Today's content

(i) N-Queens placing in chess board

(ii) Sudoku solver.

Announcement

→ Solving these in interview ⇒ Pen tablet
⇓
(3K-5K)

→ contest - [After trees]

→ next syllabus. ↘ till backtracking.

[DP + Graphs]

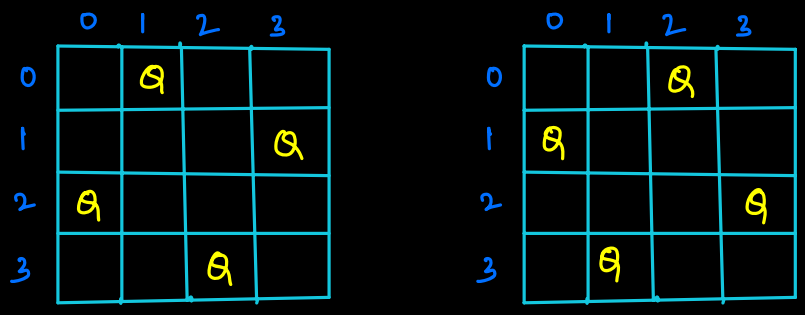
Q.3: **N-Queens.**

Given N×N matrix, print all valid placement of N Queens such that no queen can kill each other queen.

↓

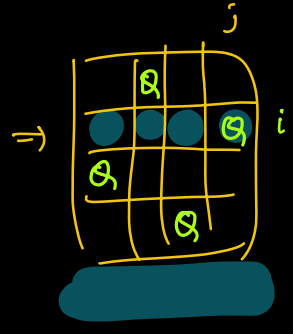
Queens in same row, column, diagonal can kill each other.

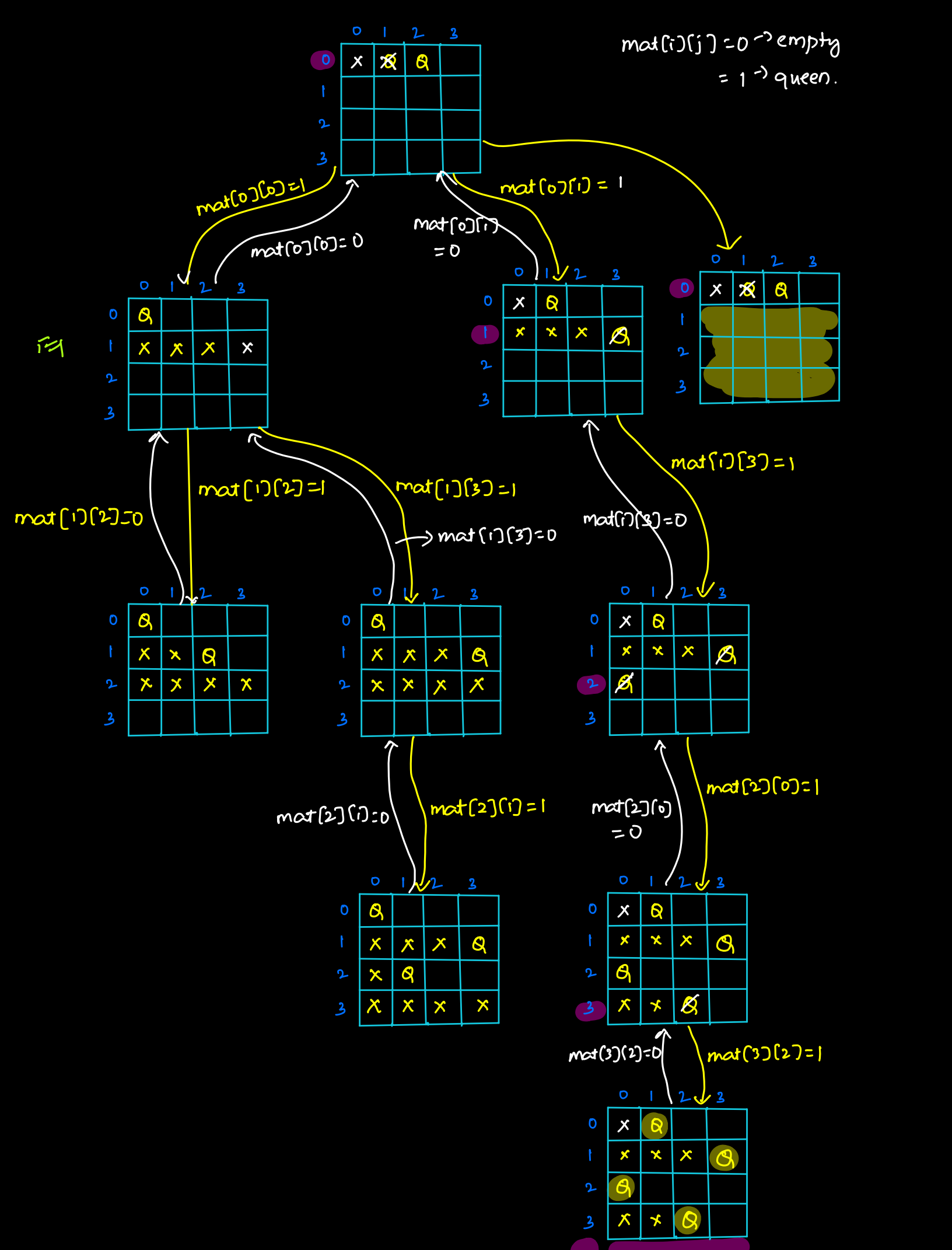
Ex: N = 4.



Observation: we must place 1 queen in every row.

```
void nQueens(int[][] mat, int i, int N)
    if(i==N)
        print(mat)
        return
    // at ith row, i'll try placing queen
    for(j=0; j<N; j++)
        // is it safe to place the queen.
        if(isQueenSafe(mat, i, j, N))
            mat[i][j]=1 // placed a queen
            nQueens(mat, i+1, N) // place for next row.
            mat[i][j]=0 // undo queens placement
```




mat[i][j] = 0 -> empty
= 1 -> queen.
mat[0][0]=1
mat[0][0]=0
mat[0][1] = 1
mat[0][1] = 0
i==1
mat[1][2]=1
mat[1][2]=0
mat[1][3]=1
mat[1][3]=0
mat[1][3]=1
mat[1][3]=0
mat[2][1]=0
mat[2][1]=1
mat[2][0] = 0
mat[2][0]=1
mat[3][2]=0
mat[3][2]=1

print(mat[][])

```
boolean isQueenSafe(mat, row, col, N)
    // can queen get killed from the top.
    for(i=0; i<row; i++)
        if(mat[i][col]==1)
            return false

    // can queen get killed from the left dia.
    for(i=row-1, j=col-1; i>=0 && j>=0; i--, j--)
        if(mat[i][j]==1)
            return false

    // can queen get killed from the right dia.
    for(i=row-1, j=col+1; i>=0 && j<N; i--, j++)
        if(mat[i][j]==1)
            return false

    return true
```

mat[][] recursion.

SC: $O(N^2+N)$

TC: $O(N*N!)$

Validation

fun calls.

Actual

		0	1	2	3
N	0	Q			
N-2	1	–	–	Q	
N-4	2				
N-6	3				

Approximations

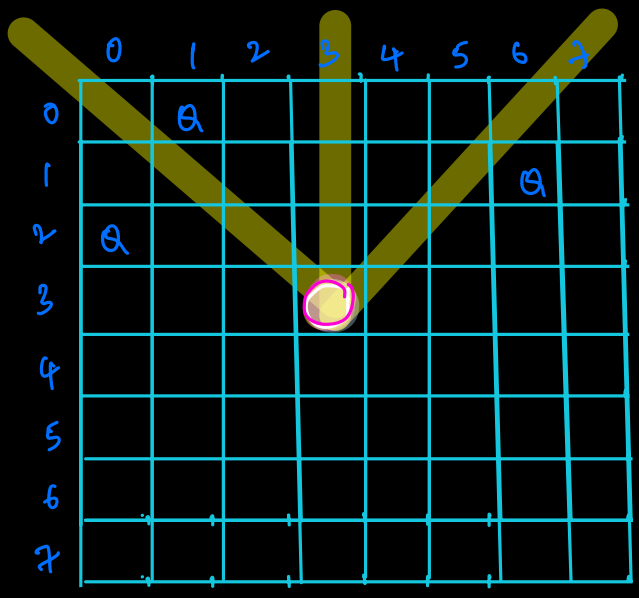
N

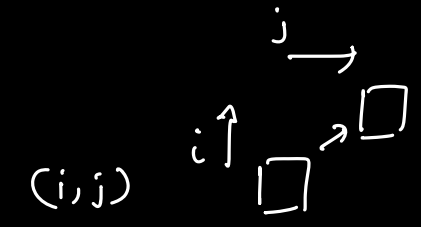
N-1

N-2

N-3

1





$i \to i-1$, $j \to j+1$

```
for(i=row-1,j=col+1; i>=0 && j<N; i--,j++)
```

Optimizing isQueenSafe [optional]

idea: As soon as you place a queen, try marking the cells where queen (future) cannot be placed.

col: top — | f | f | f | — — — | f | → (N)

0 1 2 3 — — —

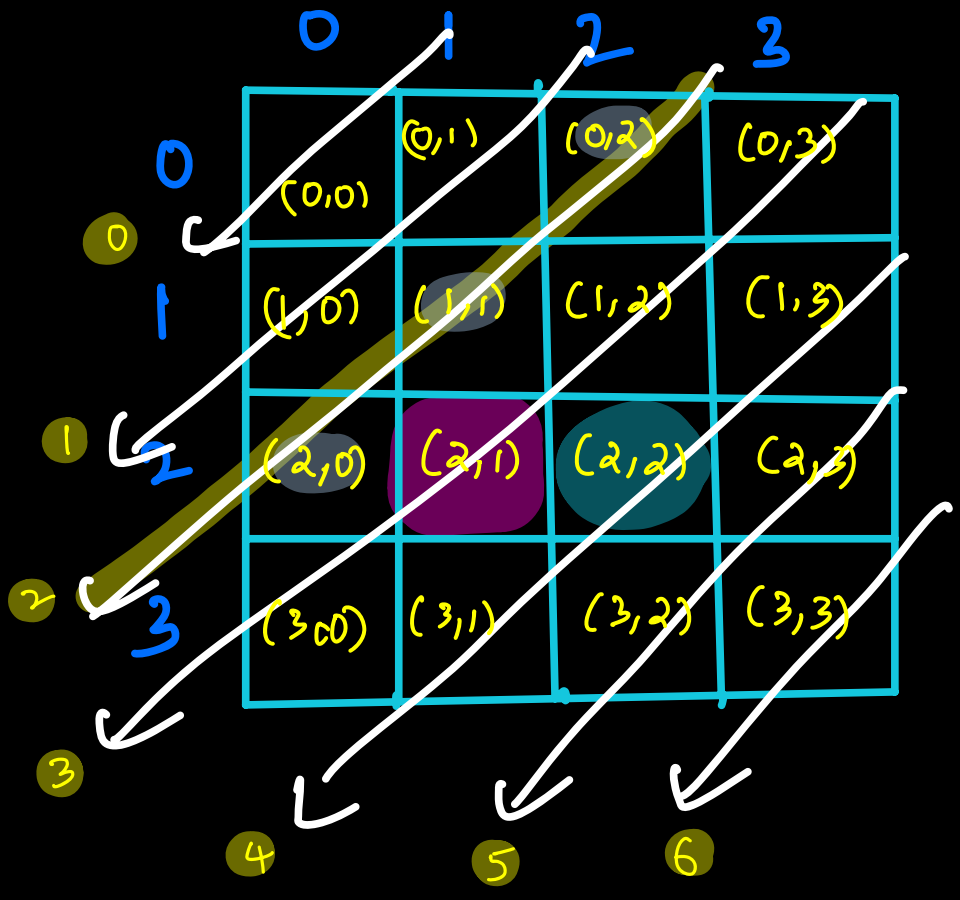
anti-dia: left — | f | f | f | — — — | f | → (2N-1)

0 1 2 3 — — —

dia: right — | f | f | f | — — — | f | → (2N-1)

0 1 2 3 — — —

Anti-diagonal.



N - upper

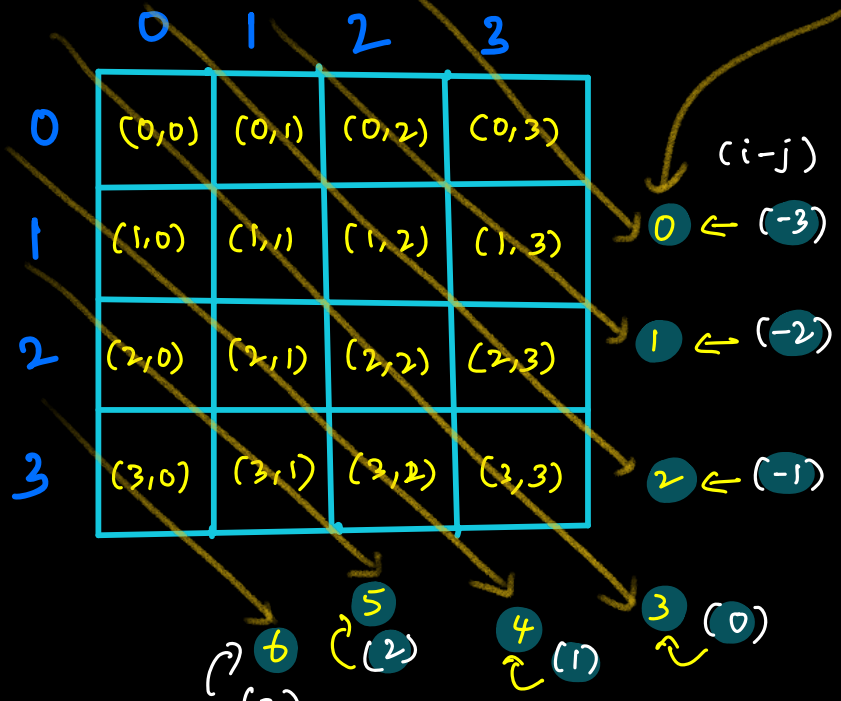
(N-1) - lower

Total = 2N-1

diagonal number = (i+j)

Diagonal

diagonal number = (i-j) +(N-1)

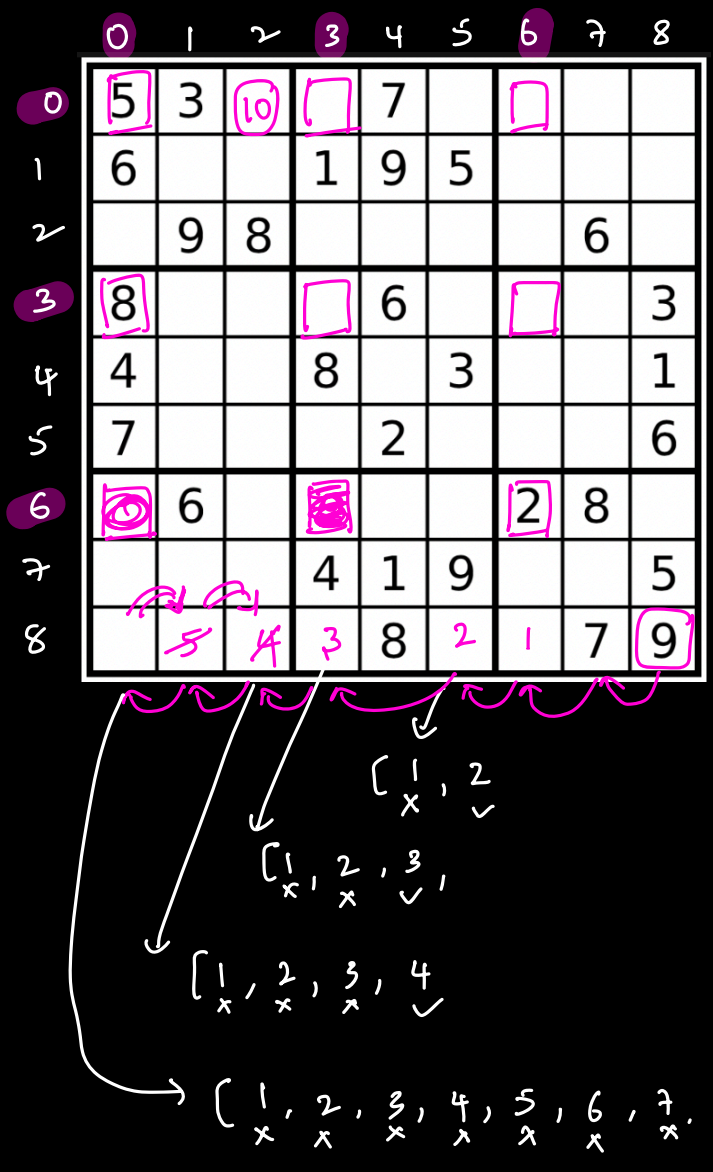


```
Void  nQueens( int[][] mat, int i, int N, top[], left[], right[] )
    if(i==N)
        print(mat)
        return
    // at ith row, I'll try placing queen
    for( j=0; j<N; j++)
        // is it safe to place the queen.
        if( top[j]!=1 && left[i+j]!=1 && right[i-j+N-1]!=1)
            mat[i][j]=1 // placed a queen
            top[j] = 1
            left[i+j] = 1
            right[i-j+N-1] =1   // mark unplaceable positions
            nQueens(mat, i+1, N) // place for next row.
            mat[i][j]=0 // undo queens placement
            top[j] = 0
            left[i+j] =0
            right[i-j+N-1] =0   // undo mark unplaceable positions
```

TC: $O(N!)$

SC: $O(N^2+N)$.

2. Sudoku solver.



(i) every cell should contain nos from [1-9]

(ii) every row, col all the nos from [1-9] should be present

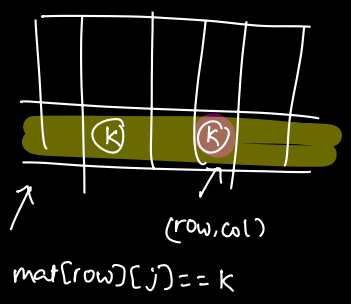
(iii) in any given box, all the nos from [1-9] should be present

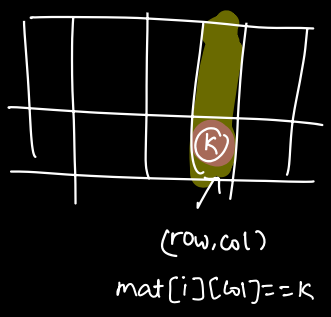
```
class Cell
    int row
    int col
    Cell(i,j)
    ) row: i, col: j
```

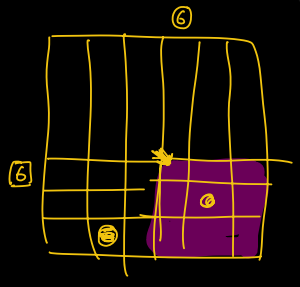
```
main()
    Cell cell = findNextEmptyCell(mat, 8,8)
    if (cell.row == -1 && cell.col == -1)
        return

    fillSudoku(mat, cell.row, cell.col)
```

```
boolean fillSudoku(int[][] mat, int i, int j)
    if (i==-1 && j==-1)
      return true

    for(k=1; k<=9; k++)
        if (isValid( mat, i,j, k))
            mat[i][j] = k
            Cell cell = findNextEmptyCell(mat, i, j)
            if(fillSudoku(mat, cell.row, cell.col)==true)
               return true

            mat[i][j] = 0

    return false


Cell findNextEmptyCell (int[][] mat, int i, int j)
    while ( mat[i][j] != 0)
            j=j-1
            if(j<0)
               j=8
               i=i-1

            if(i<0)
               return new Cell(-1,-1)

     return new Cell(i,j)
```

```
boolean isValid(int[][] mat, int row, int col, int k)
    // row validator.
    for(i=0; i<9; i++)
        if(mat[row][i]==k)
            return false

    // col validation
    for(i=0; i<9; i++)
        if(mat[i][col]==k)
            return false

    // box validation
    int startrow = // starting row of current box [row/3]*3
    int startcol = // starting col of current box [col/3]*3.

    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            if(mat[i+startrow][j+startcol]==k)
                return false

    return true.
```







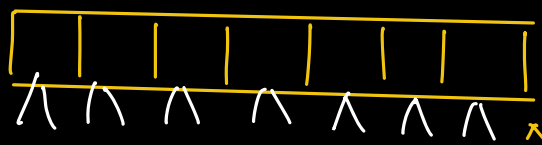
[0, 1, 2 - - - 8] —map.→ [0, 1, 2] → [0, 3, 6]

row

row/3

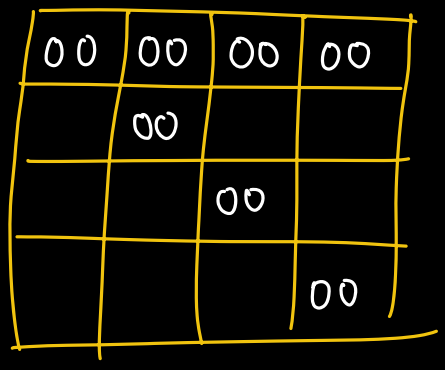
[row/3]*3

TC:

① 

N elements.

Total how many options = 2^n.

② 

$\Rightarrow N^2$ elements.

Total how many options = 2^{N^2}.

For sudoku, you've 9 options $\Rightarrow$ 9^{81} (81 cells in total).