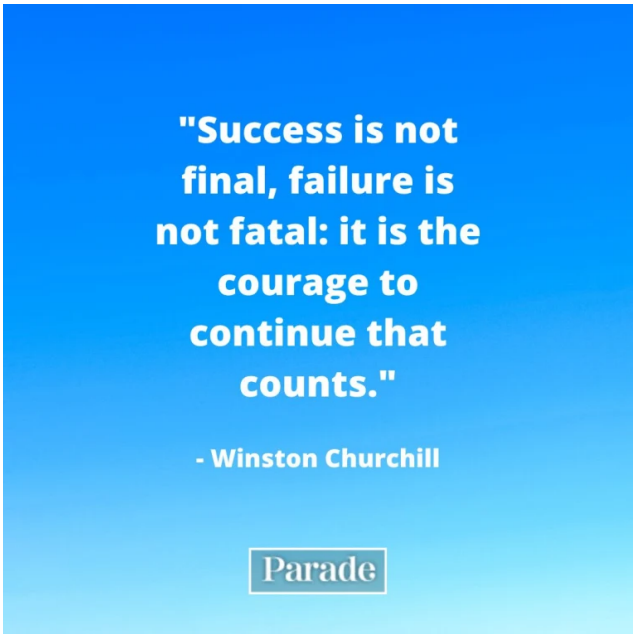
BACKTRACKING 2



Good Evening

To do list

01. N- Queens

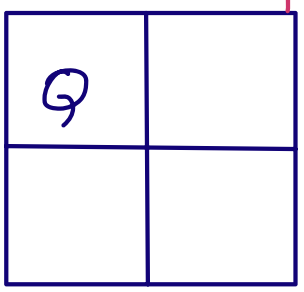
02. Sudoku

03. Word Break

Q Given N, print all valid placement of N queens such that no queen can kill other queen.

Note:- If two queens belong to same row/column/diagonal they will kill.

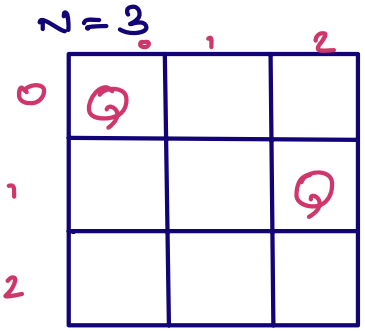
N = 2

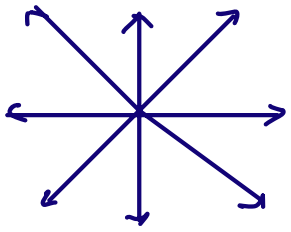


Not possible

Not possible

N = 3

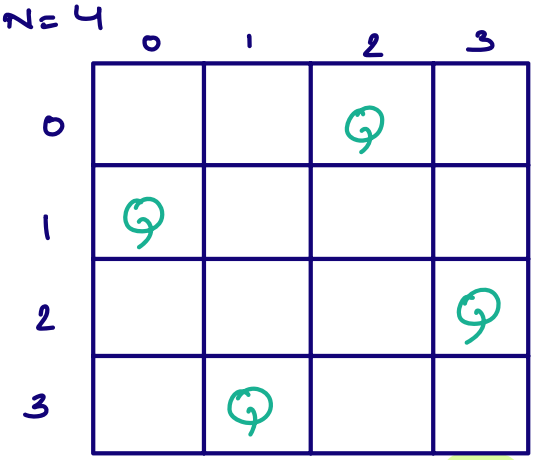




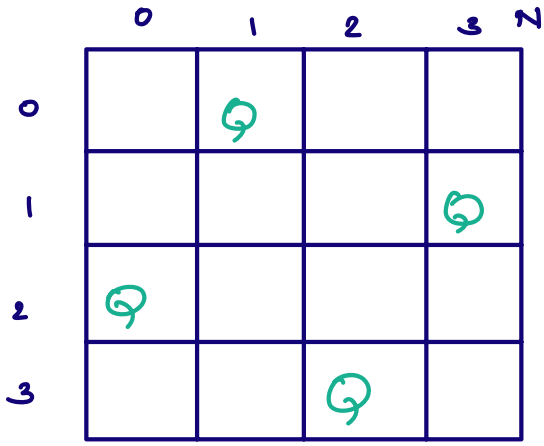
N = 1

possible

N = 4



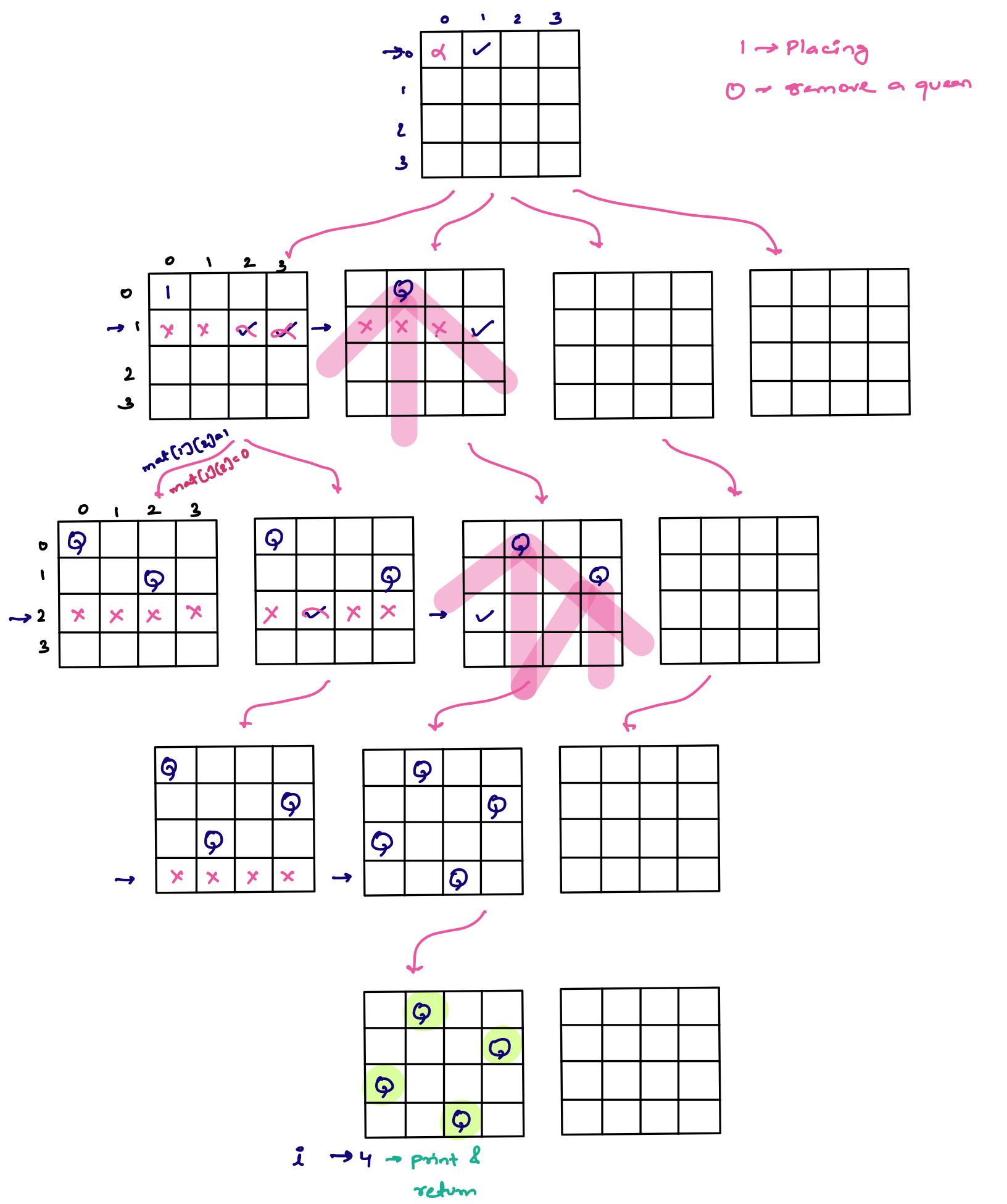
Valid



Valid

Observation → We can only place 1 queen in 1 row.

* Place queens row by row

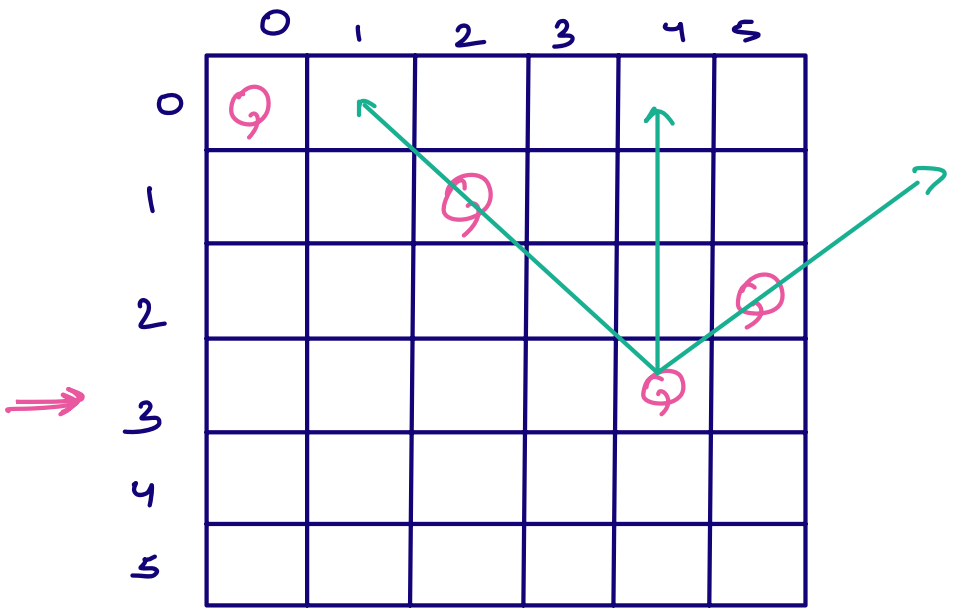

1 → Placing
0 → Remove a queen
mat[r][c]=1
mat[r][c]=0
i → 4 → print & return

```
void NQueens ( int[][] mat, N, i )        → row no.
{
    if (i == N)
    {
        print (mat[][])
        return;
    }

    for (j=0; j<N; j++){
        if ( isvalid (mat, i, j)) {
            mat[i][j] = 1;
            NQueens (mat, N, i+1)
            mat[i][j] = 0;
        }
    }
}
```

TC : $O(N * N!) \approx O(N!)$

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

- matrix creation
- stack space



TC : O(N)

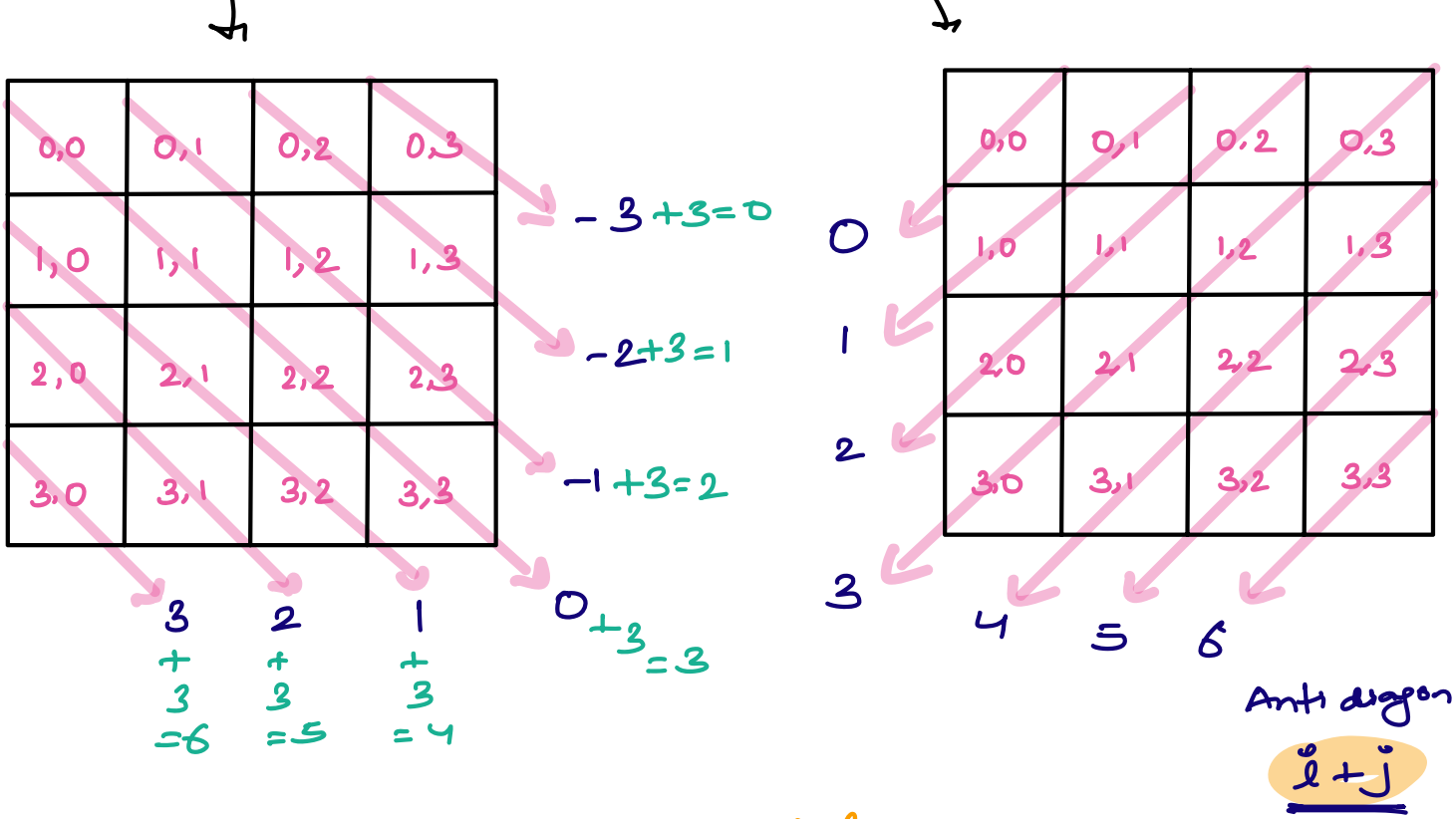
```
boolean isvalid ( int[][] mat , int row, int col)
                        3          4
```

(r, c)
(3, 4) → (2, 4) → (1, 4) → (0, 4)

```
for ( i = row - 1 ; i ≥ 0 ; i--) {
    if ( mat[i][col] == 1) return false
}

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

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

return true ;
}
```

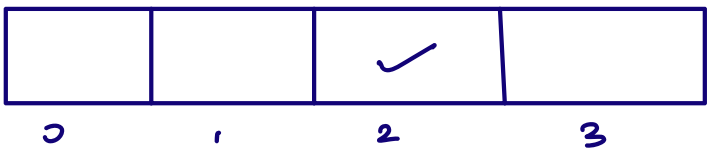
diagonals

Antidiagonals

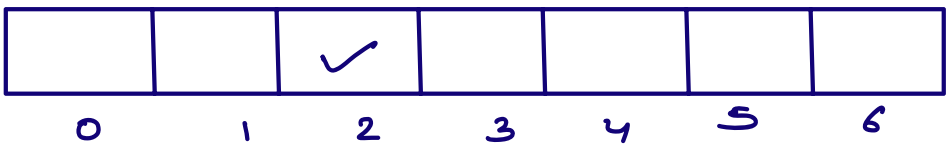


Placing queen at (0,2)

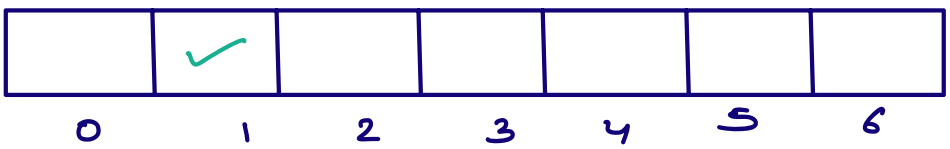
col =



Anti diagon



diagon



```
void Nqueens (int [][]mat, int N, cols[N], diag[2N-1],
                                    antidiag[2N-1], i)
{
    if (i==N)
    {
        print(mat[][])
        return;
    }

    for (j=0; j<N; j++)
    {
        if (cols[j] == false && antidiag[i+j] == false
            && diag[i-j+N-1] == false)
        {
            mat[i][j]=1
            cols[j]= true
            diag[i-j+N-1] = true
            antidiag[i+j] = true;

            NQueens(mat, N, cols, diag, antidiag, i+1);

            mat[i][j]= 0
            cols[j]= false
            diag[i-j+N-1] = false
            antidiag[i+j] = false
        }
    }
}
```

TC = O(N!)

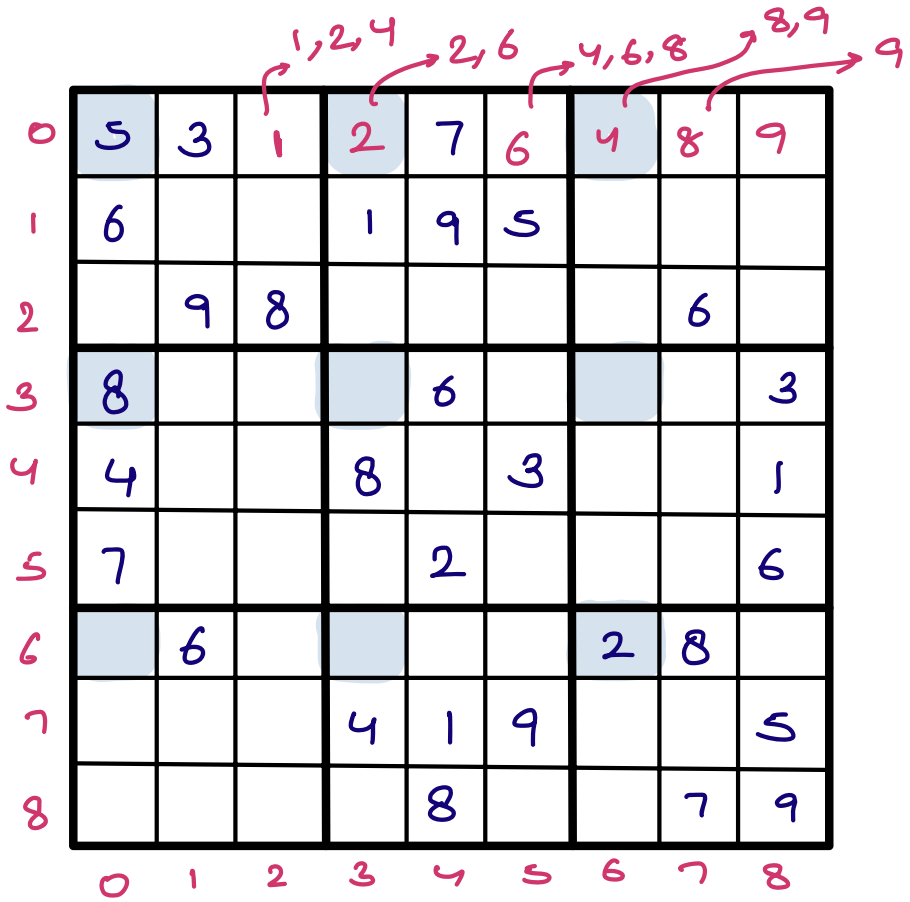
SC = O(N²)

3

10:12 pm → 10:22 pm

Sudoku

Given a partially solved state of sudoku. Find solution of sudoku. { Only 1 unique solution exist }



In row → (1 to 9)

In col → (1 to 9)

In 3*3 grid

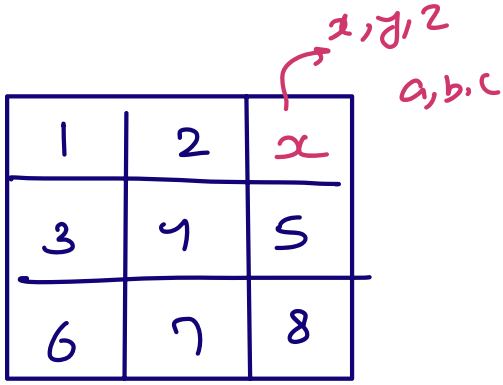
↳ (1 to 9)

empty cells

TC = $9^{}$

SC = $O(N^2)$

↳ recursive stack space



0 0

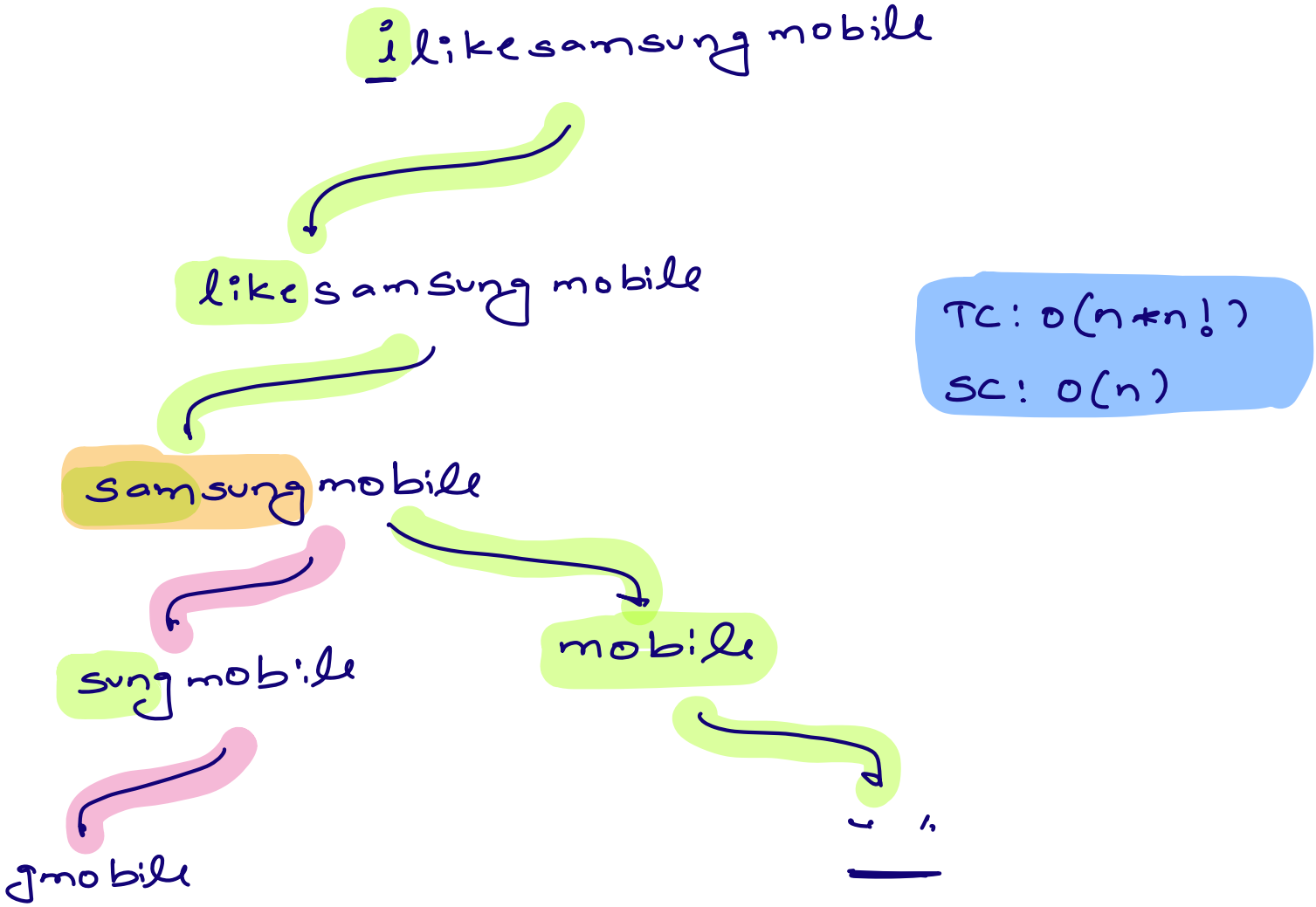
```
boolean sudoku (int[][] mat, int i, int j)
{
    if (j==9) { i=i+1 , j=0 }

    if (i==9) { return true; }

    if (mat[i][j] != 0 )
    {
        if (sudoku(mat, i, j+1) == true){
            return true;
        }
    }
    else {
        for (x=1; x<=9; x++){
            if (isvalid(mat, i, j, x)==true){
                mat[i][j] = x
                if (sudoku(mat, i, j+1) == true){
                    return true;
                }
                mat[i][j] = 0
            }
        }
    }
    return false;
}
```

```
boolean isvalid ( mat, row, col, x)
{
    for (i=0; i<n; i++){
        if (mat[row][i] == x || mat[i][col] == x)
            return false;
    }

    row = row - (row % 3)      // row = row/3 * 3
    col = col - (col % 3)      // col = col/3 * 3

    for (i=0; i<3; i++)
        for (j=0; j<3; j++){
            if (mat[row+i][col+j] == x) return false;
        }

    return true;
}
```

Q Given a dictionary of words & a string A. Determine if A can be segmented into space seperated sequence of one or more dict words

dict : { i like samsung sam sun mobile }

A Word - i likesamsung mobile



```
boolean wordbreak (HashSet<String> dict, String word)
{
    if (word.length() == 0) return true;

    for (i=1; i<word.length(); i++){

        String prefix = word.substring(0, i)

            if (dict.contains(prefix)){

                String ros = word.substring(i)

                if (wordbreak(dict, ros) == true)
                    return true;
            }
    }

    return false
}
```