Today's Content

1) 2D Matrix Intro

2) Matrix Traversal Problems

Till Now :

- → Bits Manipulations
- → Time Complexity
- → Subarrays
- → Prefix Sum
- → Carry forward
- → Sliding Window
- → Contribution Technique
- → 2D
- → Interview Problems

Maths

Sorting

Hashmap's

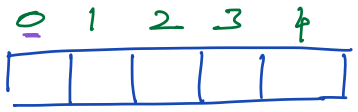
String

String + Hashmap

Recursion

Subsets / Subsequences

// int ar[5];



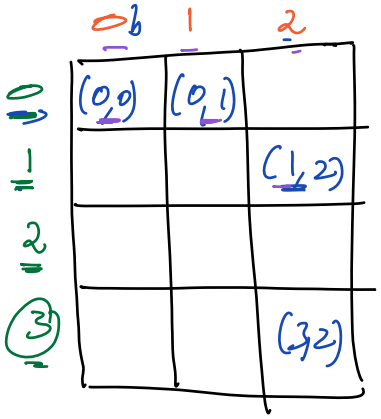
int mat[4][3]

Rows

Columns

Total Elements

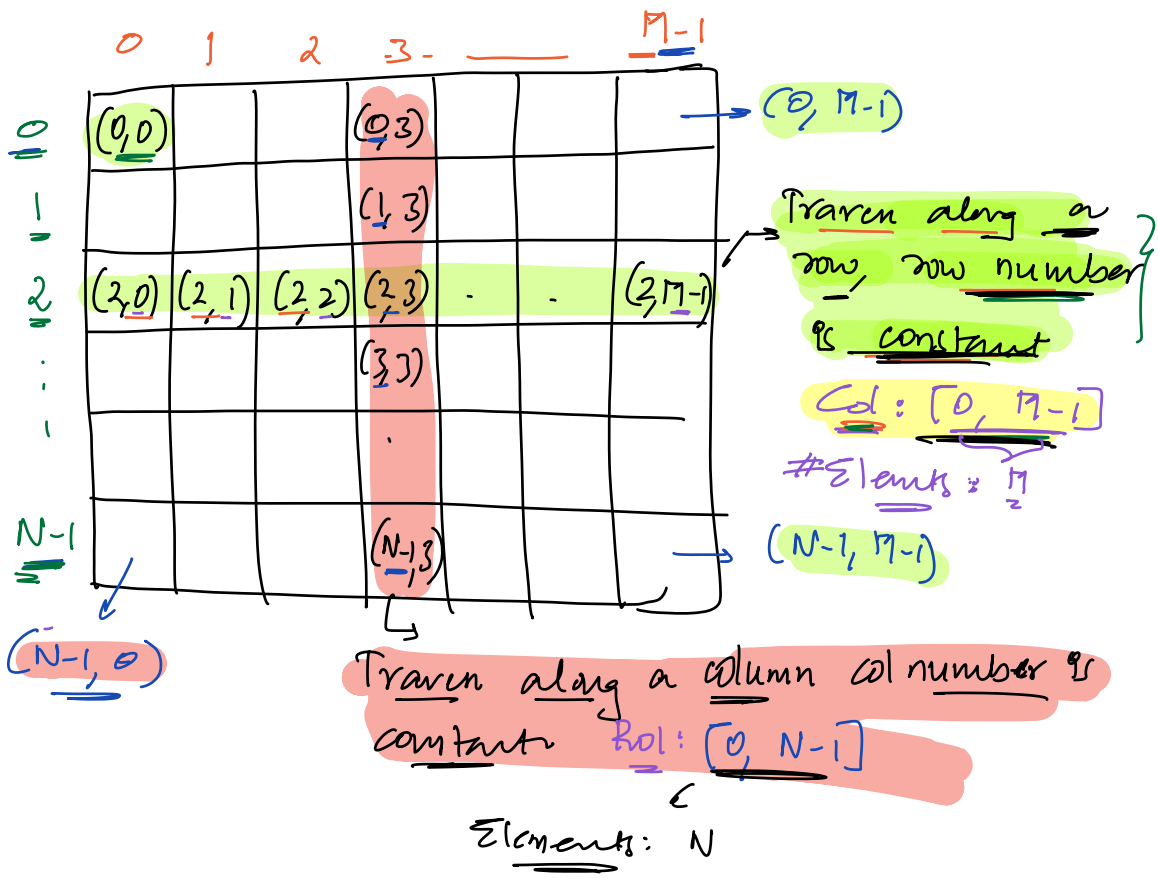
→ 12



// int mat[N][M]

N rows

M Columns



18) // Print row with sums

Given Mat[N][M], print k^{th} row's sum

k^{th} → [0 -- N-1]

```
for(k = 0; k < N; k++){
    Iterating k^th row
    sum = 0
    for(j = 0; j < M; j++){
        sum = sum + mat[k][j]
    }
    print(sum)
}
```

// Sum of every row right

Ex:

	0	1	2	3	
0	-2	1	6	3	: 8
1	2	9	3	8	: 22
2	3	-1	3	4	

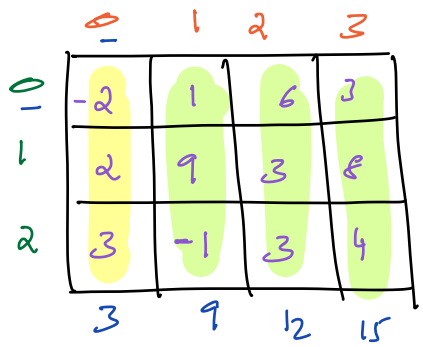
// output

k: 0 : j: [0 3] : 8

k: 1 : j: [0 3] : 22

k: 2 : j: [0 3] : 9

// Print column wise sums.




```
for(j=0; j<m; j++){
    // get j^th column sum
    sum=0
    for(i=0; i<n; i++){
        sum = sum + mat[i][j]
    }
    print(sum)
}
```

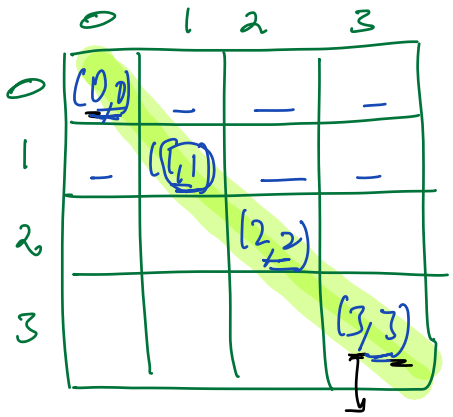
j=0 : i[0 2] : 3

j=1 : i[0 2] : 9

j=2 : i[0 2] : 12

j=3 : i[0 2] : 15

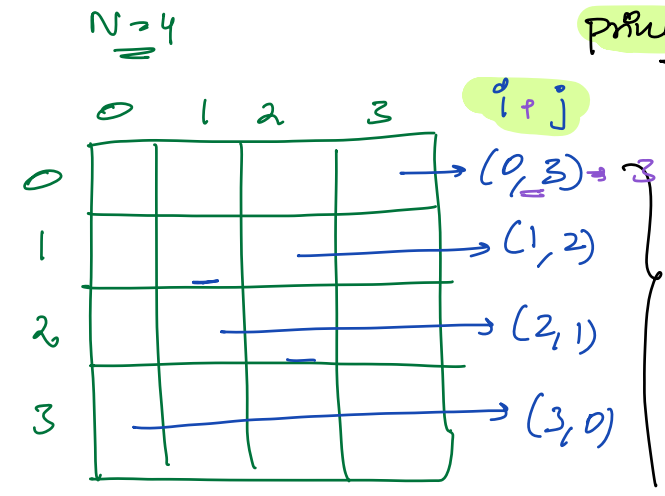
38) Given Square Matrix of N = 4,

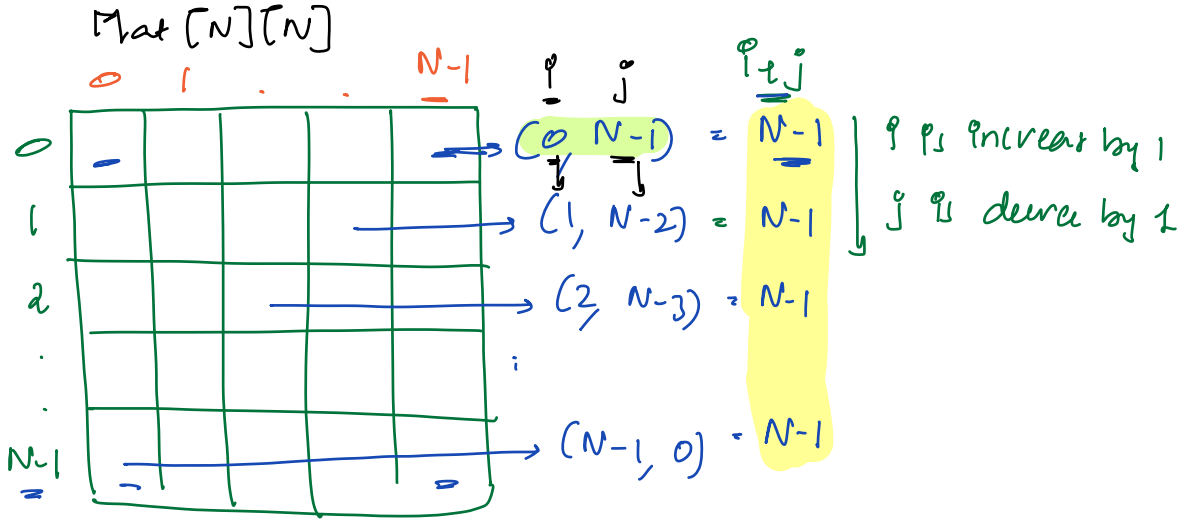


print L-R diagnol

TC: O(N)

```
for(i=0; i<N; i++) {
    print(mat[i][i])
}
```





// Printing R-L Diagonal Square Matrix

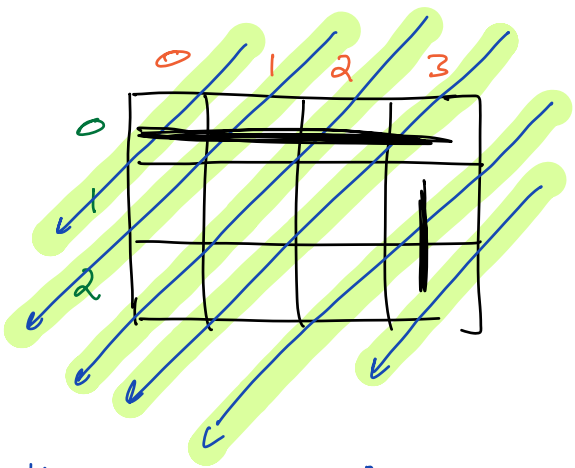
```
i = 0   j = N-1
while (i < N && j >= 0)
{
    print(Mat[i][j])
    i++; j--
}
```

TC : O(N)

4(8) Given Rectangular Matrix, Print all Diagonals going from R-L.

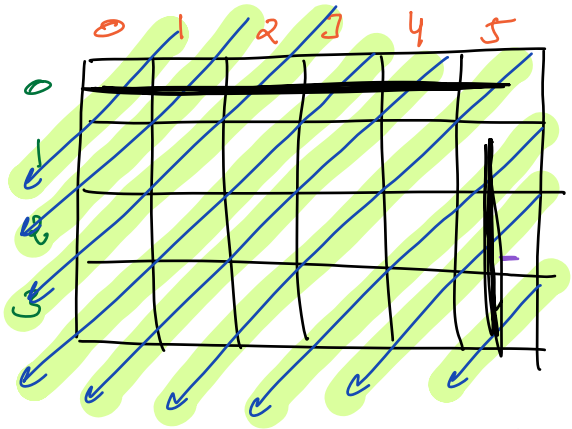
Mat[N][M]

mat[3][4]



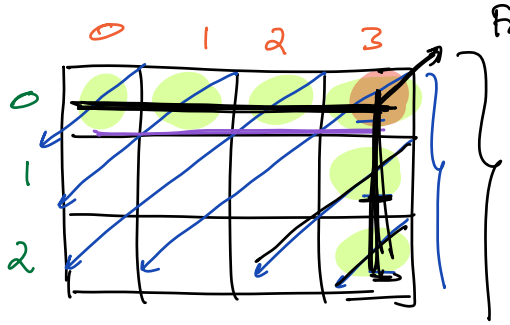
How many diagonals R-L : 6

mat[4][6]



How many diagonals from R-L = 9

How many diagonals from R-L = N+M-1



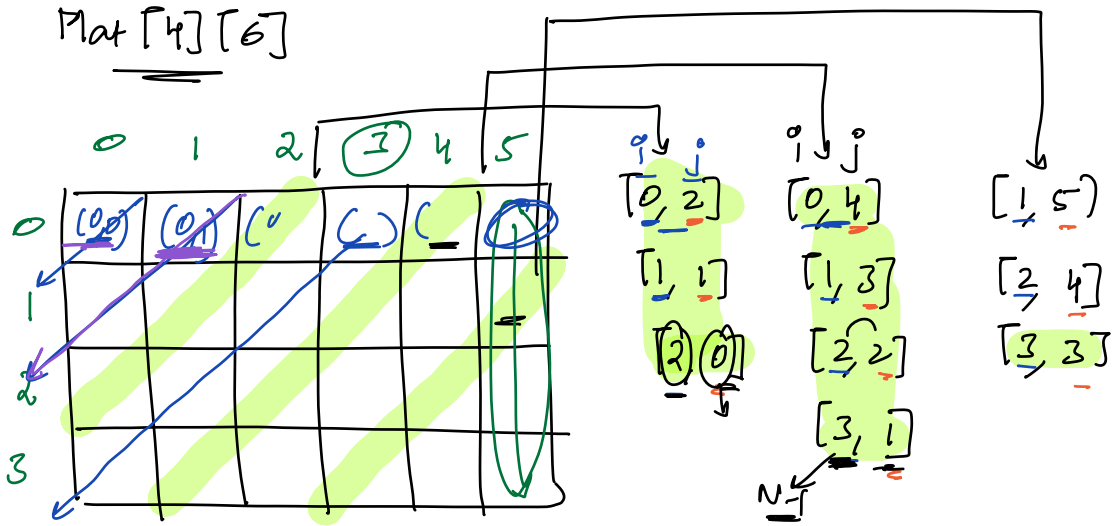
From which cells our diagonals can start

→ From Any cell in 0th Row : M #cells

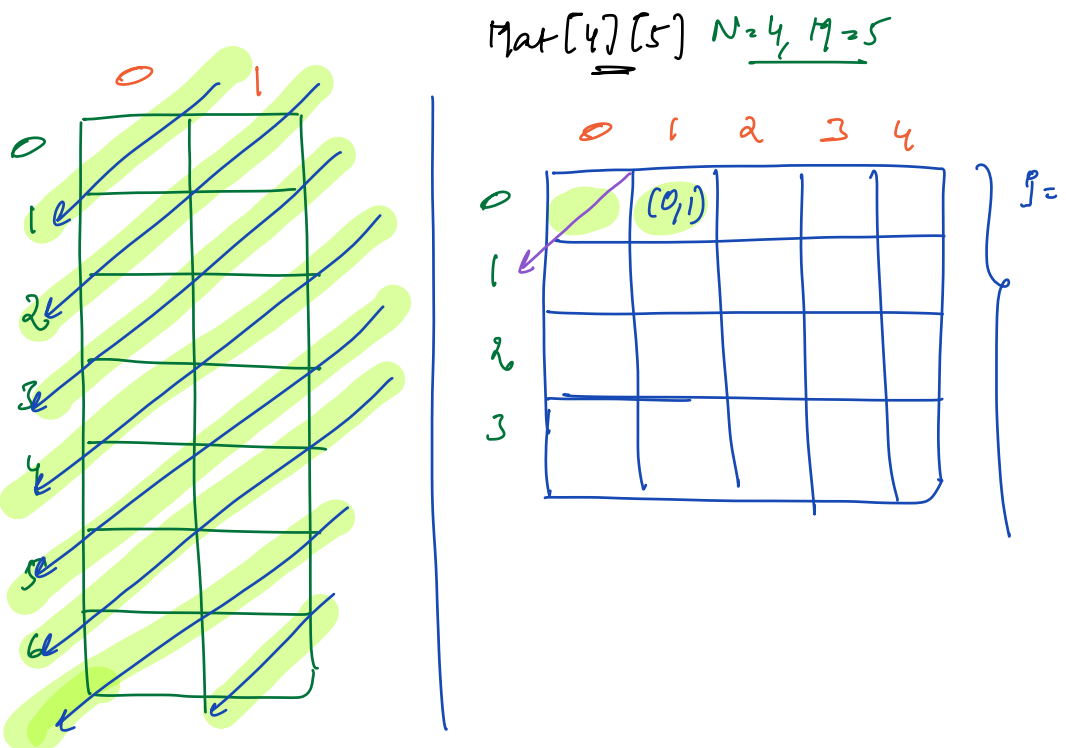
→ From Any cell in M-1th Column : N

→ Total Diagonals = N+M-1 ↦ top right cell is counted twice

// Printing all diagnols from R-L



// Across any diagnol R-L, After each cell, i++, j--



// Right diagonals can start from 0^{th} row a last column $(M-1)$

// →

// Printing all diagonals starting in 0^{th} row

```
for(j=0; j<M; j++){
    (i=0, j)
    while(i<N && j>=0)
        print(mat[i][j])
        i++; j--
    }
```

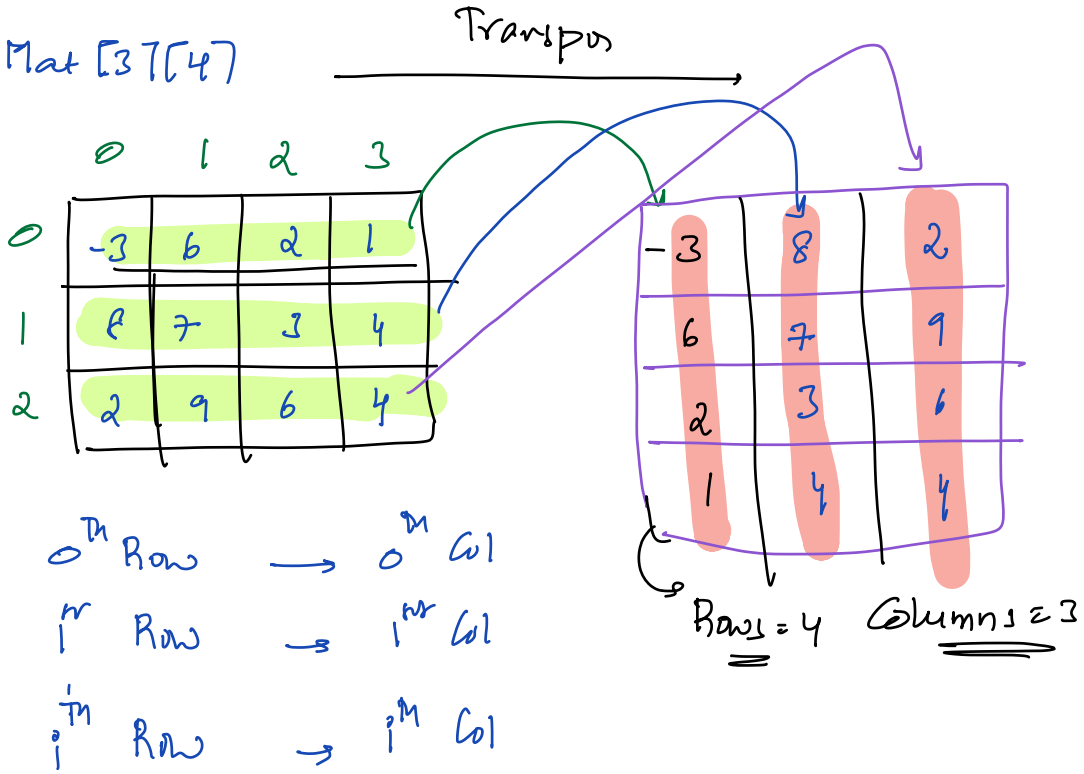
→ to avoid repeating.

// Printing all diagonals starting in M-1 column

```
for(i=1; i<N; i++){
    (i, j=M-1)
    while(i<N && j>=0){
        print(mat[i][j]
        i++; j--
    }
}
```

→ $O(N*M)$

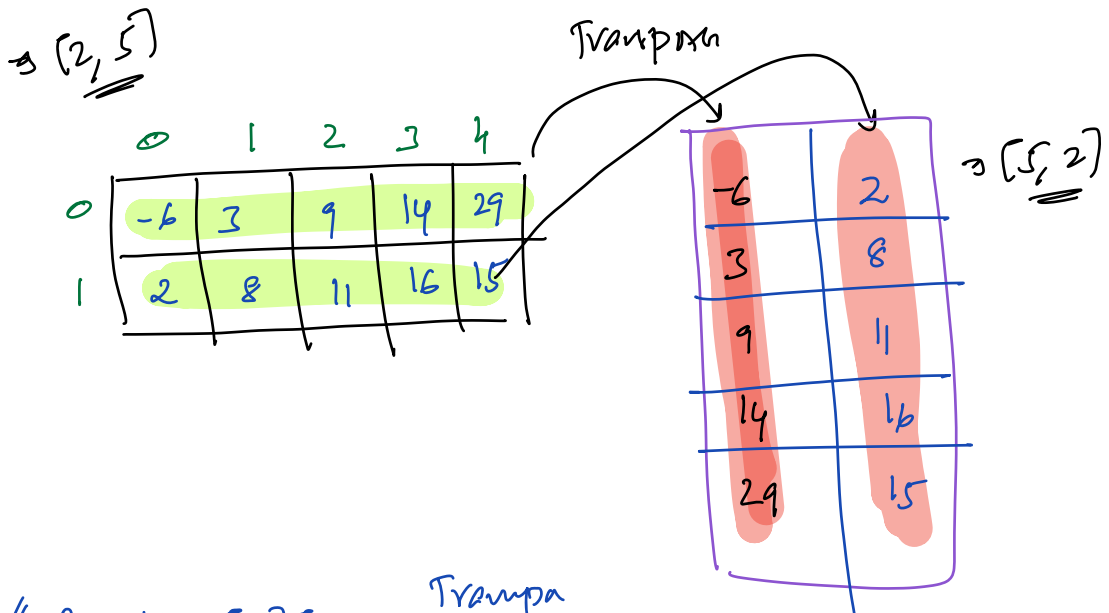
Transpose:



0^{th} Row → 0^{th} Col

1^{st} Row → 1^{st} Col

i^{th} Row → i^{th} Col



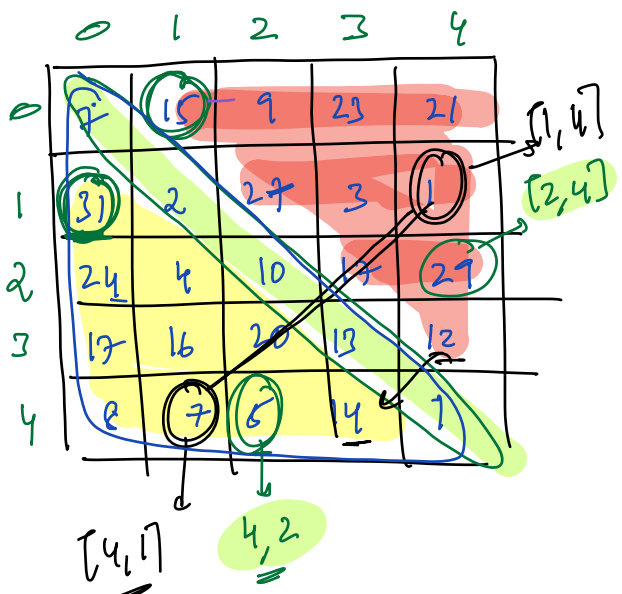
// Q: Mat [N][M] → Transpose → Mat [M][N]

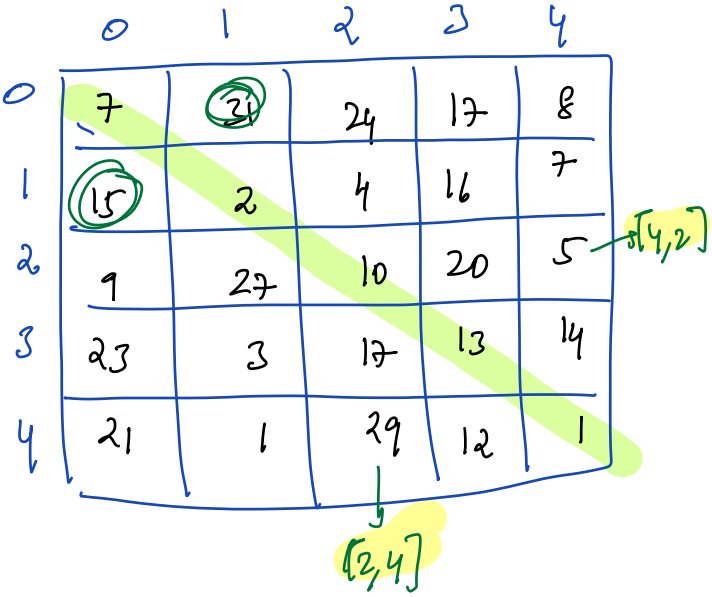
// Q: Mat [N][N] → Transpose → Mat [N][N] } ⇒ Shape Not changing

// Given a Square Matrix perform Transpose of Matrix inplace

{ without extra space }

Mat[5][5]





mat[i][j] $\overset{\text{swap}}{\rightleftharpoons}$ mat[j][i]

// Pseudocode

```
i = 0; i < N; i++) {
    j = 0; j < N; j++) {
        swap(mat[i][j], mat[j][i]
    }
}
```

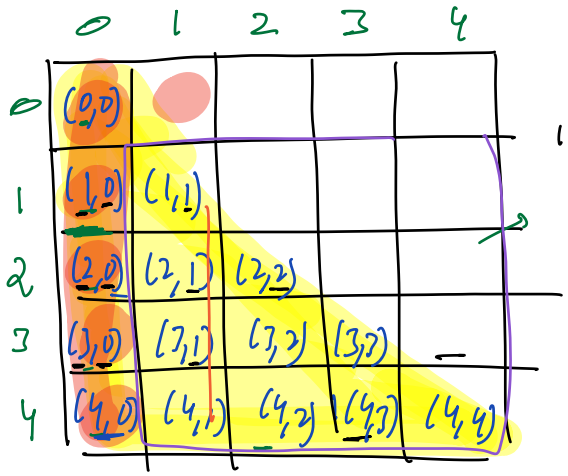
When (i, j) = (0, 3)
// swap mat[0,3] & mat[3,0]
When (i, j) = (3, 0)
// swap mat[3,0] & mat[0,3]

Same matrix

// Either Iterate on Upper Triangle or Lower Triangle

↓
TODO

→ Let's Iterate lower Triangle & Swap




```
for(i=0; i<N; i++){
    for(j=0; j<=i; j++){
        swap mat[i,j] & mat[j,i]
    }
}
```

→ $O(N^2)$

→ TC: $O(N^2)$ SC: $O(1)$

// Doubts

```
for(i=0; i<N/2; i++){
    // swap mat[i][N-1-i]
            mat[N-i-1][i]
}
```

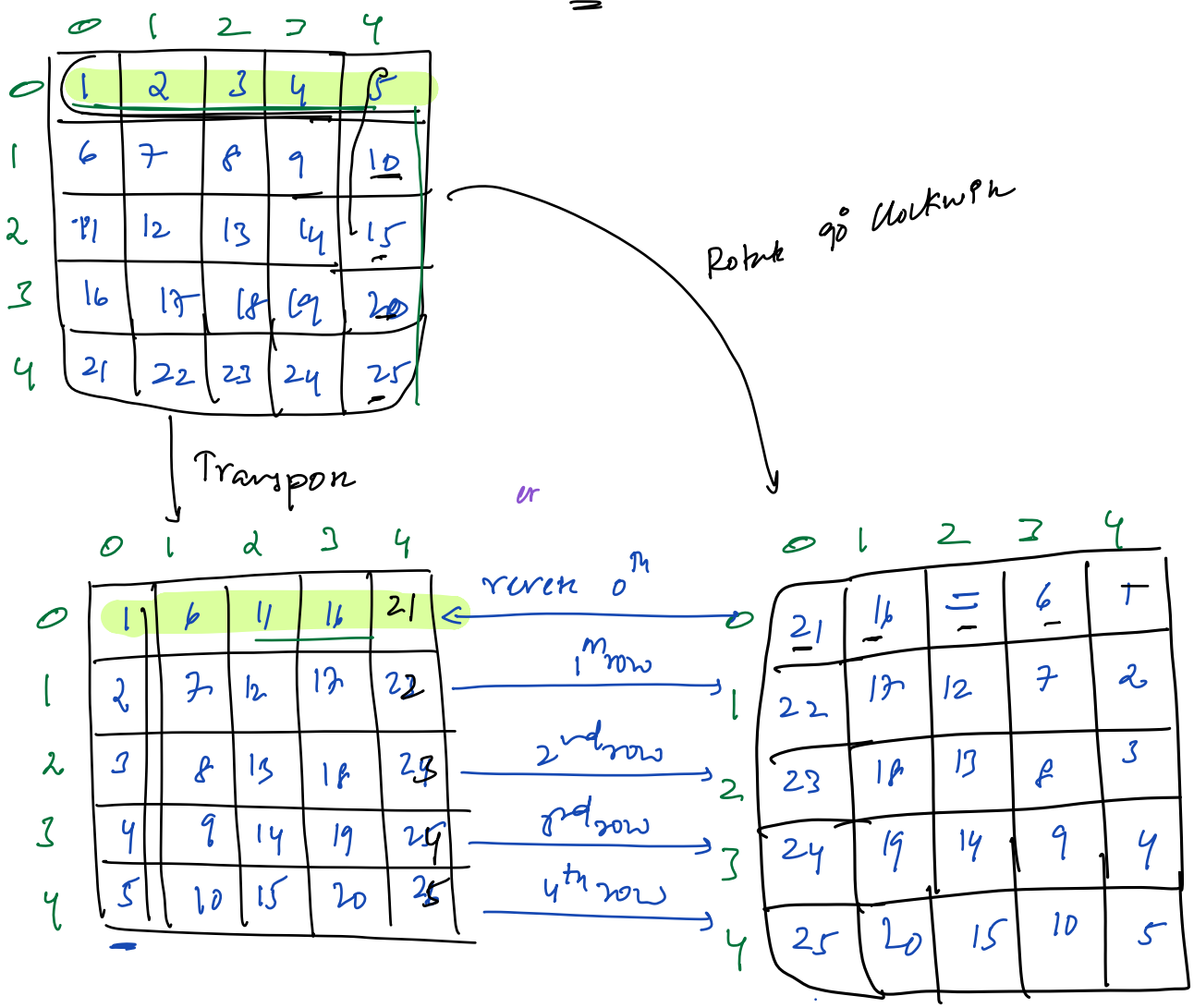
N = 5

i	N-i-1	N-i-1	0
i=0:	mat[0][4]	—	mat[4][0]
i=1:	mat[1][3]	—	mat[3][1]
i=2:	mat[2][2]	—	mat[2][2]
i=3:	mat[3][1]	—	mat[1][3]
i=4	mat[4][0]	—	mat[0][4]

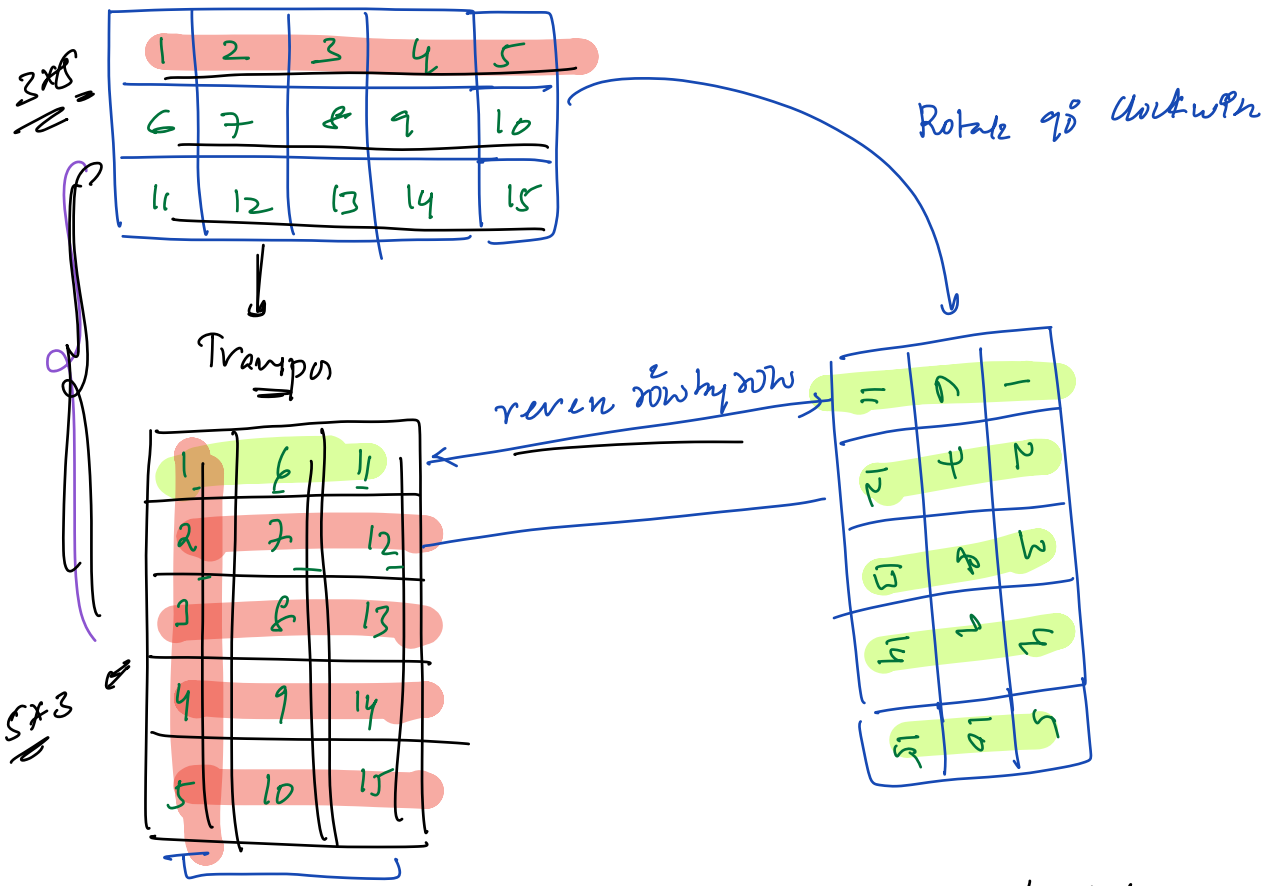
// Given mat[N x N] rotate by 90°, from clockwise

→ Step1: Transpose the mat
→ Step2: Reverse each row

N = 5



// Rotate 90° Rectangular



⇒ For a rectangular you need extra space to calculate transpose of a matrix.

⇒