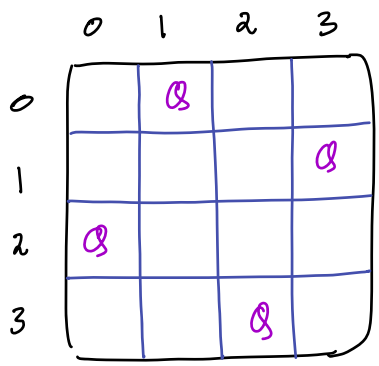
Today's Content:

- → N Queens :=
- → { Searching in 2D Sorted Matrix } → Binary Search : { Concept }
- → Sudoku
- → { Strings }

→ Back tracking : Trying all possible Solutions using Recursion.

N-Queens :

// given N*N, place N Queens

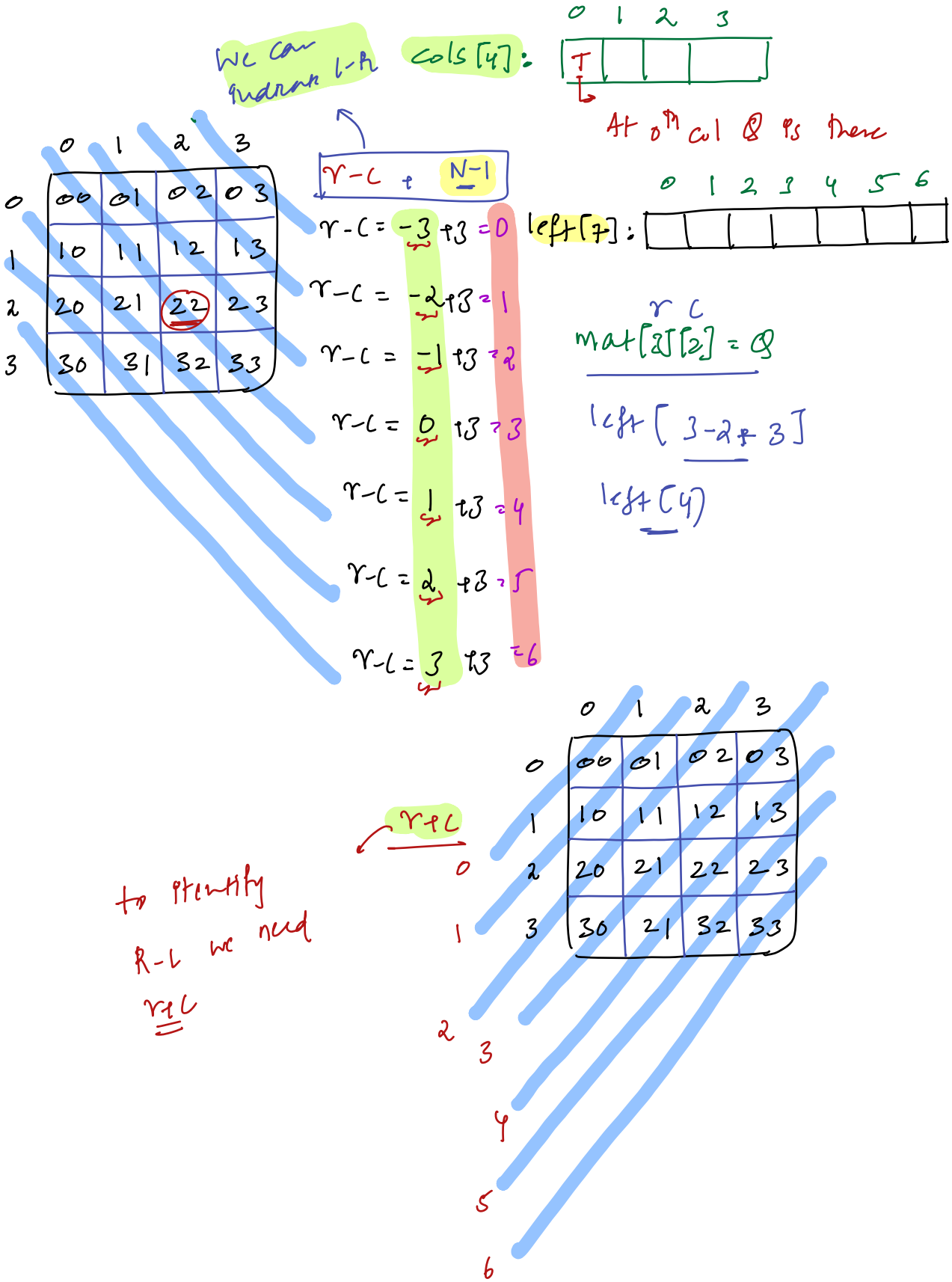


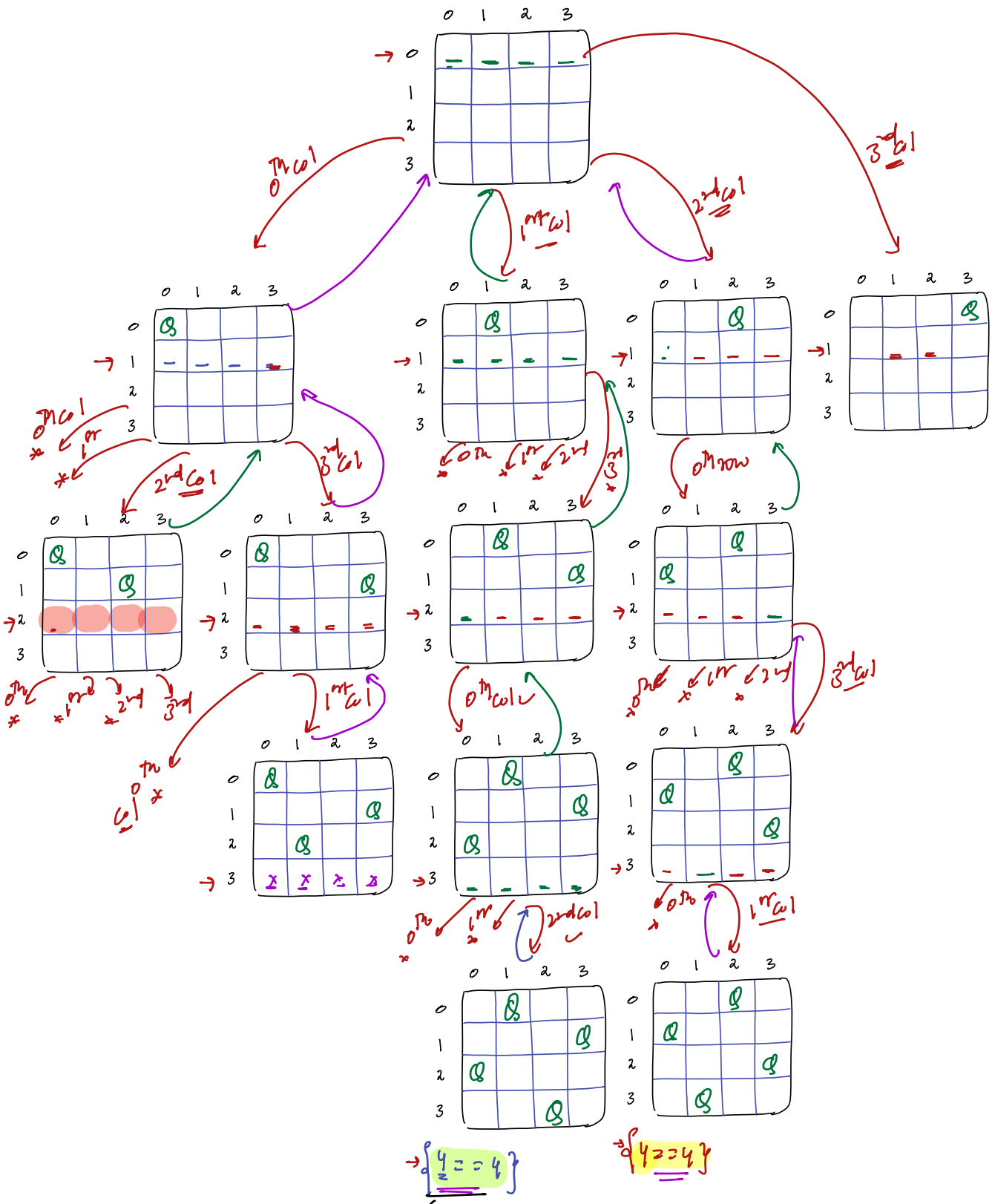
Given 4*4 place 4 Queens such that

No Queen should kill another Queen

now

obs: At every row we need to place a Queen


We can maintain l-h
cols[4]:
0 1 2 3
T
At 0th col Q is there
r-c + N-1
r-c = -3 +3 = 0
r-c = -2 +3 = 1
r-c = -1 +3 = 2
r-c = 0 +3 = 3
r-c = 1 +3 = 4
r-c = 2 +3 = 5
r-c = 3 +3 = 6
left[7]:
0 1 2 3 4 5 6
r c
mat[2][2] = Q
left[3-2+3]
left[4]
00 01 02 03
10 11 12 13
20 21 22 23
30 31 32 33
r+c
to identify
R-L we need
r+c
0 1 2 3 4 5 6



we filled all rows : print ans, return

we filled all rows : print ans, return

Parameters:

→ mat[N][N], row,

(All function calls share same mat[][])

```
void Queen(int mat[][], N, r)        // r is current row number
{
    if(r == N){ // Couple N-Queen
        print(mat[][])
        return;
    }
    c=0; c<N; c++){
        if(check(mat[], r, c)){
            mat[r][c] = 1
            Queen(mat, N, r+1)
            mat[r][c] = 0
        }
    }
}
```

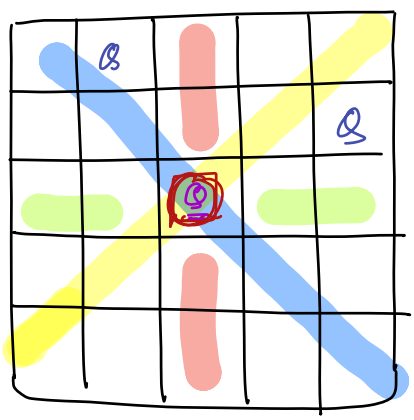
function calls:

→ Iterate q cases for { can col

mat[r][c] == 1 : Queen

mat[r][c] == 0 : Empty cell

//



→ mat[r][c] = 1

{ : check at column c
: Check for L-R diagnol
: Check for R-L diagnol }

Appl: Prevent in column & both L→R && R→L diagnols

↳ TC: O(N * N * N) SC: O(1)

```
void Queen(int mat[][], N, r, bool col[], bool left[], bool R[])
{
    if(r == N){ // Couple N-Queen        r → row number
        print(mat[][])
        return;
    }
    for(c=0; c<N; c++){
        if( col[c] == F && left[r-c+N-1] == F && R[r+c] == F )
            mat[r][c] = 1, col[c] = T, left[r-c+N-1] = T, R[r+c]=T
            Queen(mat, N, r+1)
            mat[r][c] = 0, col[c] = F, left[r-c+N-1] = F R[r+c]=F
        }
    }
}
```

to get leftdiagonal number

to get right diagonal number

bool col[N]

bool left[2N-1]

bool R[2N-1]

TC: $T(N) = N * T(N-1)$ (Recursive Relation)

$= N * \{N-1\ T(N-2)\}$

$= N * N-1 * N-2 * T(N-3)$

Ban can Time complexity

$T(N) = N! * N^2$ {Number of Complete Solutions}

$SC = O(N^2 + N + 2N-1 + 2N-1 + N)$

mat col[] left[] Right[] Stack Space

2Q) Given a 2D Sorted Matrix, Search for a element?

N = 4, M = 6 } In every row M elements

For every M Elements a row will increase

→ Solve 1 BS:

→ 2D → 1D: Apply BS

TC: $O(N*M) + \log(N*M)$

SC: $O(N*M)$ storing 1D array

	0	1	2	3	4	5
0	0: -4	1: 2	2: 5	3: 7	4: 11	5: 14
1	6: 18	7: 21	8: 25	9: 29	10: 32	11: 40
2	12: 41	13: 44	14: 47	15: 50	16: 52	17: 54
3	18: 55	19: 58	20: 60	21: 62	22: 65	23: 70

→ 2D —map→ 1D : Apply BS

TC: $O(\log_2(N*M))$

SC: $O(1)$

l = 0

h = N*M - 1

while (l <= h) {

p = (l+h)/2

{1D index} → {2D index}

r = p/M , c = p%M

if (mat[r][c] == k) {

return True

}

else if (mat[r][c] > k) {

h = p - 1

}

else {

l = p + 1

}

}

return False

	p/M = r	p%M = c
16 :	2	4
23 :	3	5
9 :	1	3
14 :	2	2

TODO: Please try using discussed approach

30) **Fill Sudoku** : Fill Sudoku → given 9*9 matrix

→ Every cell [1-9]

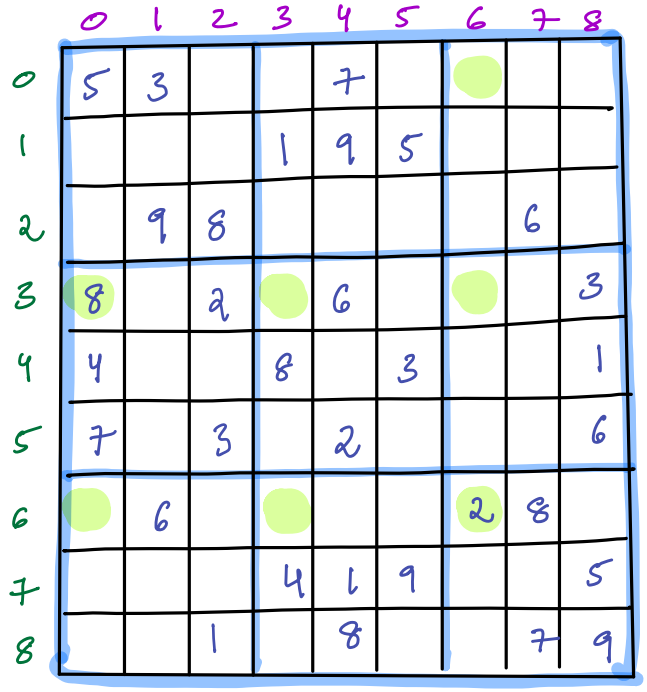
Rules:

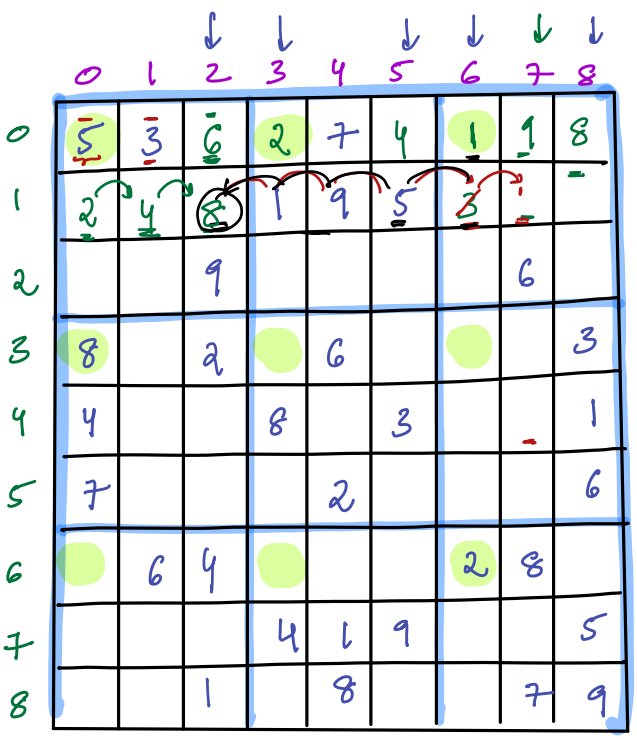
→ In row data cannot repeat

→ In col data cannot repeat

→ In 3*3 matrix data cannot repeat

→ mat[i][j] = 0 : not filled

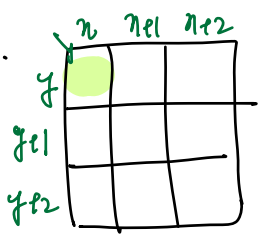




Empty cell:

1, 2, 3, 4, 5, 6, 7, 8, 9

→ start point of submatrix



```
q = x ; i < x+3 ; i++){
    j = y ; j < y+3 ; j++){
        mat[i][j]
    }
}
```

Q): given cell find start point of its submatrix ?

: r - r%3 , c - c%3

↳ start point of submatrix

parameters:

int mat[][], int i (i is ind in 1D)

function calls:

: {1, 2, 3, -- 9}

```
bool sudoko(int mat[][], int i){
    if(i==81){ //we filled our sudoko
        return True
    }

    i -> {1D} <=> {r = i/9, c = i%9}

    if(mat[r][c] != 0) { //goto next cell
        return sudoko(mat, i+1)
    }

    k = 1; k = 9; k++){
        if(check(mat[][], r, c, k)){
            mat[r][c]=k
            if(sudoko(mat, i+1)) {return True}
            mat[r][c]=0
        }
    }
    return false
}
```

check: private in r^{th} row check k; private in c^{th} col check k; private in {r-r%3, c-c%3} subbox check k

Ass: n empty cells:

TC: 9^n — SC: O(81)

it will be further less than that

worst case

TC: {9 * 9 * 9 -- 9} = 9^{81} (81 times)

SC: O(81) ≈ O(1)

Any Back Tracking

: parameters

: At a cell : All possibilities

updates

function( )

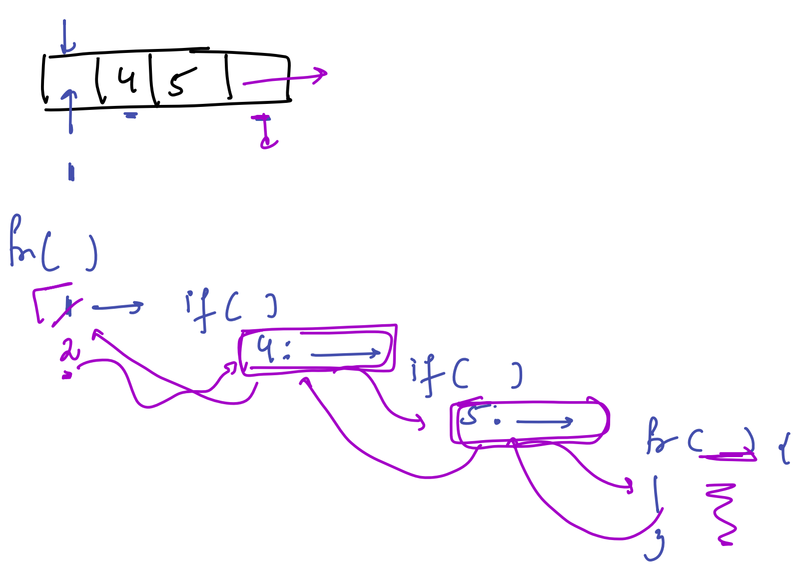
reverse

: Edge Case

$$T(n) = 9T(n-1)$$
$$= 9 \times 9\, T(n-2)$$
$$= 9 \times 9 \times 9 \times T(n-3)$$
$$= 9^n$$

Saturday: { Dynamic Programming }

- → Recursion + Back tracking
- → Please Revise



→ Store all index which are empty in ___

→ Set: { }