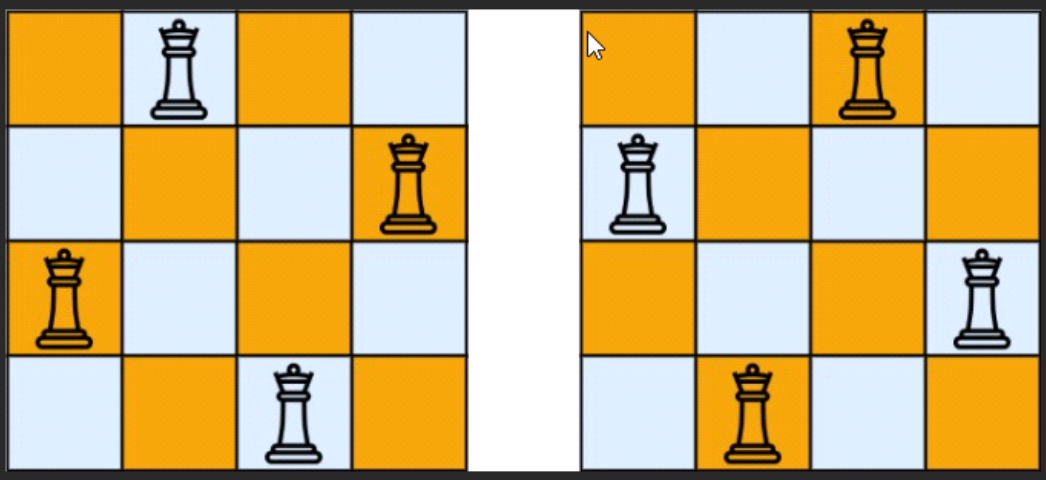
30 November 2022 14:32

The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

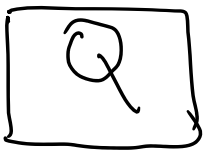
Example 1:



Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above

n = 1 to 9

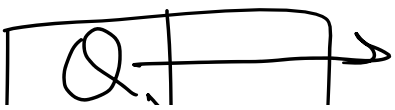
n = 1



* हमारा हर row में Queen

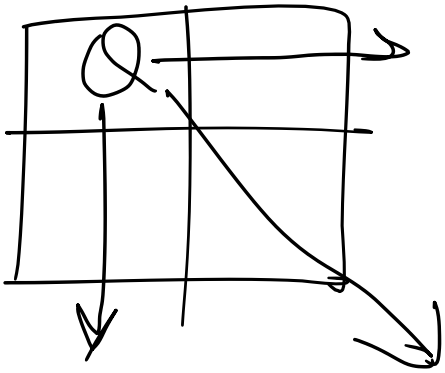
* हर column में Queen

n = 2

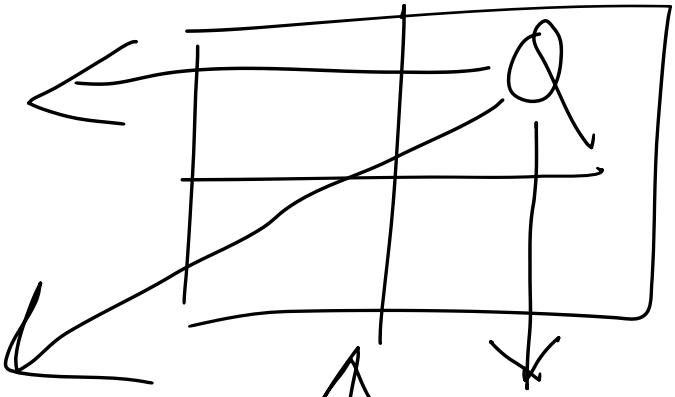


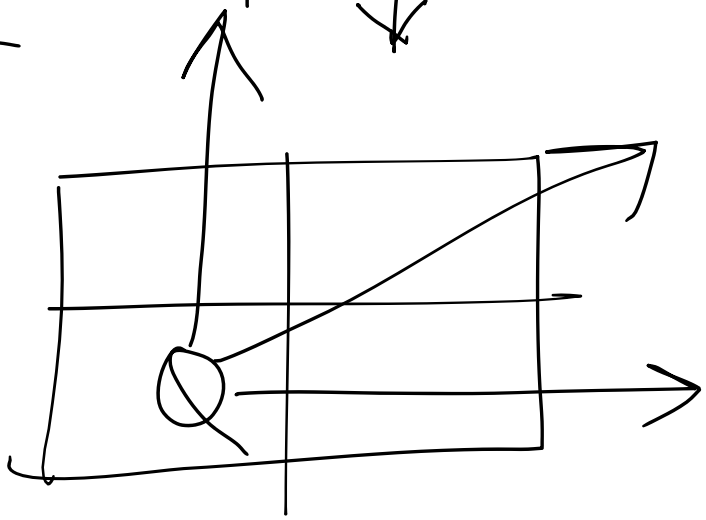
* No conflict

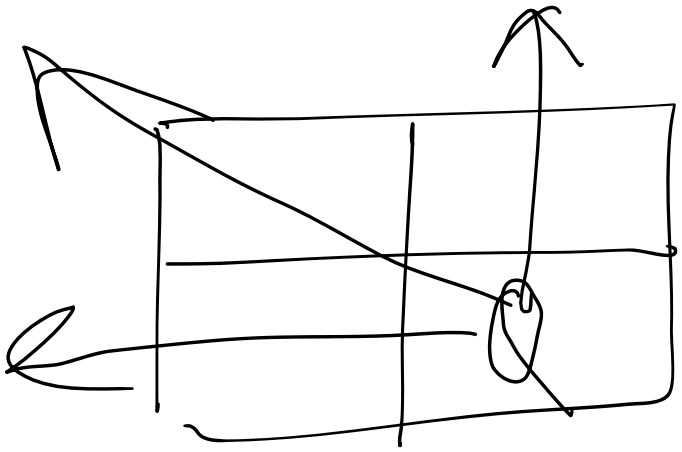
n=2



☆ No conflict



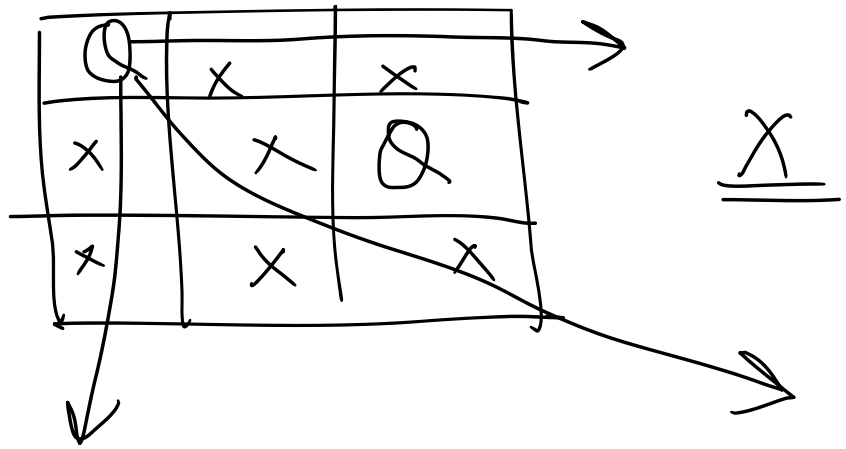




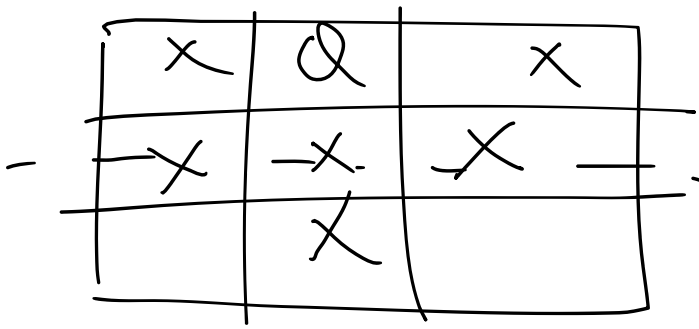
n=2

[ ]

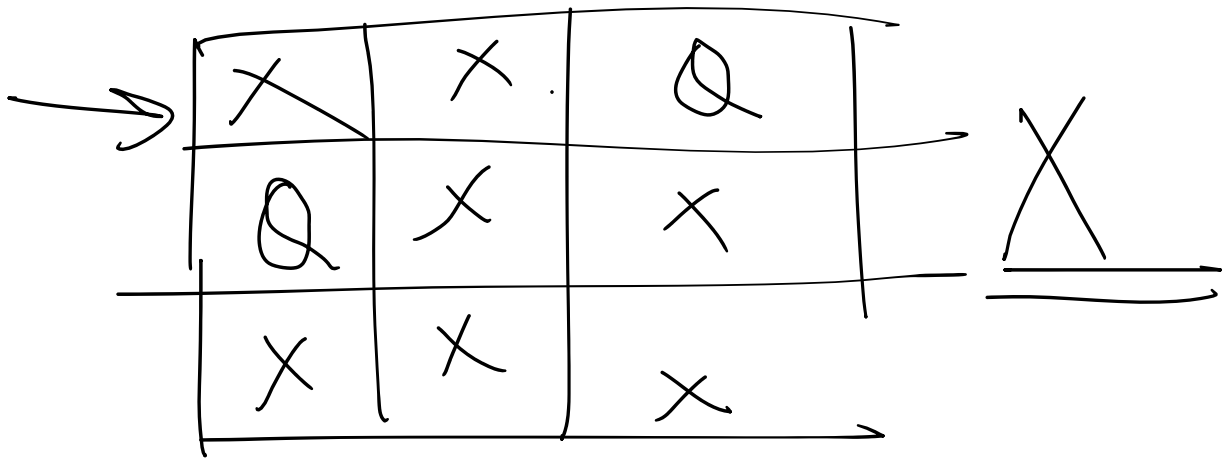
n=3



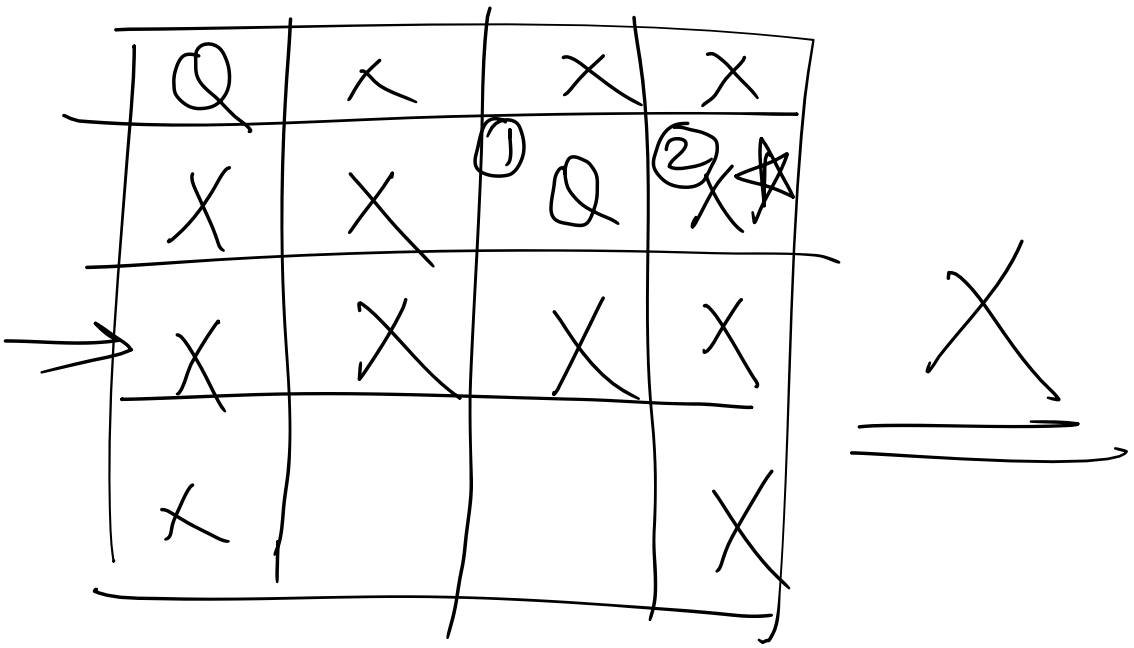
X

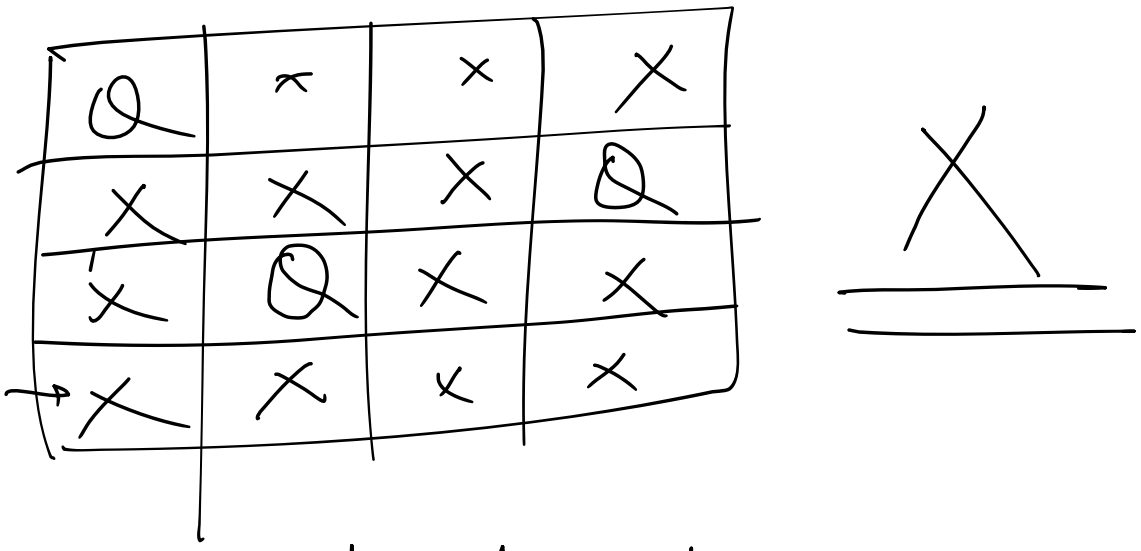


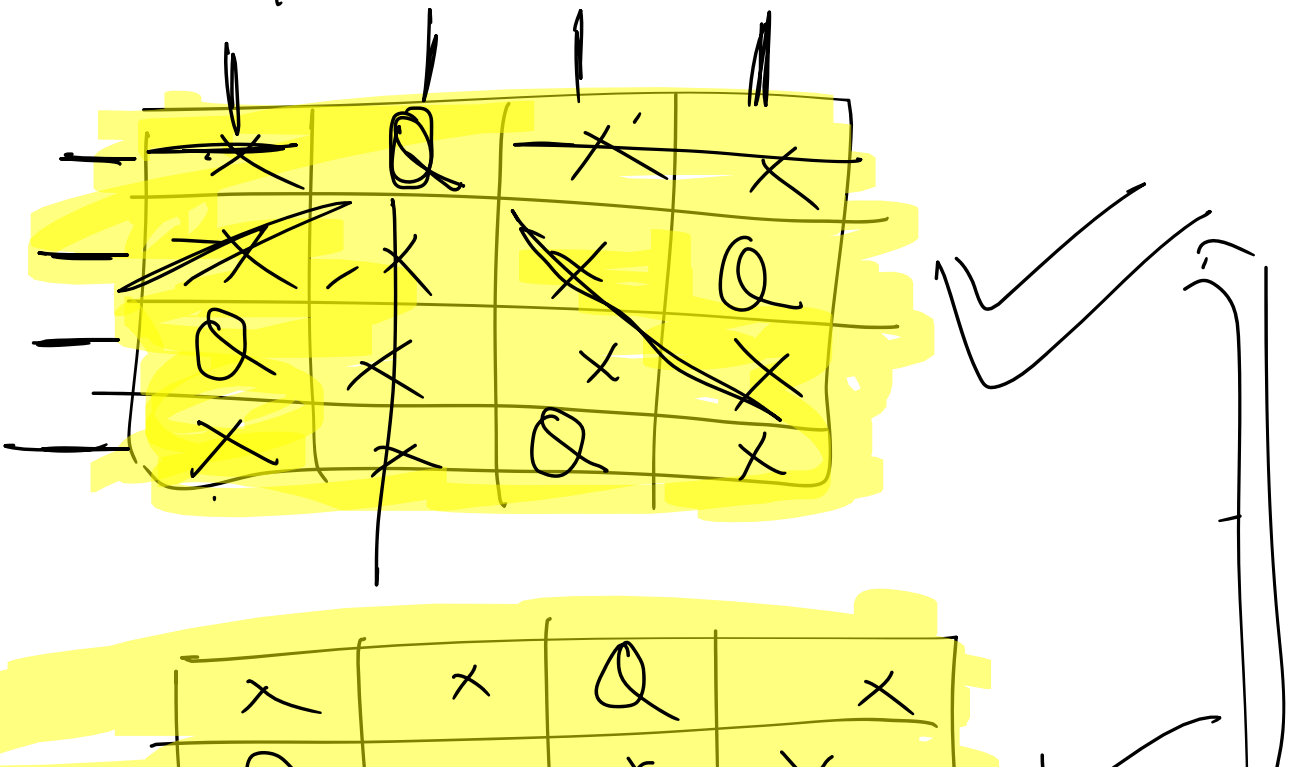
X

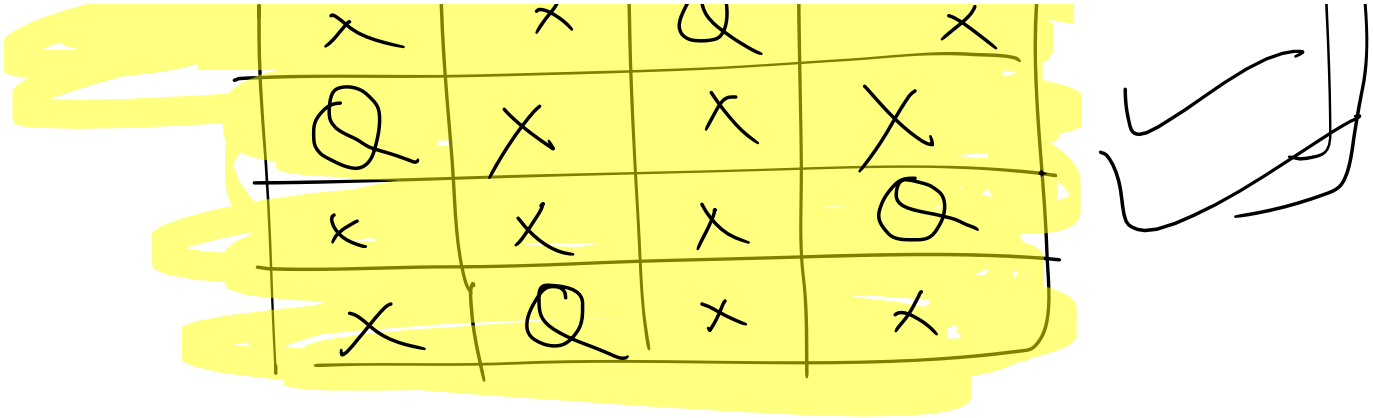


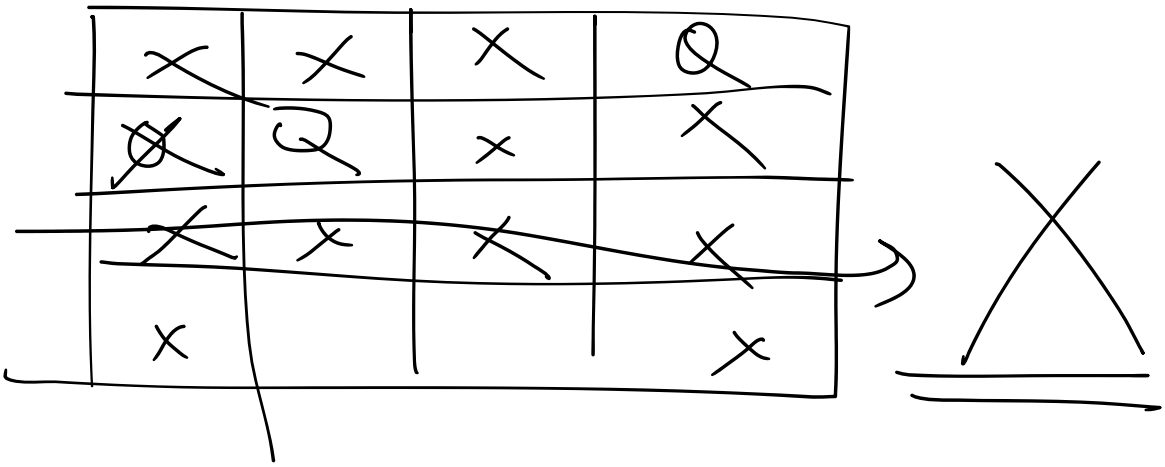
n=4

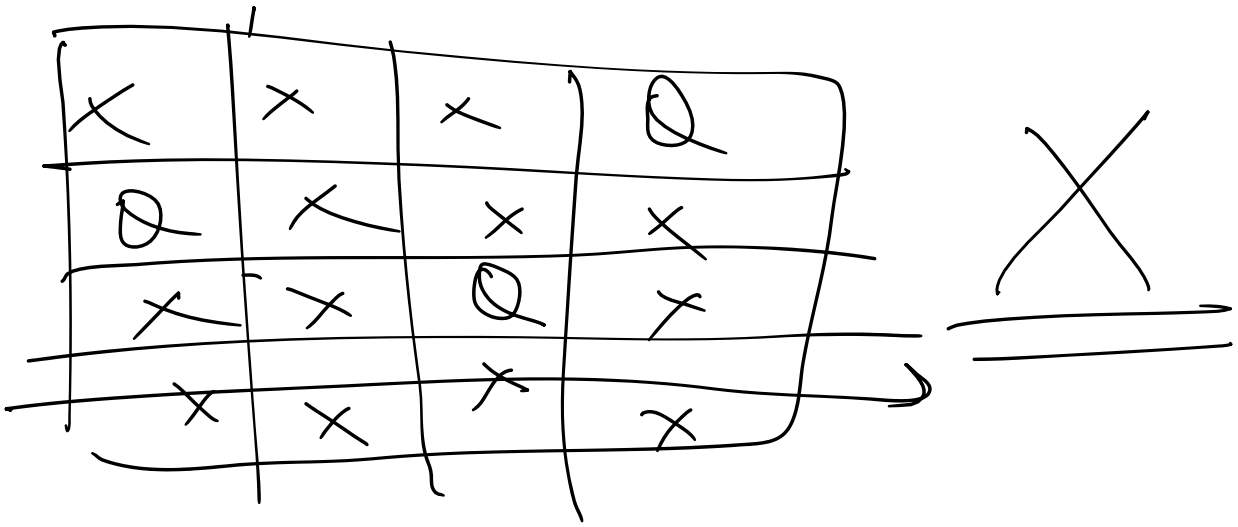








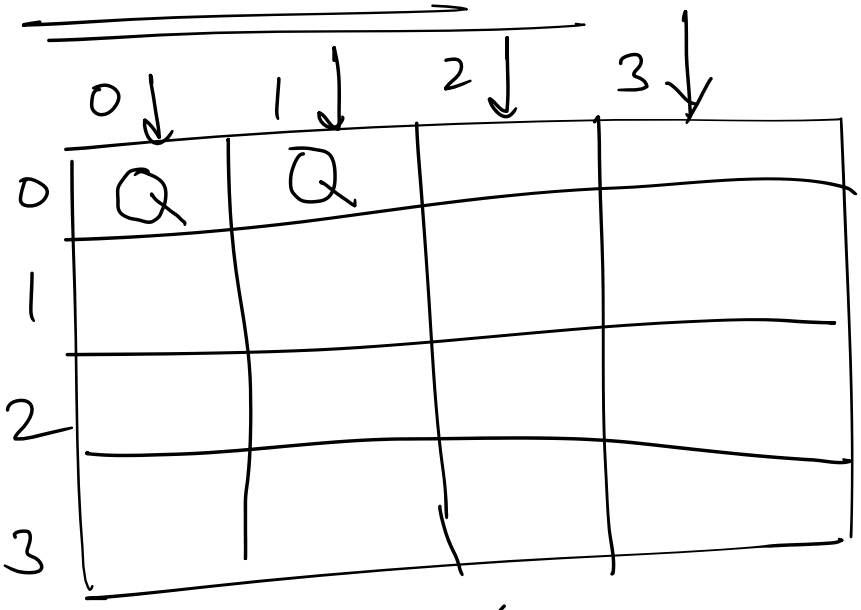




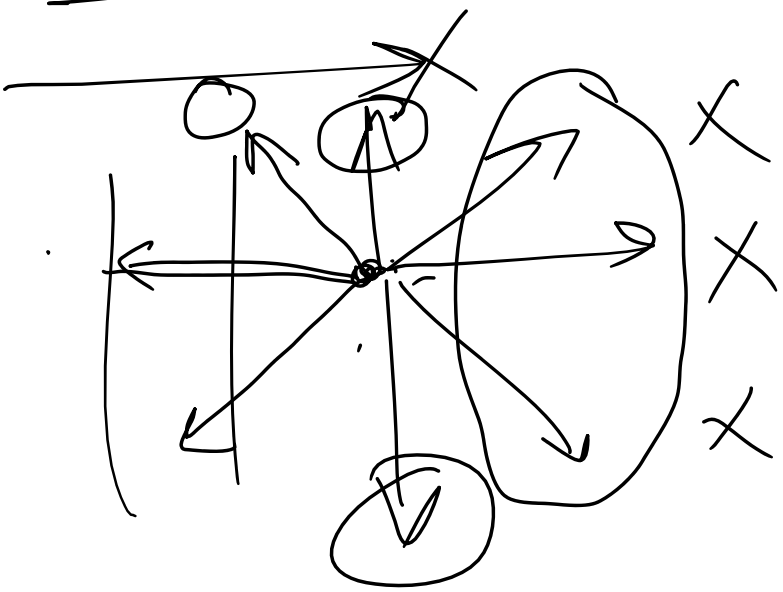
2 ते choices Explore

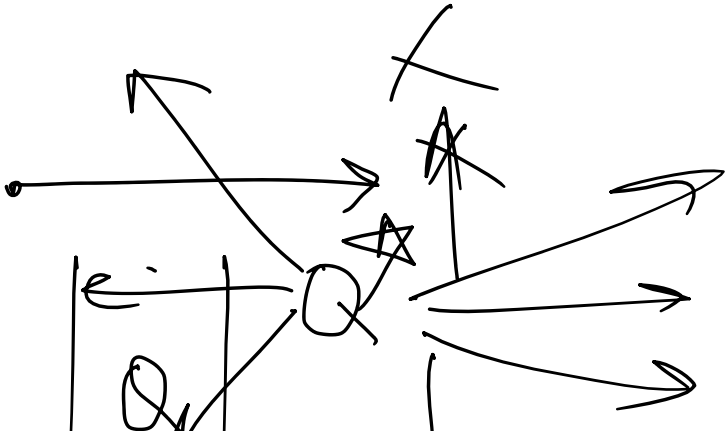
Recursion + Backtracking

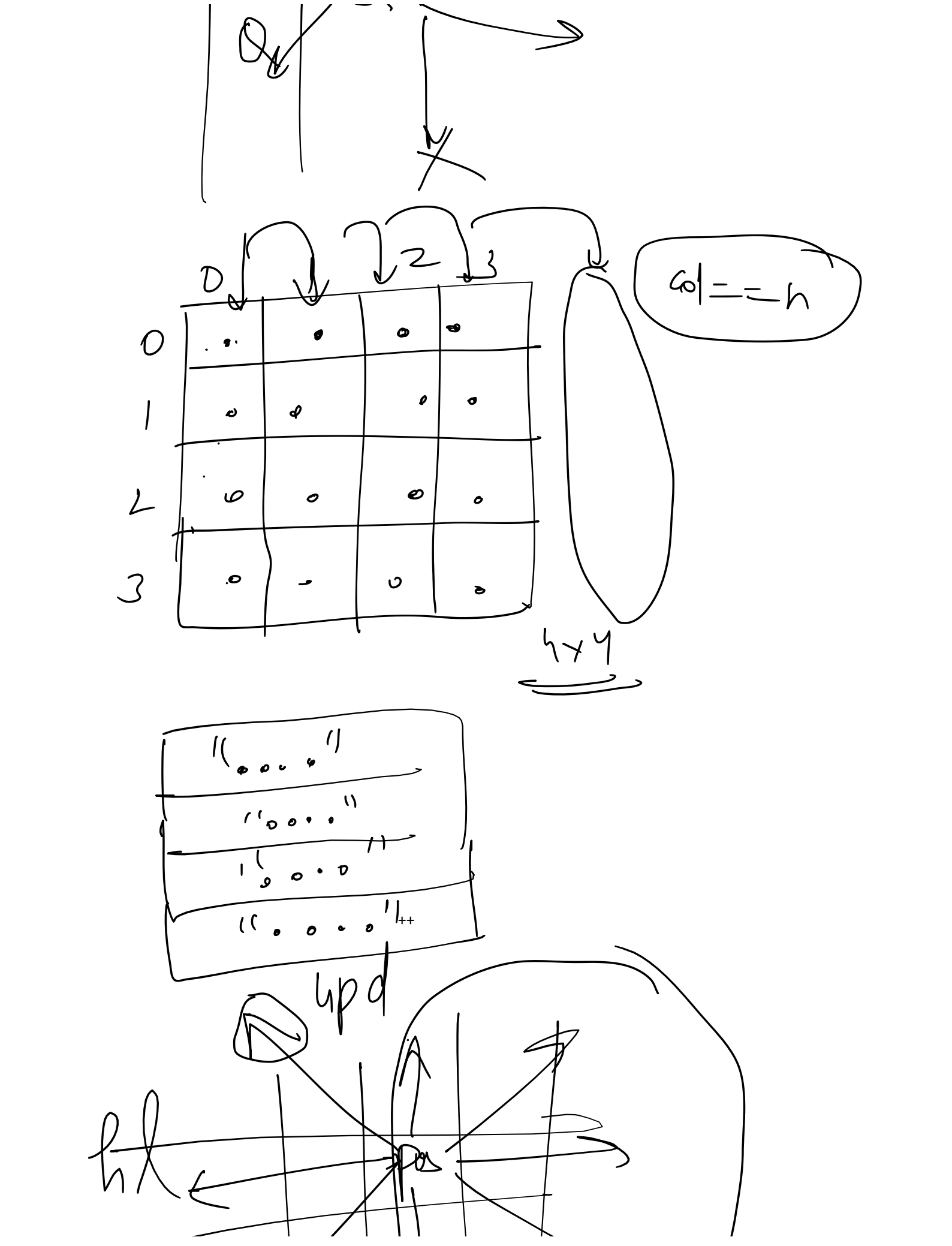
Pseudocode

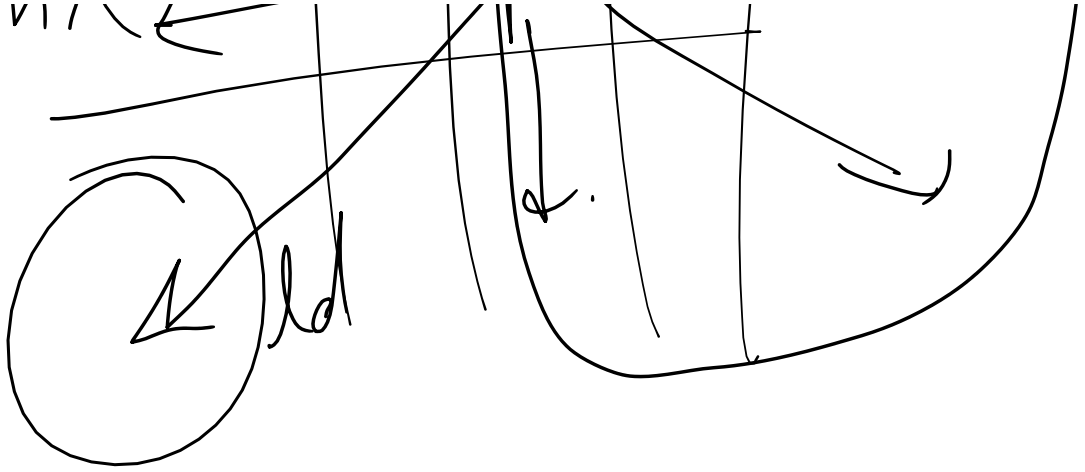


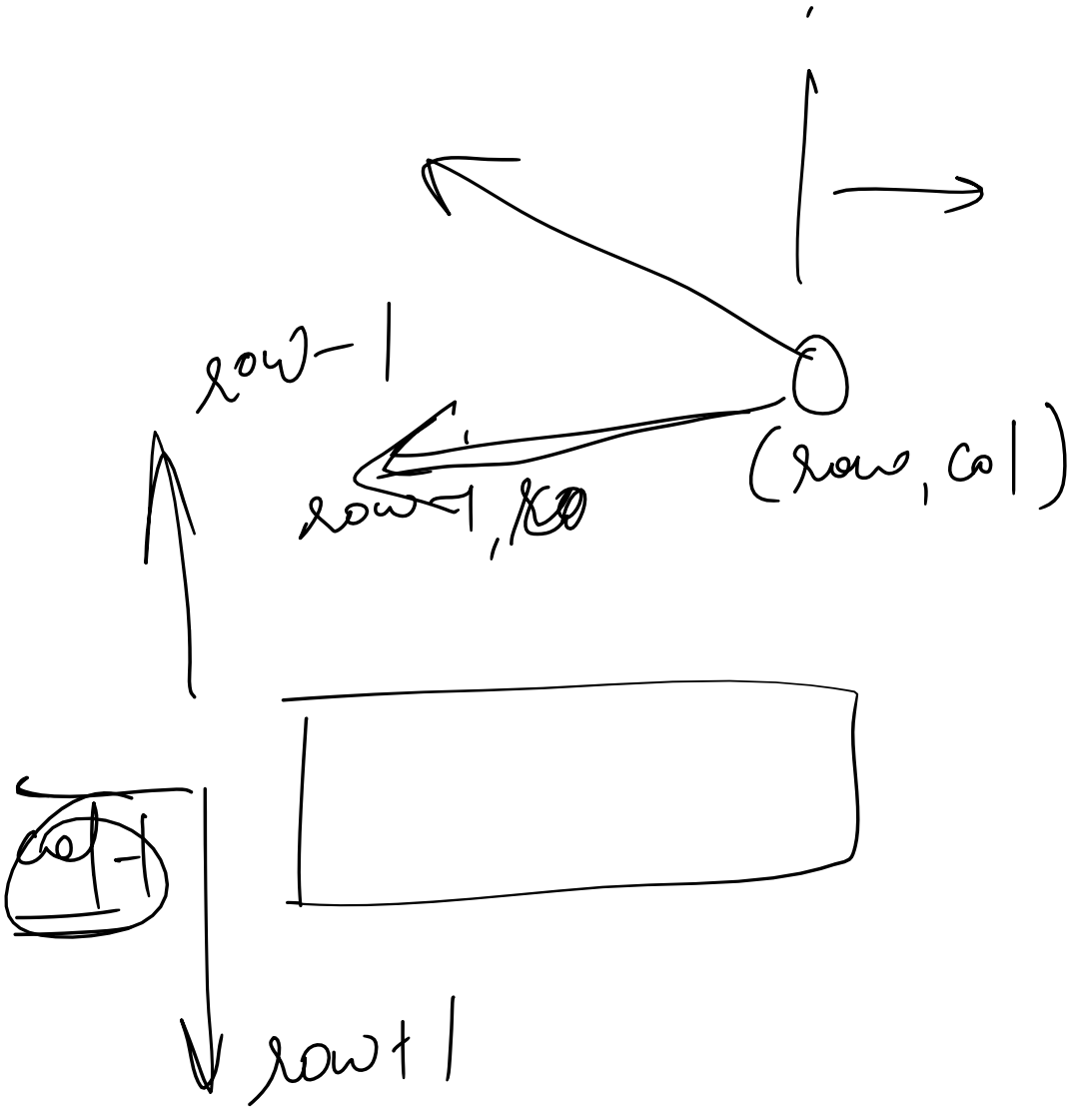
n x n

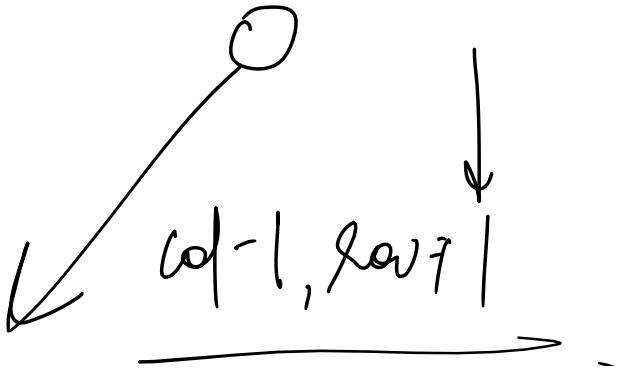


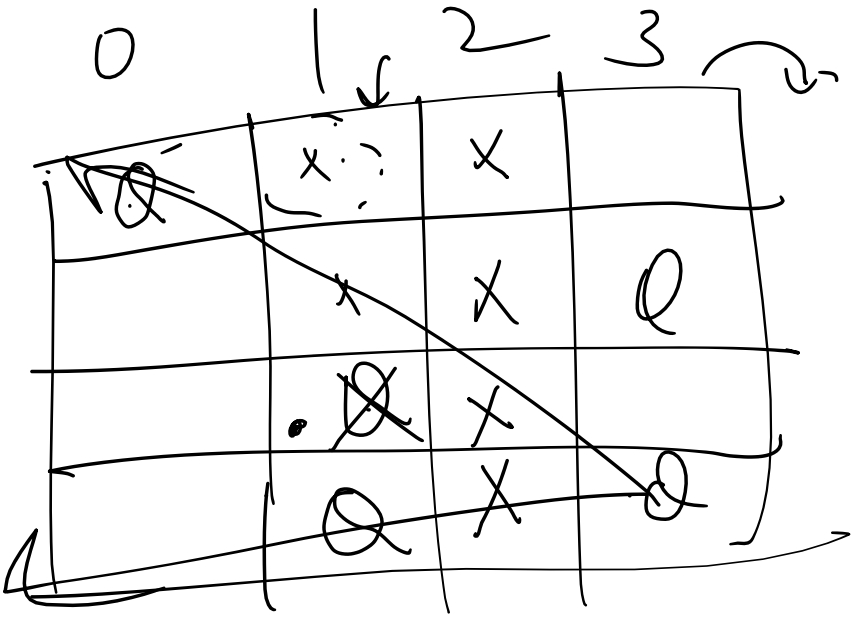



0
1
2
3
4x4
upd

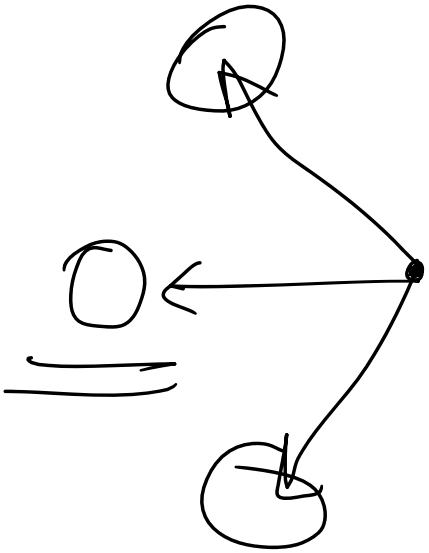


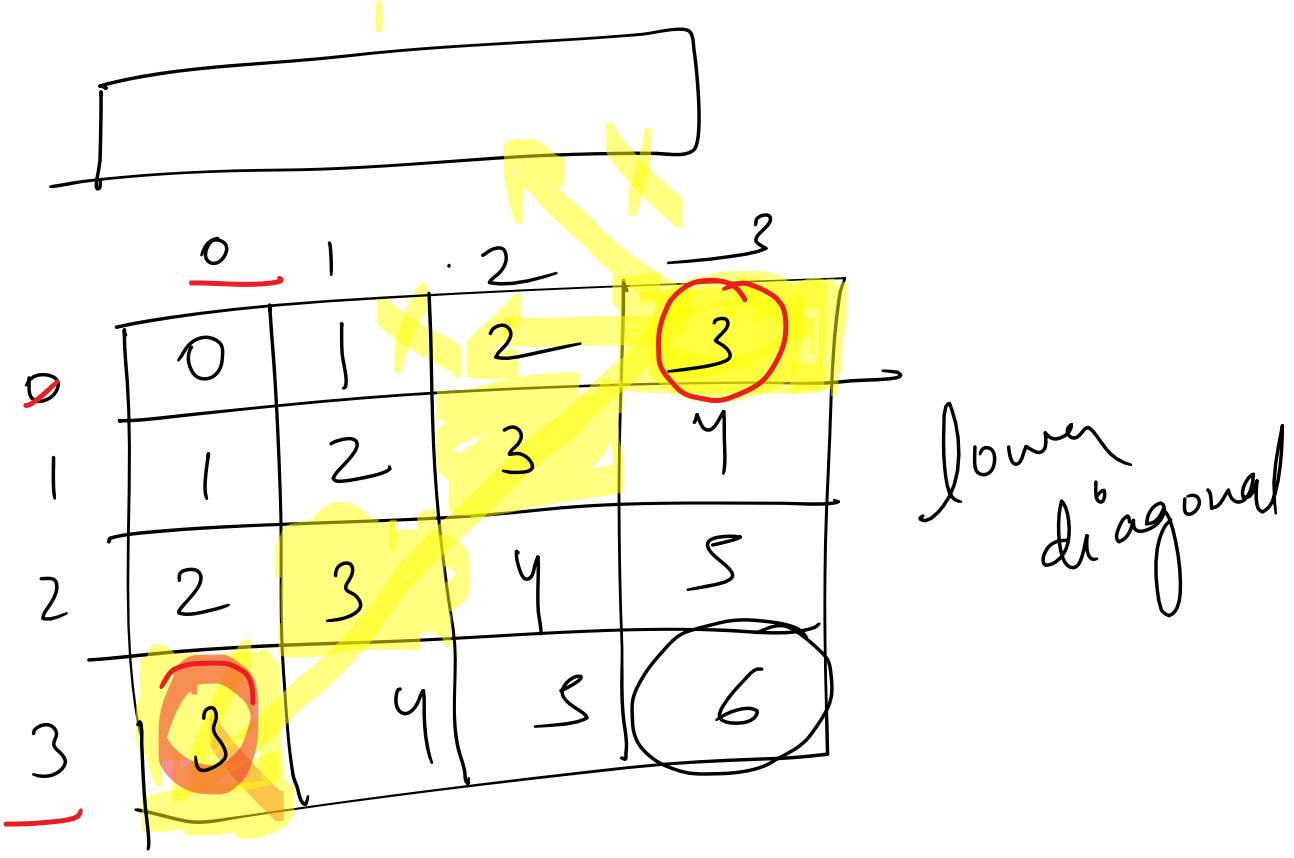

row-1
(row, col)
row-1, col
col-1
row+1


col-1, row+1



$O(N^3)$





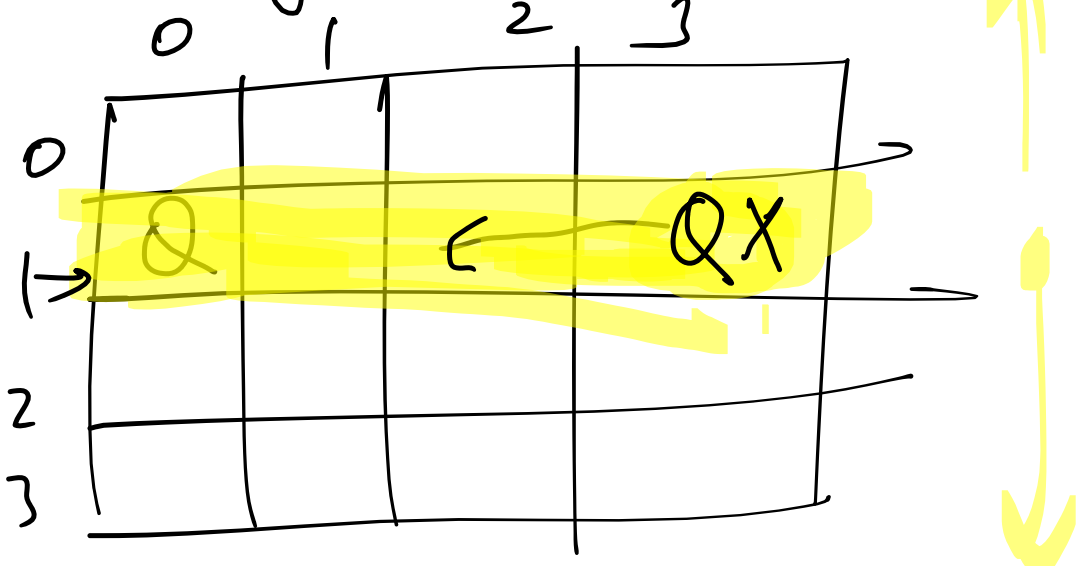
row+col

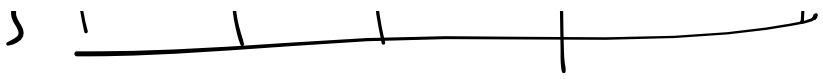
Q

3

row + col

horizontally left





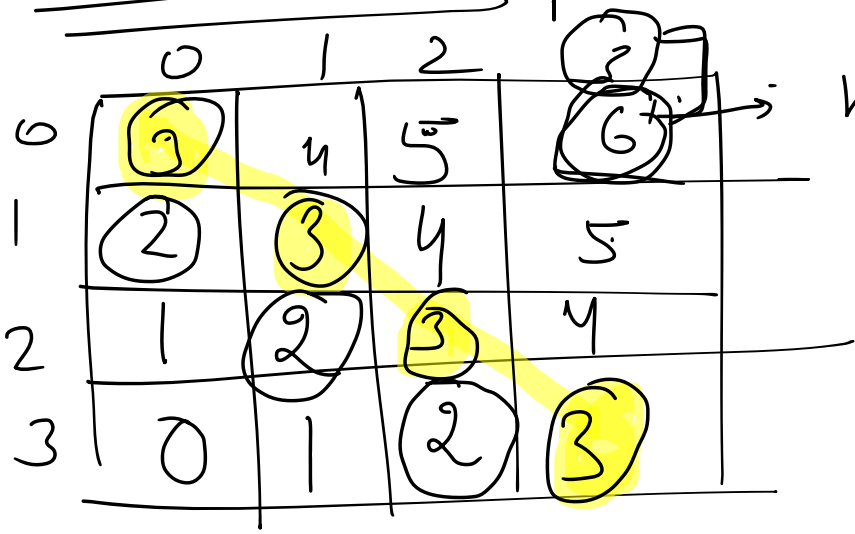
row

1

upper diagonal

$(n-1+col-row)$ $(4-1+3-0)$

$3+3=6$



$n=4$

$2 \times n - 1$

$r=1 \quad c=0$

$n-1+c-r$

$4-1+0-1$

$4-2=2$

$r=2 \quad c=1$

$4-1+1-2$

2

$r=3 \quad c=2$

$4-1+2-3$

$n-1+c-r$

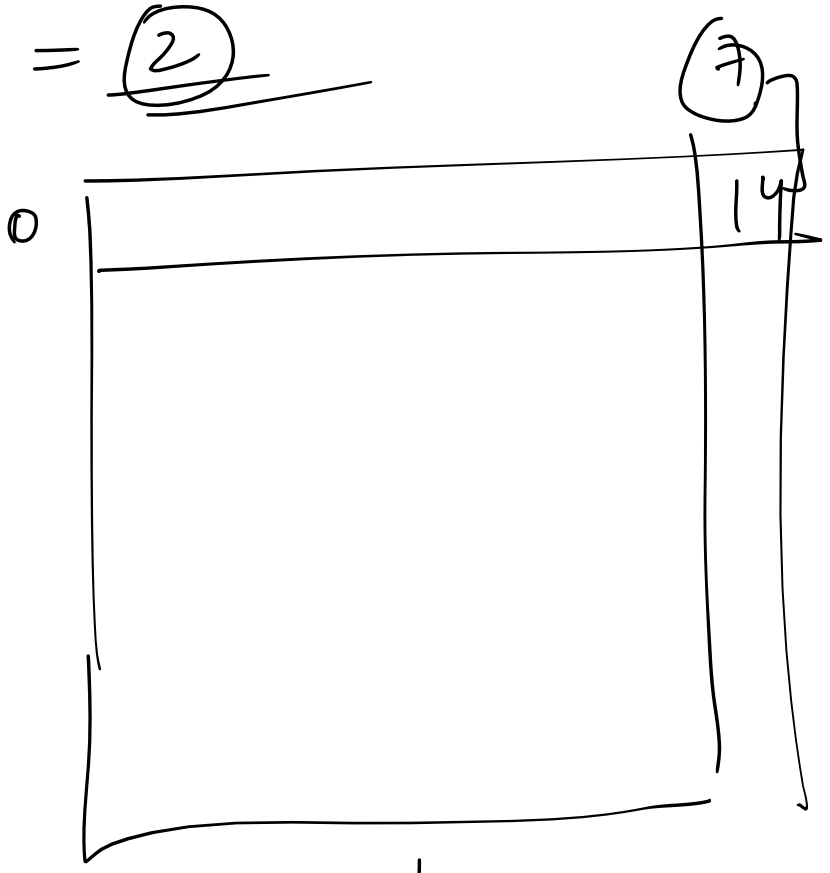
$4-1+2-3$

$= 2$

$n-1+c-r$

$8-1+7-0$

$15-1=14$



$7 = 14$

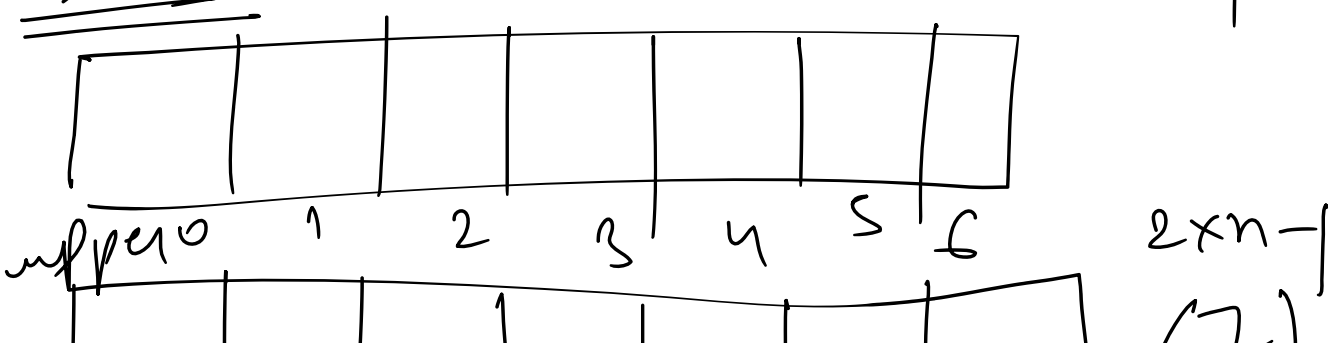
$3 = 6$

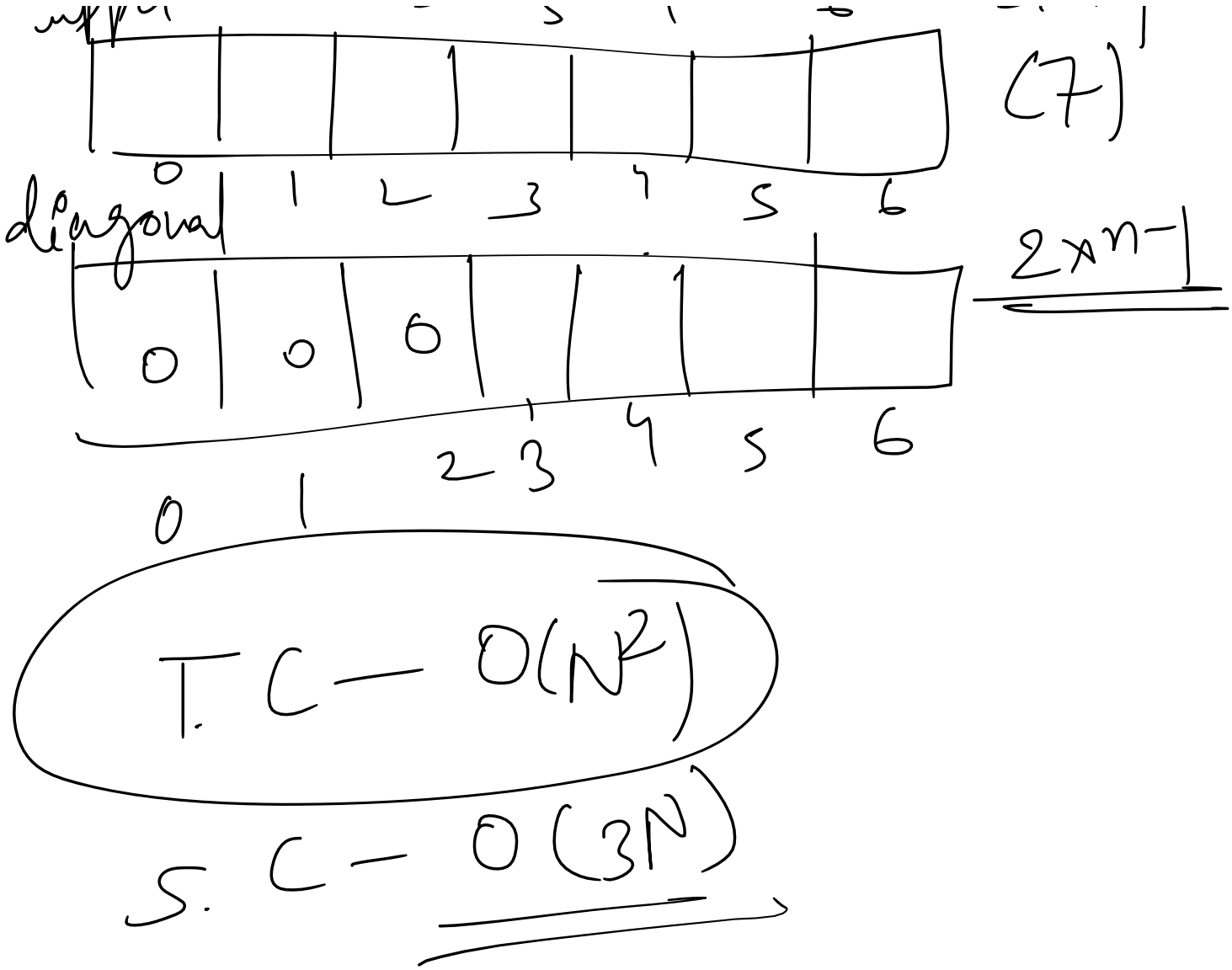
$n-1+col-row$

row

row

$n=4$






```
class Solution {
public:
    bool position_valid(int row,int col,vector<string> &board){
        int temp_row = row;
        int temp_col = col;
        // diagonally upward
        // row  decrese
        // col decrese
        while(row >= 0 && col >= 0){
            if(board[row][col] == 'Q'){
                return false;
            }
            row--;
            col--;
        }
        // horizonally leftward
        // column decrese

        row = temp_row;
        col = temp_col;
        while(col >= 0){
            if(board[row][col] == 'Q'){
                return false;
            }
            col--;
```

```
        }
        // diagonally downwards
        // row increase
        // col decrese
        row = temp_row;
        col = temp_col;
        while(row < board.size() && col >= 0){
            if(board[row][col] == 'Q'){
                return false;
            }
            row++;
            col--;
        }
        return true;
    }
    void solve(int col,vector<string> &board,vector<vector<string>> &answer,int n){
        // base case
        if(col == n){
            answer.push_back(board);
            return;
        }
        for(int row = 0;row < n;row++){
            if(position_valid(row,col,board)){
                board[row][col] = 'Q';
                solve(col+1,board,answer,n);
                board[row][col] = '.';
            }
        }
    }
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> answer;
        vector<string> board;
        string row(n,'.');
        for(int i = 0;i < n;i++){
            board.push_back(row);
        }
               solve(0,board,answer,n);
        return answer;
    }
};
```

T.C - $O(N^3)$

S.C - $O(N)$

```
class Solution {
public:
       void solve(int col,vector<string> &board,vector<vector<string>> &answer,int
n,vector<int> &row_checker, vector<int> & upper_diagonal_checker, vector<int> &
lower_diagonal_checker){
        // base case
        if(col == n){
            answer.push_back(board);
            return;
        }
        for(int row = 0;row < n;row++){
            if(row_checker[row] == 0 and upper_diagonal_checker[n - 1 + col - row] == 0
and lower_diagonal_checker[row + col] == 0){
                board[row][col] = 'Q';
                row_checker[row] = 1;
                upper_diagonal_checker[n - 1 + col - row] = 1;
```

```
                lower_diagonal_checker[row + col] = 1;
                solve(col+
1,board,answer,n,row_checker,upper_diagonal_checker,lower_diagonal_checker);
                row_checker[row] = 0;
                upper_diagonal_checker[n - 1 + col - row] = 0;
                lower_diagonal_checker[row + col] = 0;
                board[row][col] = '.';
            }
        }
    }
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> answer;
        vector<string> board;
        string row(n,'.');
        for(int i = 0;i < n;i++){
            board.push_back(row);
        }
        vector<int> row_checker(n,0);
        vector<int> upper_diagonal_checker(2*n-1,0);
        vector<int> lower_diagonal_checker(2*n-1,0);
        solve(0,board,answer,n,row_checker,upper_diagonal_checker,lower_diagonal_checke
r);
        return answer;
    }
};
```