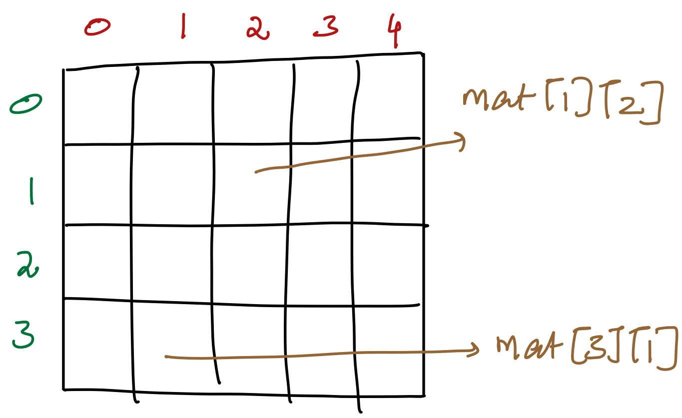
→ Max Subarray Sum : [Only Problem for TLE]

$\underbrace{N = 10^6}_{\text{Constraints}}$: ⇒ $O(N^2)$ ⇒ 10^{12} iterations ⇒ TLE

int mat[4][5]

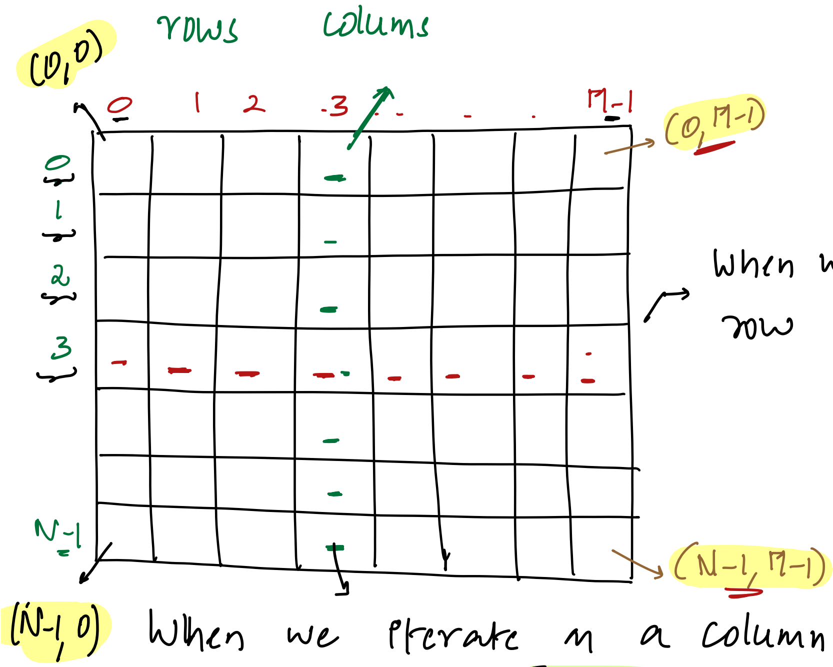
→ rows (4), columns (5)

→ Hori, Vertical



int mat[N][M]

→ rows (N), columns (M)



When we iterate on a row col change ⇒ [0, M-1]

M Elements

When we iterate on a column row is change ⇒ [0, N-1]

N Elements

// 1Q) Given mat[N][M], print row-wise

Ex:

	0	1	2	3
0	3	8	9	2
1	1	2	3	6
2	4	10	11	17

```
i = 0; i < N; i++) {
    j = 0; j < M; j++) {
        print(mat[i][j])
    }
    print("\n")
}
```

TC: $O(N*M)$ SC: $O(1)$

// 1Q) Given mat[N][M], find max column-wise sum.

Idea: For every column get sum & calculate overall sum

	0	1	2	3
0	3	8	9	2
1	1	2	3	6
2	4	10	11	13
Sum :	8	20	23	21

ans = 23

```
maxsum = INT_MIN
j = 0; j < M; j++) {
    sum = 0
    i = 0; i < N; i++) {
        sum = sum + mat[i][j]
    }
    maxsum = max(sum, maxsum)
}
return maxsum
```

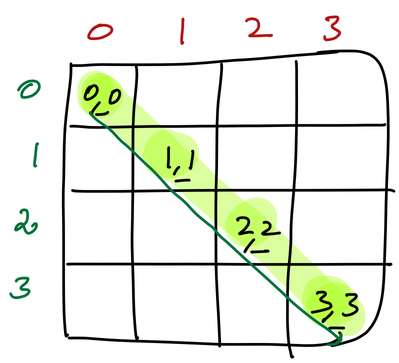
TC: $O(N*M)$

SC: $O(1)$

// Given mat[N][N] print diagnols : Top-left → Bottom Right

Top-right → Bottom left

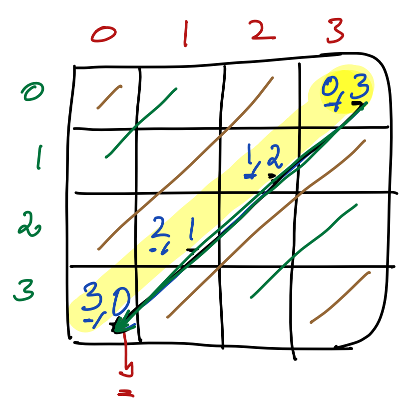
mat[4][4]




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

TC: O(N)

SC: O(1)




```
void PDiagRight( Mat[], N, i, j)
{
    while( i < N && j >= 0)
    {
        print(mat[i][j])
        i++, j--
    }
}
```

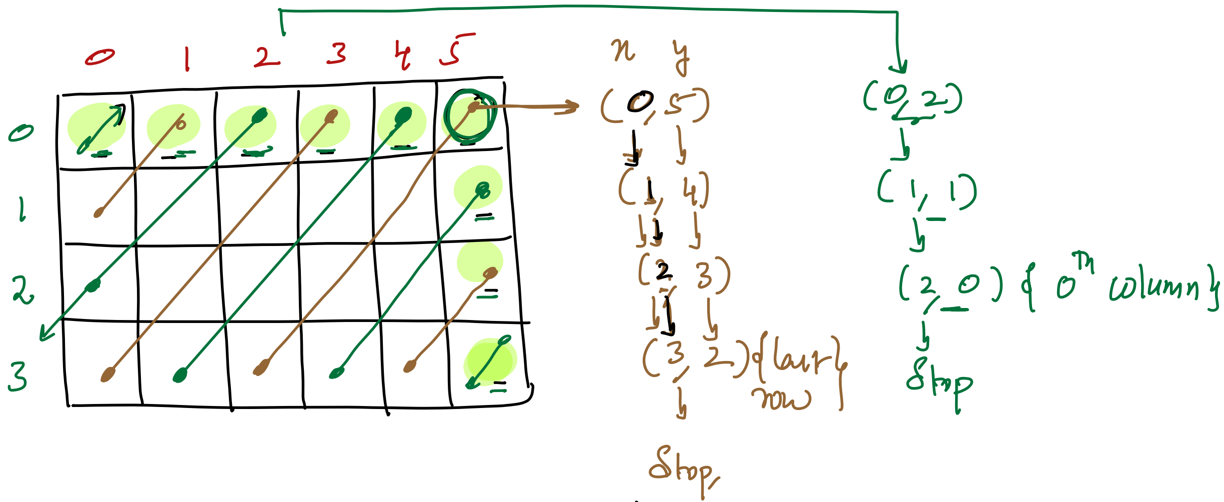
TC: O(N)

SC: O(1)

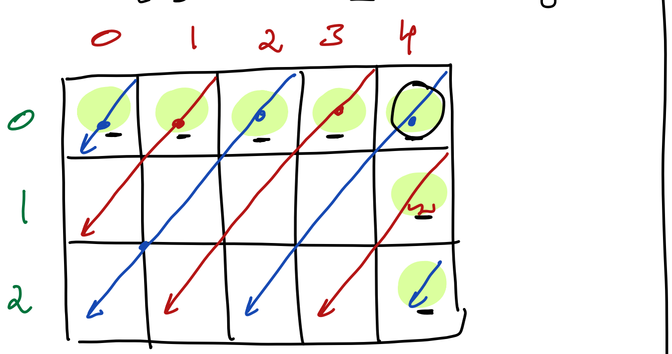
PdiagRight(mat[], N, 0, N-1)

4Q) Given a rectangular (N*M) matrix print all Diagnols going from R-L. } TC: O(N*M) SC: O(1)

mat[4][6] = 9 diagnols going R-L



mat [3] [5] → 7 diagonls going R-L



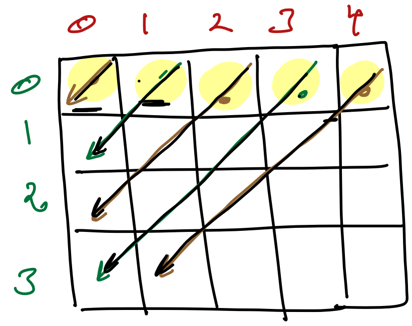
Note: Diagonls R-L either can start from 0^{th} row or Last Column

N M

$\boxed{\text{Total Diagnols R-L} = N+M-1}$

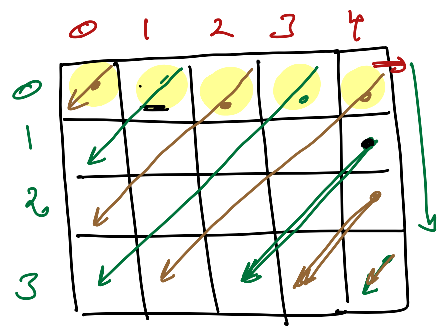
Top right counted in Both of them.

Step - I : Print all diagnols starting at 0^{th} row



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

Step - II Printing all diagons starting at last Column



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

TC : $O(N*M)$ SC : $O(1)$

→ 10:30 pm break

5Q)

SC: O(1)

Given a mat[N][N] → find transpose without any Extra space

→ {Matrix itself have to change}

mat[5][5] → mat[5][5]

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

5*5

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

mat[3][4] → 4*3

1	2	3	4
5	6	7	8
9	10	11	12

1	5	9
2	6	10
3	7	11
4	8	12

Note: For rectangle matrix to transpose you will need a Extra matrix mat[5][5]

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 6 & 7 & 8 & 9 & 10 \\ 2 & 11 & 12 & 13 & 14 & 15 \\ 3 & 16 & 17 & 18 & 19 & 20 \\ 4 & 21 & 22 & 23 & 24 & 25 \end{array}$$

mat[5][5]

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 6 & 11 & 16 & 21 \\ 1 & 2 & 7 & 12 & 17 & 22 \\ 2 & 3 & 8 & 13 & 18 & 23 \\ 3 & 4 & 9 & 14 & 19 & 24 \\ 4 & 5 & 10 & 15 & 20 & 25 \end{array}$$

// mat[0][1] $\overset{\text{swapped}}{\rightleftarrows}$ mat[1][0]

mat[0][2] $\rightleftarrows$ mat[2][0]

mat[3][2] $\rightleftarrows$ mat[2][3]

mat[i][j] $\rightleftarrows$ mat[j][i]

Pseudocode

(i = 0, j = 3) : swap mat[0][3] $\rightleftarrows$ mat[3][0]

i = 3, j = 0 : swap mat[3][0] $\rightleftarrows$ mat[0][3]

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

} // Original matrix itself.

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Note: Either iterate on upper triangle or on lower triangle.

→ TODO upper ✓

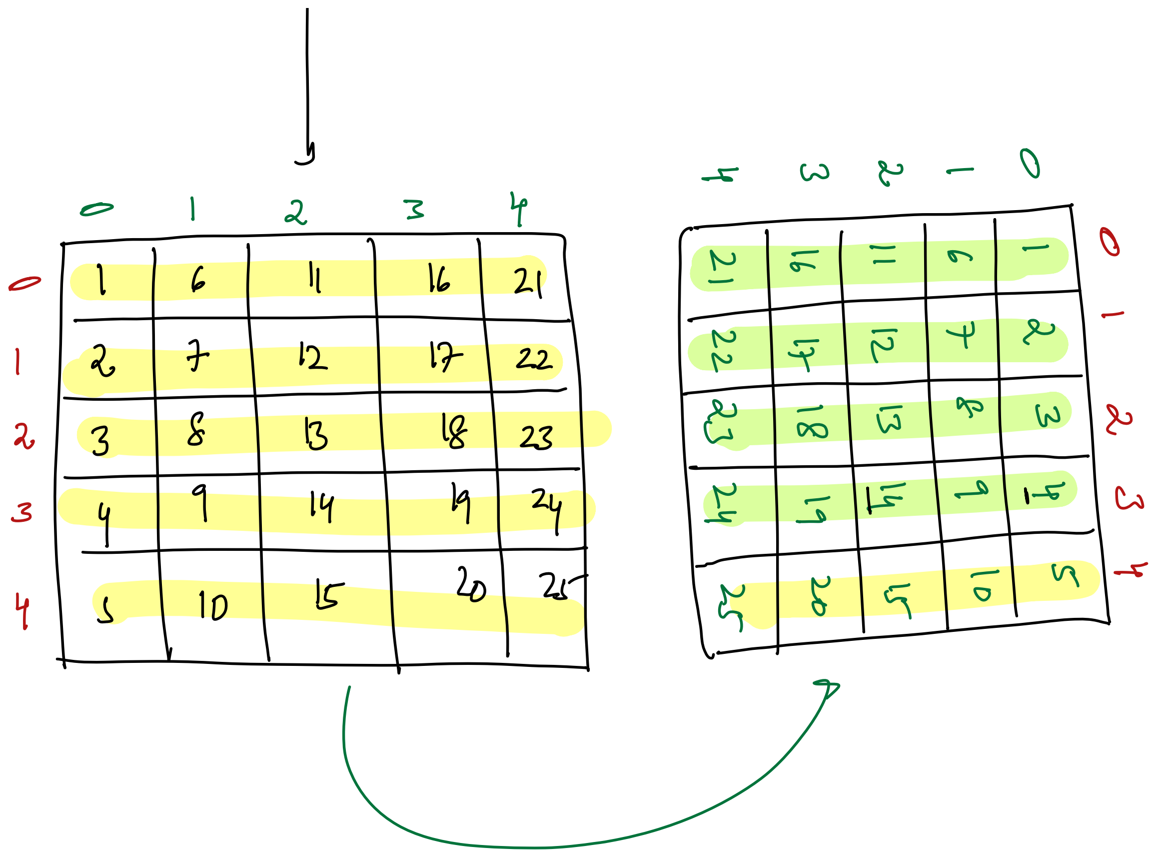
TODO lower ✓

Q8) Given a mat[N][N], rotate 90° in Clockwise, with O(1) space

Sol: Calculate Transpose & reverse each row.

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Calculate Transpose



Reverse each row

8Q) Given a N*N square matrix print boundary in clockwise direction.

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	2	22	23	24	25

Idea:

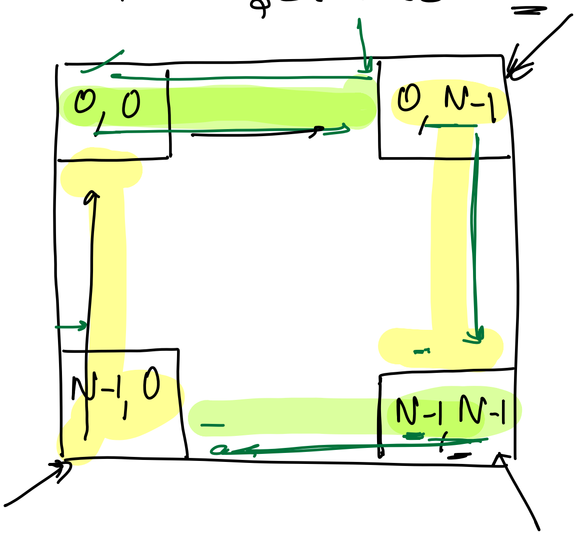
→ Print 0^{th} Row : [0, 0] → [0 3]

→ Print $N-1^{th}$ Col : [0 4] → [3 4].

→ Print $N-1^{th}$ row : [4 4] → [4 1].

→ Print 0^{th} Col : [4 0] → [1 0]

// Generalized Case N*N



Iterations

: [0, 0] → [0, N-2] : N-1 iterations

: [0, N-1] → [N-2, N-1] : N-1 iterations

: [N-1, N-1] → [N-1, 1] : N-1 iterations

: [N-1, 0] → [1, 0] : N-1 iterations

Boundary Printing

(i, j) = {0, 0}   [1, N) ⇒ N-1 times

k = 1; k < N; k++){ // k is just an iterator

print(mat[i][j]) } → N-1 prints

j++; } → N-1 it is increase

}

(i, j) = {0, N-1}

k = 1; k < N; k++)

print(mat[i][j])

i++;

}

(i, j) ⇒ {N-1, N-1}

k = 1; k < N; k++){

print(mat[i][j]

j--

}

(i, j) ⇒ {N-1, 0}

k = 1; k < N; k++){

print(mat[i][j])

i--;

}

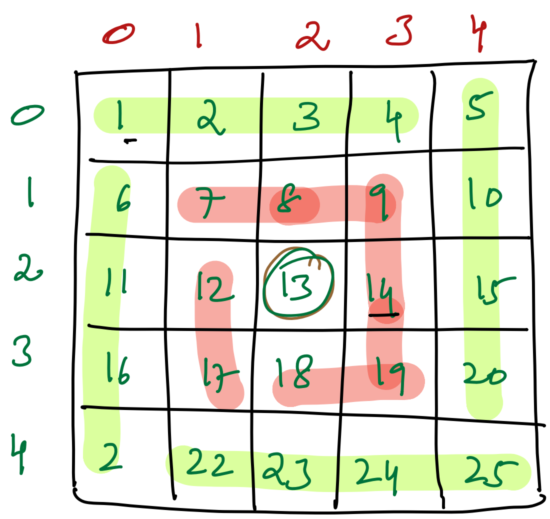
(i, j) = {0, 0}

Spiral Printing

```
(i, j) = {0, 0}

while( N >= 1) {
    k = 1; k < N; k++){
        print(mat[i][j])
        j++;
    }
    k = 1 ; k < N; k++)
        print(mat[i][j])
        i++;
    }
    k = 1 ; k < N; k++){
        print(mat[i][j]
        j--
    }
    k = 1; k < N; k++){
        print(mat[i][j])
        i--;
    }
    i++, j++,   N = N-2
                ↳ Because Square gets
                  shrinked
}
if( N % 2 == 1) {
    print( mat [i][j])
}
```

N=5

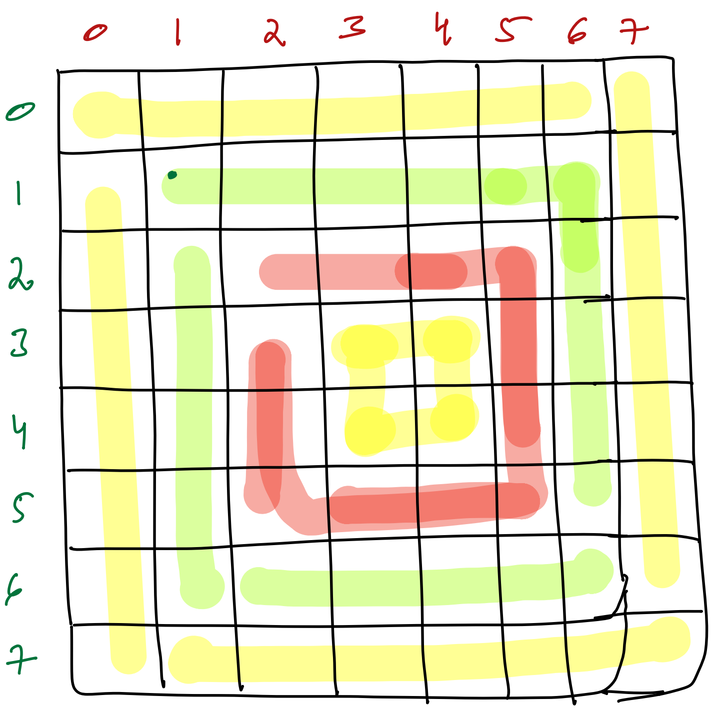


$i, j = (0, 0), \; N = 5 : \longrightarrow (0, 0)$

$i, j = (1, 1) \; N = 3 : \longrightarrow (1, 1)$

$i, j = (2, 2) \; N = 1 :$

N = 8×8



(i, j)	N	(i, j)	i	j	N
(0, 0)	8	0, 0	+1	+1	-2
(1, 1)	6	1, 1	+1	+1	-2
(2, 2)	4	2, 2	+1,	+1,	-2
(3, 3)	2	(3, 3)	+1,	+1,	-2
$\boxed{(4, 4)}$	$\boxed{0}$				