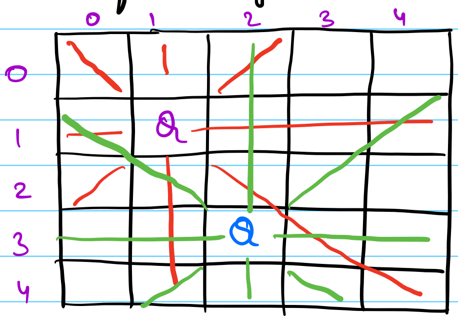
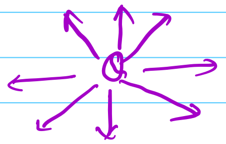
25/8/23

Backtracking -2

Q Given N×N chess board & location of 2 queens. Check if they can attack each other.





i/p ⇒ (1, 1) , (3,2) → false

i/p ⇒ (1,1), (3, 1) → true

(r1, c1) → Q1
(r2, c2) → Q2

Directions

1> ↔ Same row (r1 == r2)

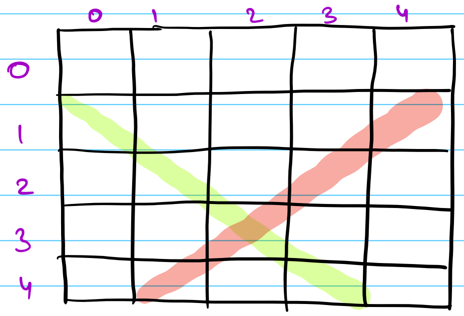
2> ↕ same col (c1 == c2)

3> ↖↘

$(r1-r2) == (c1-c2)$

4> ↗↙

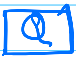
$(r1+c1) == (r2+c2)$

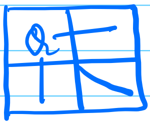


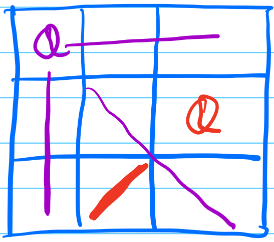
(1,0)	(1,4)
(2,1)	(2,3)
(3,2)	(3, 2)
(4,3)	(4,1)

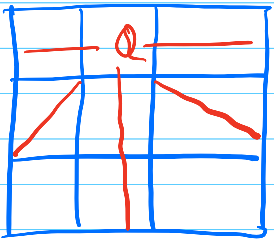
Q: Given an integer N, check if it is possible to place N queens on a N*N chessboard s.t no queen attacks each other.

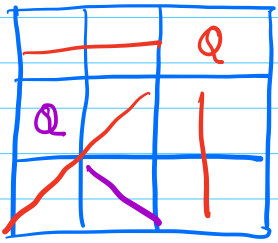
N	Ans
1	true
2	false
3	false
4	true

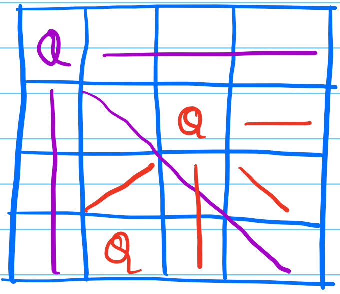


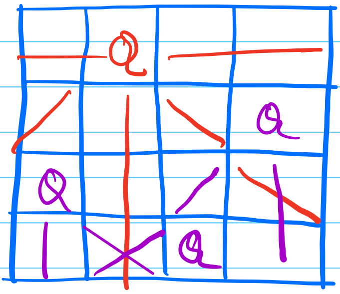










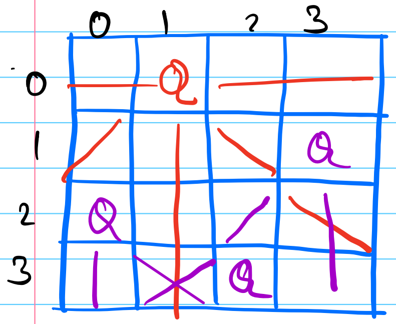


N Queens & N*N chessboard.

1) Every row should have exactly 1 queen.
2) Every col should have exactly 1 queen.

1) Place the queens row by row.
2) Place the queens col by col.

Do we really need N*N extra space to keep track of placed queens?



col

0	1	2	3
1	3	0	2

(i, col[i]) → loc^n of i^th queen.

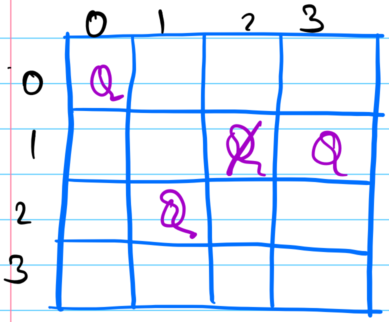
row

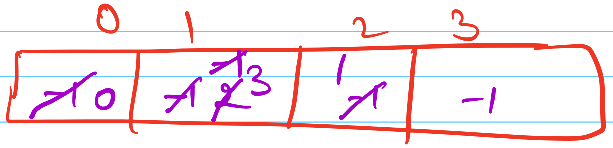
0	1	2	3
2	0	3	1

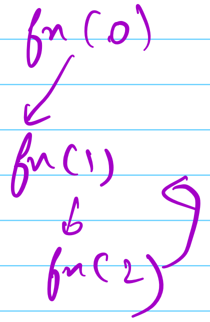
SC: O(N)

```
              0
boolean nqueen(r, col[]) {      i/p → N

    if (r==N) return true;   //Base case.

    for c → 0 to (N-1) {   // All possibilities
        if (isValid(col[], r, c)) {  //Valid possibility

            col[r] = c    //Do

            if (nqueen(r+1, col)) {  //Recursion
                return true;
            }

            col[r] = -1   //Undo (optional)
        }

    return false;

}
```

```
boolean isValid (col[], r, c) {
    for i = 0 to (r-1) {
        j = col[i]      (i, j)

        if (j == c || (i-r) == (j-c) || (i+j) == (r+c)) {
                return false;
        }
    }
    return true;
}
```

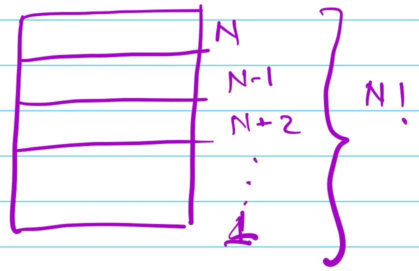






TC ≤ O(N! * N)

SC: O(N + N) ≈ O(N)

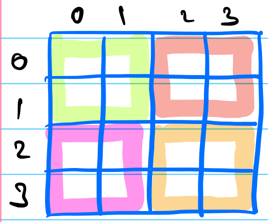


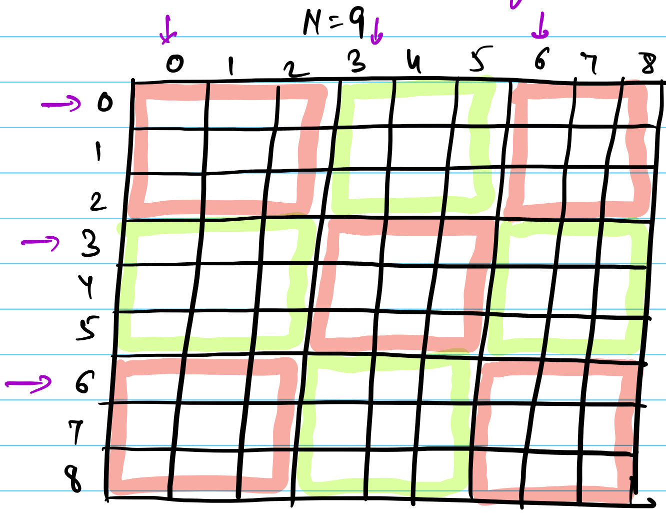
Meet at 8:40 am IST

Q Solve the given incomplete sudoku.

Sudoku is a N*N grid where N is a perfect square & every row, column or block has unique elements.

N=4





	0	1	2	3
0	2	4	1	3
1	1	3	2	4
2	3	2	4	1
3	4	1	3	2

✓

i/p ⇒

	0	1	2	3
0	2	4	1	3
1	1	3	2	4
2	3	2	4	1
3	4	1	3	2

0 1 2 3 4 5 6 7 8

(7, 3) (6 3)

(8, 7) (6, 6)

$\left(\frac{x}{3}\right)*3$

$x-(x\%3)$

```
boolean sudoku (A[][], N, r, c) {
    if (c==N){
        r+=1; c=0;
    }
    if (r==N){ return true;} // Base case.

    if (A[r][c] >0){ // Already filled.
        return sudoku(A, N, r, c+1);
    }

    for i-> 1 to N { // All possibilities.
        A[r][c] = i  // Do

        if(check(A, N, r, c) && sudoku(A, N, r, c+1)){
            return true;
        }

        A[r][c]=0;
    }

    return false.
}
```

Valid possibility of recursion.

N×N×N . . . N² times

$TC < O(N^{N^2})$

$SC: O(N^2)$

```
boolean check( A[][], N, r,c) {
    for i-> 0 to(N-1) {
        if ( i != c && A[r][c] == A[r][i])  ⟷
            return false;

        if( i != r && A[i][c] == A[r][c])  ↕
            return false.
    }
    sq = sqrt( N)

    u = r - r % sq
    v = c - c % sq
    for i-> 0 to (sq-1) {
        for j-> 0 to (sq-1) {

            x = u+i
            y = v+j

            if((x != r || y != c) && A[x][y] == A[r][c]) {
                    return false;
            }
        }
    }
    return true;
}
```

i/p ->

	0	1	2	3
0	2	4	1	3
1	1	3	2	4
2	3	2	4	1
3	4	1	3	2

	1	2	3	4
1	0	~~0~~	0	~~0~~
2		1		1
3				
4				

N*4

N*4

N*4

N*4

```
                                             ↑0
void permutation (A[], vst[], ans[], ind){ N→A.length
    if (ind == N){                  // Base case
        printArray(ans);
        return;
    }

    for i→0 to (N-1){  // All possibilities.
        if (!vst[i]){          //Valid possibility
            vst[i] = true       //do
            ans[ind] = A[i]

            permutate(A, vst, ans, ind+1) //Recursion.

            vst[i] = false;   //Undo
        }
    }
}
```

$$\begin{array}{lcccc} & 0 & 1 & 2 & \\ [a,b,c] \quad \text{ans}[& b & a & c &] \\ \text{vst}[& T & F & F &] \end{array}$$

ind=0