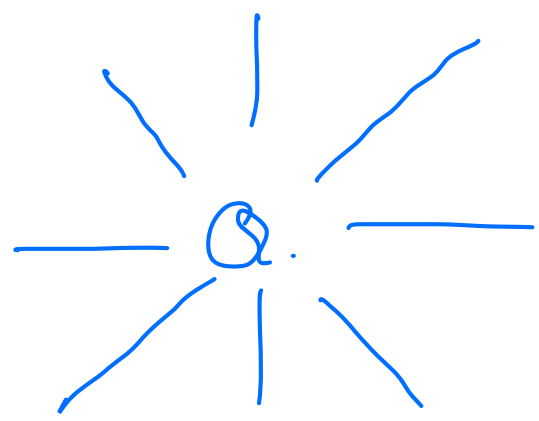
N Queens

Given N. Chessboard of dimension N*N.
N queens → such that no queen is killing another queen.
[print all valid configuration]

N = 2.

Q.	x
x	x

N = 3.



Q.		
		Q.

N = 4.

	Q.		
			Q.
Q			
		Q.	

Multiple ans.

		Q.	
Q			
			Q
	Q.		

Obs:- There will be exactly 1 queen in each row & each column.

try to place each queen row by row.

row = 0

	0	1	2	3
0				
1				
2				
3				

0,0 0,1 0,2 0,3

	0	1	2	3
0	Q			
1	✗	✗		
2				
3				

	0	1	2	3
0		Q		
1				
2				
3				

row = 1

1,2 1,3 1,3

	0	1	2	3
0	Q			
1	✗	✗	Q	
2	✗	✗	✗	✗
3				

	0	1	2	3
0	Q			
1	✗	✗	✗	Q
2	✗		✗	✗
3				

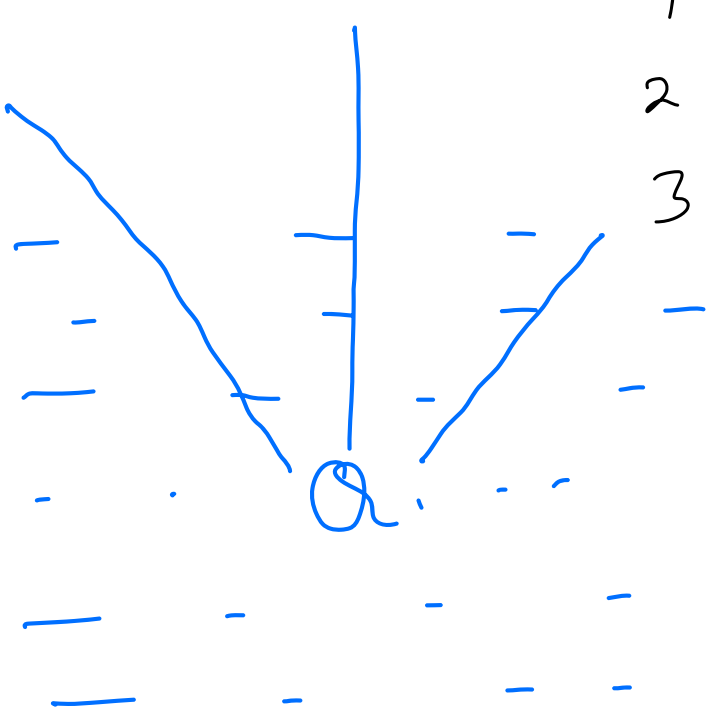
	0	1	2	3
0		Q		
1	✗	✗	✗	Q
2		✗	✗	✗
3				

row = 2

2,1 2,0

	0	1	2	3
0	Q			
1	✗	✗	✗	Q
2	✗	Q	✗	✗
3	✗	✗	✗	✗

	0	1	2	3
0		Q		
1	✗	✗	✗	Q
2	Q	✗	✗	✗
3				



row = 3

3,2

	0	1	2	3
0		Q		
1	✗	✗	✗	Q
2	Q	✗	✗	✗
3			Q	

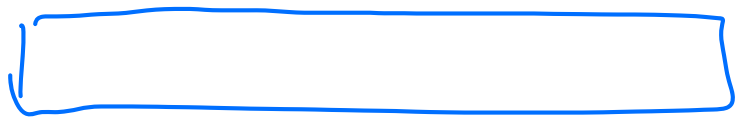
row = 4.

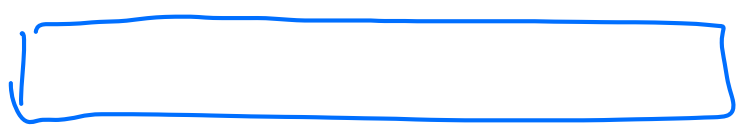
Code-

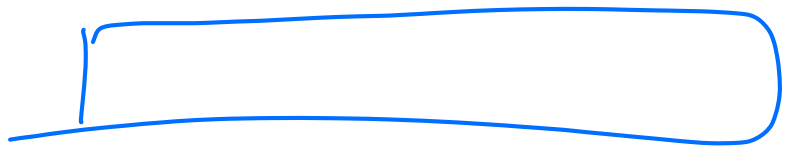
```
void nQueens (mat[N][N], row) {
if (row == N) { print matrix; return; }.
                    # trying out all possibilities
  for (col = 0; col < N; col ++) {
                    //valid possibilities
    if ( isQueenSafe (mat[][], row, col)
                    == true) {
      mat[row][col] = 'Q'; //place Queen
      nQueens (mat, row+1); // make recursive call
      mat[row][col] = '.'; //unplace Queen
    }.
  }
}
```

```
boolean isQueenSafe (mat[N][N], row, col){
  # Check for row
                    X Not needed.

  # Check for col.
  for (i = 0; i < row; i++) {
      if (mat[i][col] == 'Q') {return false;}

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

  # Check for anti digonal.
```

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

return true;
}.
```

N. 

N-2. 

N-3 

$N * (N-2) * (N-3) * \ldots (1)$

$O(N!) \times N$

$= O(N!)$ → To be revisited.

S.C. → $O(N^2)$ $\simeq O(N^2 + N)$.

8:35

SUDOKU

Given a partially solved state of sudoku. Find the solution of sudoku. [Unique Solution exists]

1, 2, 4

	0	1	2	3	4	5	6	7	8
0	5	3	1	2	7	6	.	.	.
1	6	.	.	1	9	5	.	.	.
2	.	9	8	.	.	.	.	6	.
3	8	.	.	.	6	.	.	.	3
4	4	.	.	8	.	3	.	.	1
5	7	.	.	.	2	.	.	.	6
6	.	6	.	.	.	.	2	8	.
7	.	.	.	4	1	9	.	.	5
8	.	.	.	.	8	.	.	7	9

1, 2, 4, 6, 8, 9.

Rules:

1. Every Row: 1 to N
2. Every Column: 1 to N
3. Every Grid: 1 to N.

Assume N to be a perfect square.

row → $i - (i \% \sqrt{N})$; $4 - (4 \% 3) = 3$

col → $j - (j \% \sqrt{N})$; $8 - (8 \% 3)$

$1 - (1 \% 3) = 0$

$8 - (8 \% 3) = 6$

Code.

```
bool sudoku (mat[N][N], i, j, N)
    if (j == N) { i++; j = 0; }
    if (i == N) { return true; }

    if (mat[i][j] != '.') {
        if (sudoku (mat, i, j+1, N) == true))
            { return true; }
```

```
}
else {
    # trying out all possibilities
    for (x = 1; x <= N; x++) {
        // consider only valid poss..
        if (isValid(mat, i, j, x)
            == true) {
            # do-change
            mat[i][j] = x;
            #recursive call
            if (sudoku(mat; i, j+1, N) == true))
                {return true;}.
            mat[i][j] = '.'.
            # Undo-change
        }
    }
    return false;
}
```

```
boolean isValid(mat[][], row, col, x) {
```

① Check for row;

TODO

② Check for column;

TODO

③ Check for grid.

```
rowG = row - (row % sqrt(N));
colG = col - (col % sqrt(N));

for (i = rowG ; i < rowG + √N ; i++)
    for (j = colG ; j < colG + √N , j++){
        if (mat[i][j] == x).
            {return false;}
    }
}

return true;
}
```

Z empty cells.

N^Z possibilities.

$= (N)^{N^2}$.

$= SC(O(N^2)) \simeq O(\text{Empty cells})$