→ N-Queens

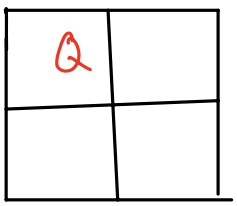
→ Solve SudoKu

N-Queens

Given N. chessboard of dimensions N*N.

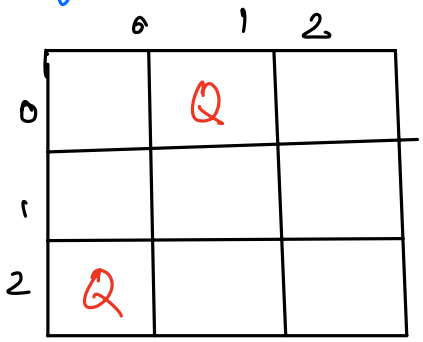
N queens → such that no queen is killing another queen. [print all valid configurations]

N=2.



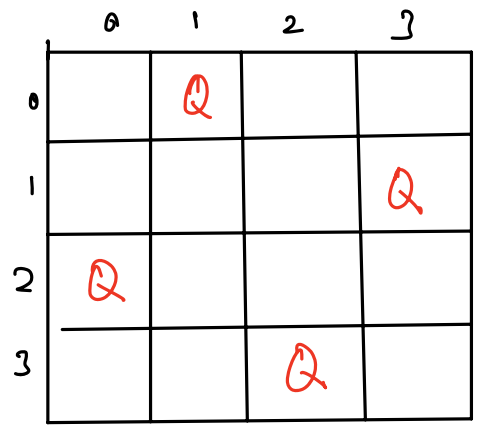
X

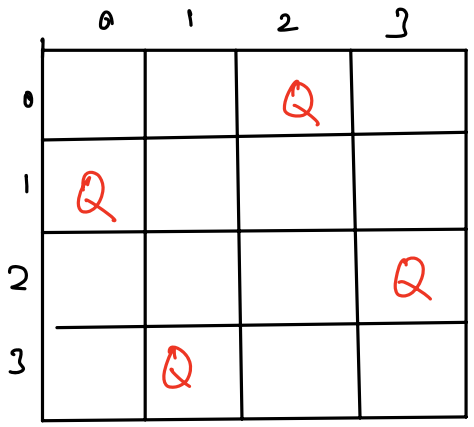
N=3



X

N=4

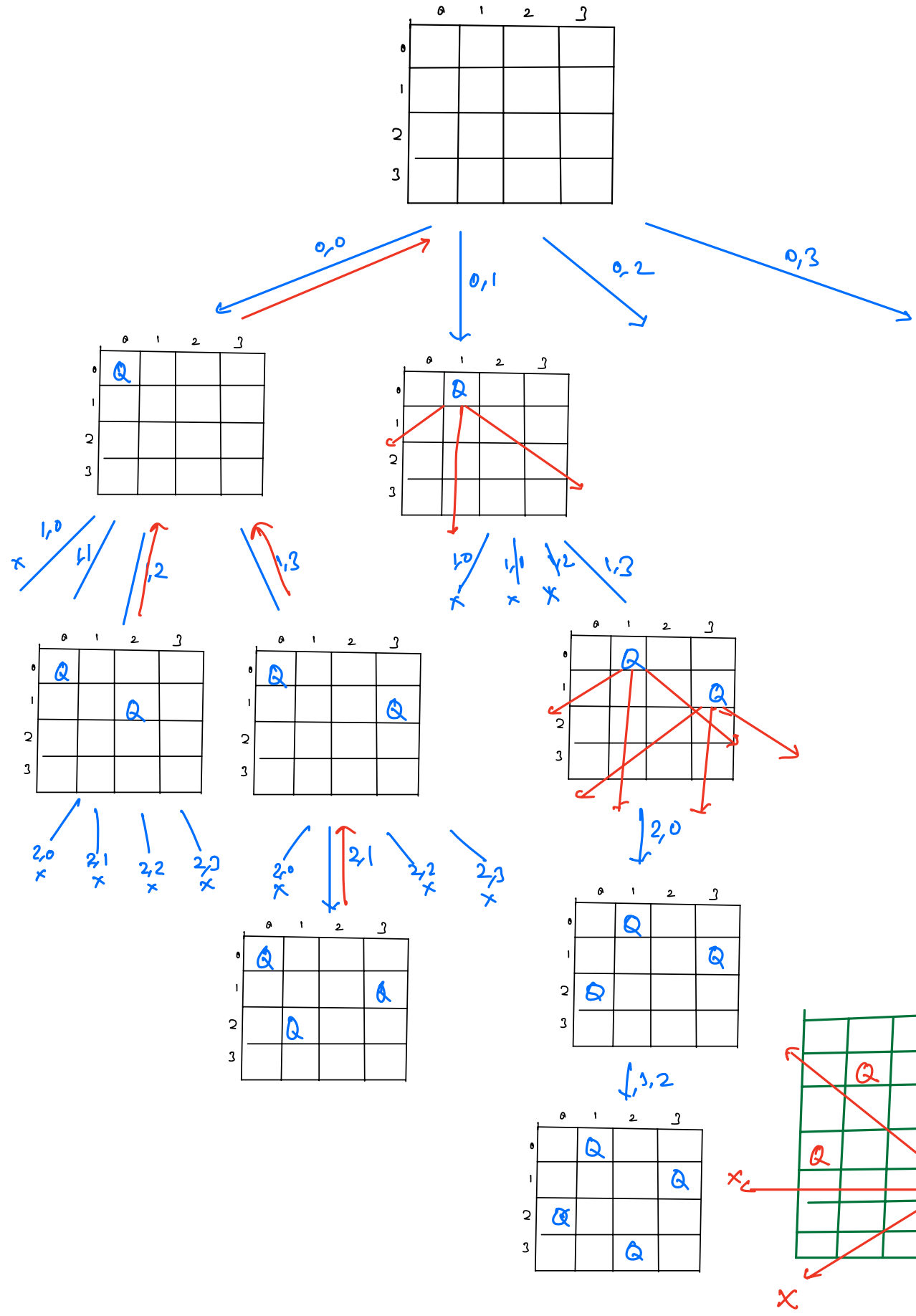




Multiple ans. possible.

Observation → There can be only 1 queen in every row/column.

Place N queens - try to place them row by row
or column by column.

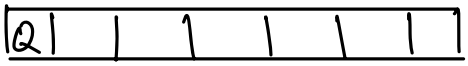


code →

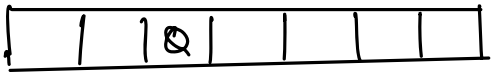
```
                                                    0
                                                    ↑
void nQueens ( mat[N][N] , row , N ){
    if (row == N) { print (mat[][]), return }
    for( int col = 0; col < N; col++){
        if ( isQueenSafe( mat[][], row, col) == true){
            mat[row][col] = 'Q';
            nQueens ( mat[][], row+1, N);
            mat [row] [col] = '.';
        }
    }
}
```

```
boolean isQueenSafe ( mat[][], row, col){          → O(N)
    for( i = 0; i < row; i++) {
        if ( mat[i][col] == 'Q') { return false }
    }

    for( i = row-1, j = col-1 ; i ≥ 0 && j ≥ 0 ; i--, j--){
        if ( mat[i][j] == 'Q') { return false }
    }

    for( i = row-1, j = col+1 ; i ≥ 0 && j < N ; i--, j++){
        if ( mat[i][j] == 'Q') { return false }
    }

    return true;
}
```

0th row → N



1st row N-1



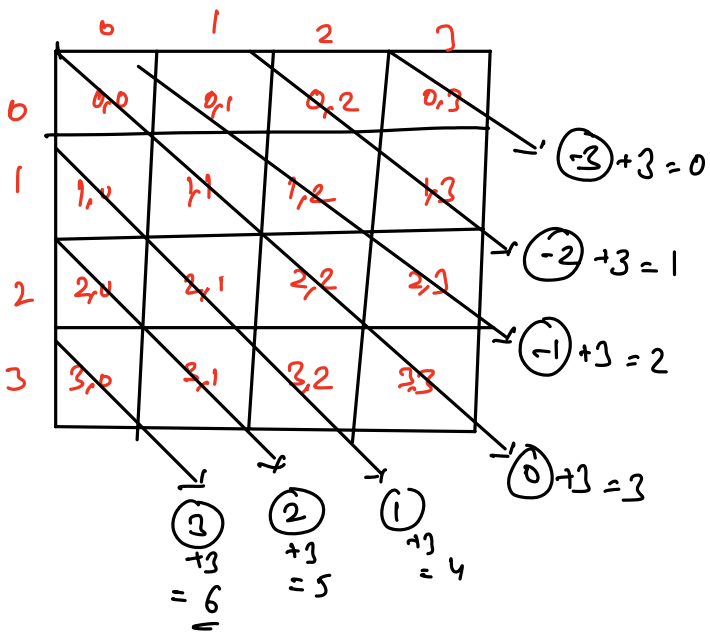
2nd row N-2

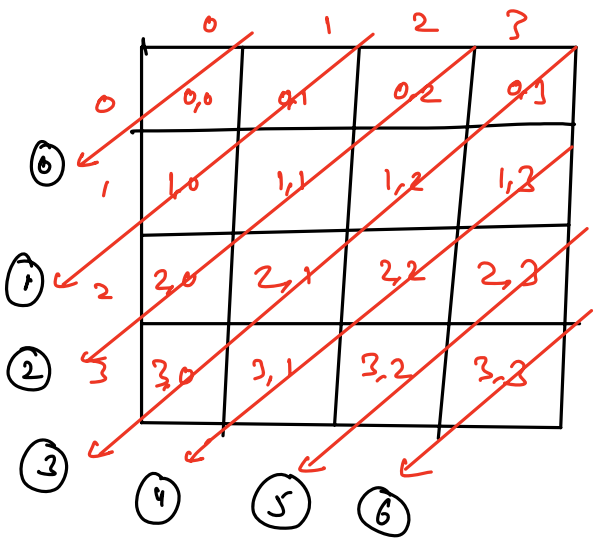
3rd row N-3 →

T.C → O(N*N!)

S.C → O(N²+N) → O(N²)

i-j+N-1.

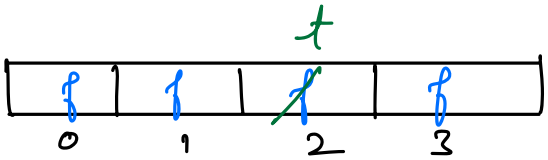




i+j.

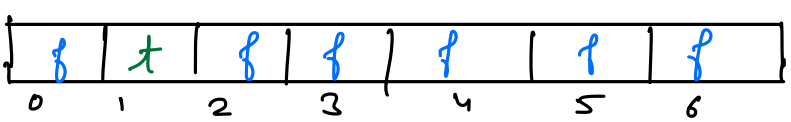
row [0]
0,2.

col[N]

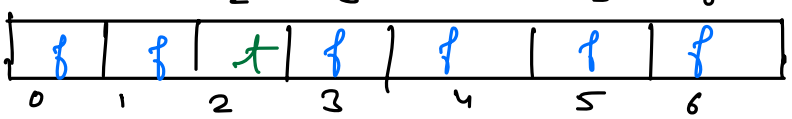


check 2,1.

diag[2N-1] ⇒



anti-diag[2N-1] ⇒



code →

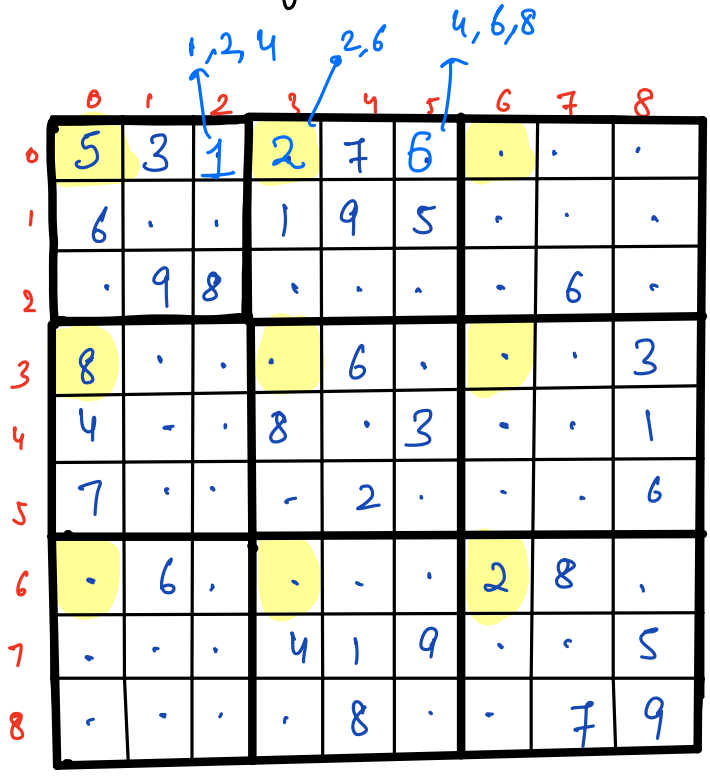
```
void nQueens( mat[N][N], i, N, cols[N], diag[2N-1], anti-diag[2N-1]){
    if( i == N) { print mat[N][N], return }
    for( j = 0 ; j < N; j++){
        if ( cols[j] == false && anti-diag[i+j] == false
            && diag[i-j+N-1] == false) {
            mat[i][j] = 'Q';
            cols[j] = true;
            anti-diag[i+j] = true;
            diag[i-j+N-1] = true;
            nQueens( mat[][], i+1, N, cols[], diag[], anti-diag[]);
            mat[i][j] = '.';
            cols[j] = false ;
            anti-diag[i+j] = false;
            diag[i-j+N-1] = false;
        }
    }
}
```

T.C → $O(N!)$

S.C → $O(N^2)$

Sudoku

Given a partially solved state of sudoku. Find the solution of sudoku. [1 unique solution exists]



N = 9.

Rules for sudoku -

- Each row must contain all the numbers from 1 to N.
- Each column must contain all the numbers from 1 to N
- Each $\sqrt{N} \cdot \sqrt{N}$ grid must contain all the numbers from 1 to N.

dimensions of every grid → $\sqrt{N} \cdot \sqrt{N}$

4 → closest multiple of 3 which is ≤ 4 ⇒ 3
8 → closest multiple of 3 which is ≤ 8 ⇒ 6

i → Closest multiple of $\sqrt{N}$ which is $\leq$ row ⇒ row − (row % $\sqrt{N}$)
j → Closest multiple of $\sqrt{N}$ which is $\leq$ col ⇒ col − (col % $\sqrt{N}$)

code :-

```
                                    0   0
                                    ↑   ↑
boolean sudoku (mat[N][N], i, j, N) {

    if ( j == N){
        i = i+1,   j = 0;
    }

    if ( i == N){
        return true;
    }

    if ( mat[i][j] != '.'){
        if ( sudoku(mat[][], i, j+1, N) == true){
            return true;
        }
    }
    else{
        for( x = 1;  x <= 9;  x++){
            if ( isValid (mat[N][N], i, j, x) == true){
                mat[i][j] = x;
                if ( sudoku(mat[][], i, j+1, N) == true){
                    return true;
                }

                mat[i][j] = '.';
            }
        }
    }

    return false;
}
```

T.C → $O(N^{\text{no. of empty spots}})$

S.C → $O(N^2)$

```
boolean isValid( mat[N][N], row, col, x ) {        → O(N)
    for( j = 0; j < N; j++){
        if ( mat[row][j] == x) { return false }
    }

    for( i = 0; i < N; i++){
        if (mat[i][col] == x) { return false }
    }

    row = row - (row % √N)
    col = col - (col % √N)        ] indices of top-left corner.

    for( i = 0; i < √N; i++) {
        for( j = 0; j < √N; j++) {
            if( mat[i+row][j+col] == x){
                return false;
            }
        }
    }

    return true;
}
```

→ D.P

→ Graphs