Recursion Concepts & Qns ...

Motivation (भाषण) ...

"Whenever you feel like quitting, remember

→ Why you started

→ what will success feel like

→ Mom, Dad, your happiness"

video 14

#codestorywithMIK

Facebook
Instagram → codestorywithMIK

(Twitter) → CSwithMIK

codestorywithMIK →

Company :- Google amazon Qualcomm

and many more companies

51. N-Queens

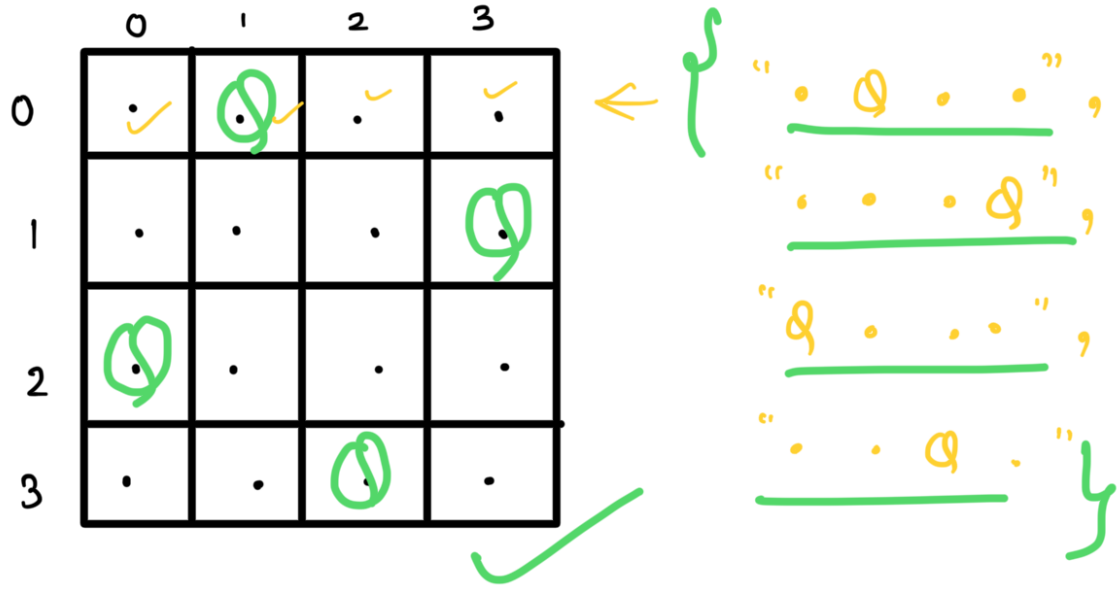
Hard | Topics | Companies

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**.

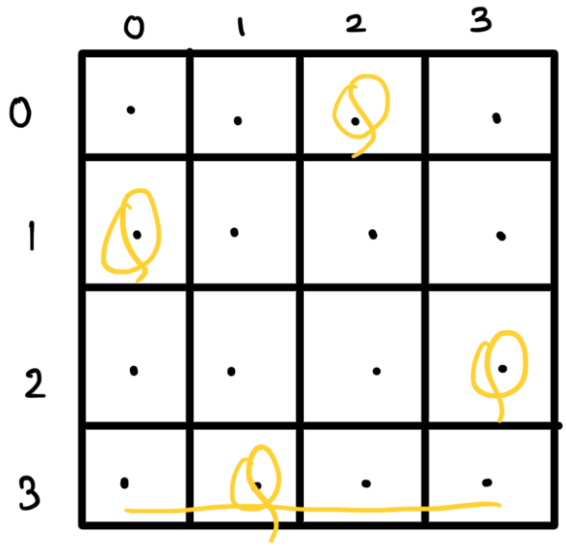
Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example :- n = 4



Thought Process

n = 4



Story Points :-

n = 3

(·) Board →

```
vector<string> board(n, string(n, '.'));
```

n → strings

string(n, '.') → each string

(·) Solve(Board, 0); // row = 0

(·) Place Queen in column & explore.

"Recursion Magic"

```
Solve(Board, row) {
    if (row >= n) {
        result.push_back(Board); return;
    }
    for(col = 0; col < n; col++) {
        if(isValid(Board, row, col)) {
```

Explore ⇒ ⇒ Solve (Board, row+1); //Trust

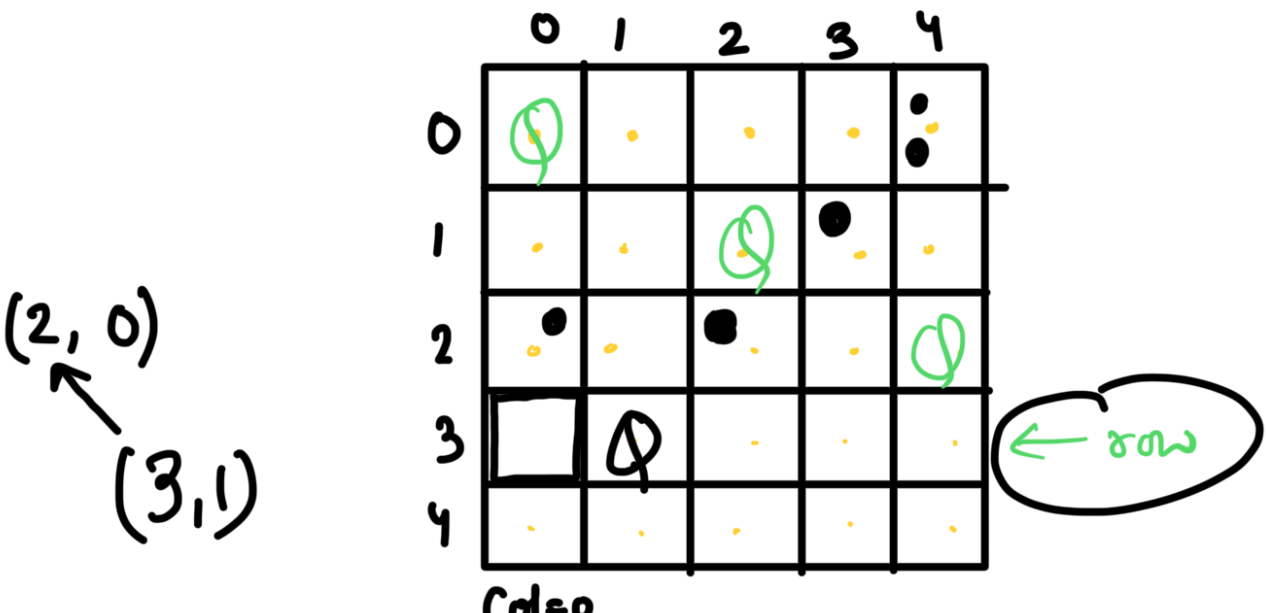
Undo ⇒ ⇒ Board[row][col] = '.' ;

}

}

}

isValid (Board, row, col)



row = 3, col = 0

(3,1)
(2),2)
(1, 3)
(0, 4)

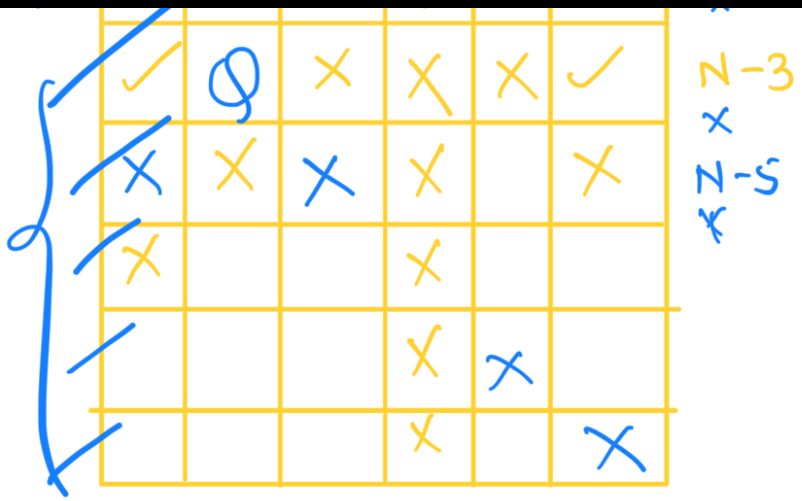
r → [3] [1] ↓ c

- Upward [row-1] [c]
  [row-2] [c]
  ⋮
  [0] [c]
- UpwardDiag. Righ
- UpwardDiag. left

Time Complexity :-

...

```
void solve(vector<string>& board, int row) {
    if(row >= N) {
```




```
            result.push_back(board);
            return;
        }

        for(int col = 0; col < N; col++) {

            if(isValid(board, row, col)) {
                board[row][col] = 'Q';

                solve(board, row+1);

                board[row][col] = '.';
            }

        }
    }
```

$$T(n) = n * (n + T(n-1))$$

$$T(n) = n * T(n-1) + n^2$$

$$= n * [(n-1) * T(n-2) + (n-1)^2] + n^2$$

$$= n*(n-1) * T(n-2) + n*(n-1)^2 + n^2$$

$$= n*(n-1)*(n-2)*\ldots + \text{sum}$$

$$T.C \simeq O(N!)$$

Space Complexity

Space Complexity :-

Extra auxiliary space. = O(1)

System Stack = O(n).

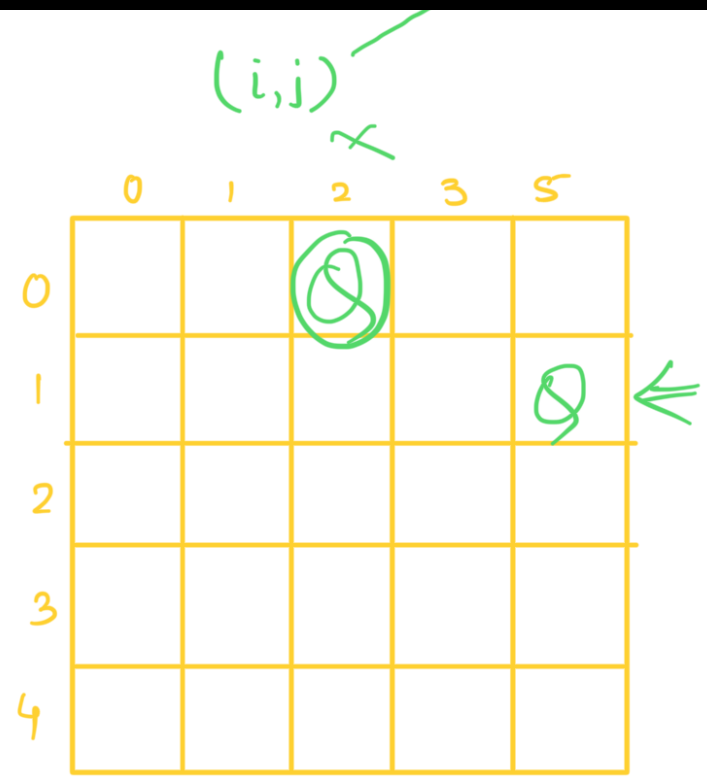
```
void solve(vector<string>& board, int row) {
    if(row >= N) {
        result.push_back(board);
        return;
    }

    for(int col = 0; col < N; col++) {

        if(isValid(board, row, col)) {
            board[row][col] = 'Q';

            solve(board, row+1);

            board[row][col] = '.';
        }

    }
}
```

row = n

Approach - 2

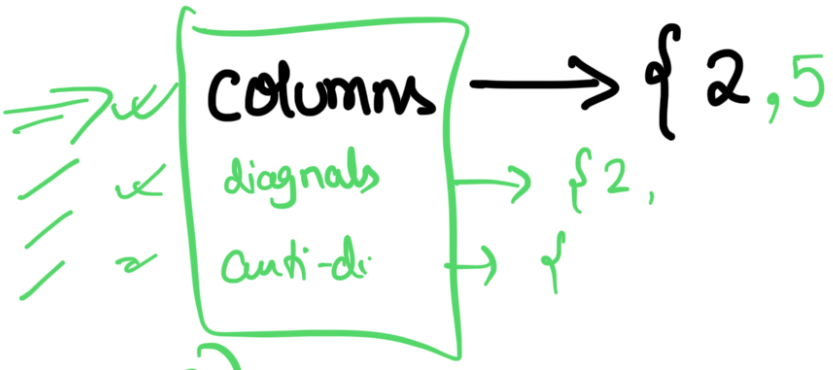
what was the problem in 1st Approach :-

```
void solve(vector<string>& board, int row) {
    if(row >= N) {
        result.push_back(board);
        return;
    }

    for(int col = 0; col < N; col++) {

        if(isValid(board, row, col)) {
            board[row][col] = 'Q';

            solve(board, row+1);

            board[row][col] = '.';
        }

    }
}
```



2-1 = 1

(2,1) = 2+1 = 3
(1,2) = 1+2 = 3
(0,3) = 0+3 = 3

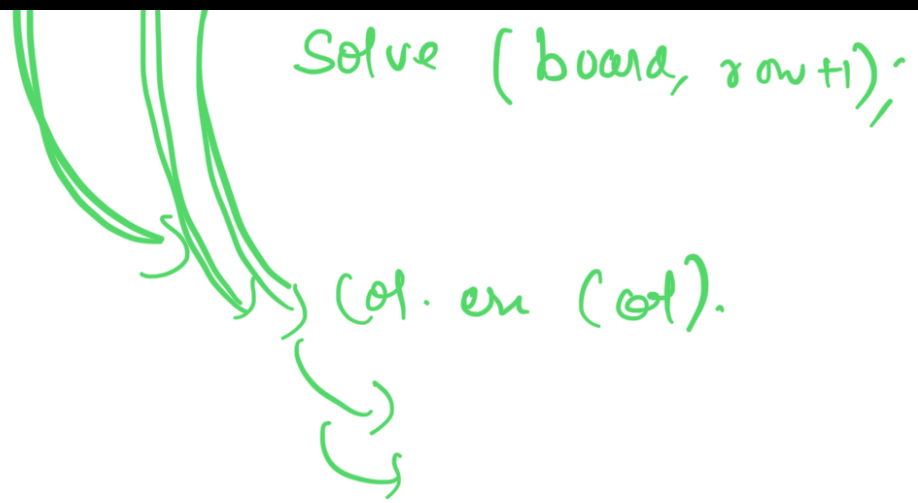


(2,1) = 2-1 = 1
(1,0) = 1-0 = 1

SC:- O(N)

(i-j) Constant → Anti-diag.
(i+j) Const → Diagn.

Columns.insert (col);
diag. inst (row+col); // [row] (col]
antia. inst (row-col);


Solve (board, row+1);
col. en (col).