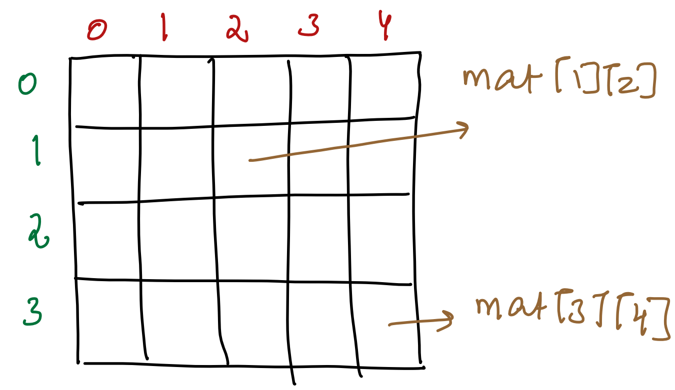
→ Satya Sai : 2019 VNR CSE 2 years DSA

→ Max Subarray Sum :
- $N \leq 10^6$ (TLE) : Time Limit Exceed
- Easy → Advanced Batch

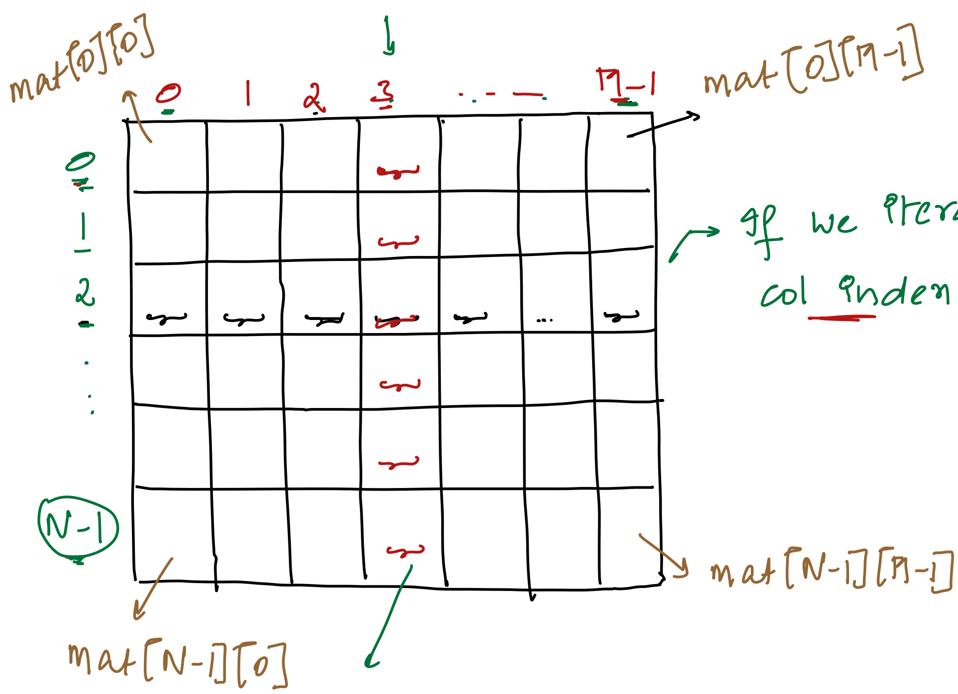
int mat[4][5]

rows — Horizontal

cols — Vertical



int mat[N][M]



If we iterate in a row

col index : [0, M-1] : M

If we iterate in a col

row index : [0, N-1] : N Elements

1Q) Given mat[N][M], print row-wise **TC: O(N*M) SC: O(1)**

mat[3][4]

	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")
}
```

2Q) Given mat[N][M], find max column sum

Idea: For each column get sum & calculate overall max.

	0	1	2	3
0	3	8	9	2
1	1	2	3	6
2	4	10	11	8
	8	20	23	16

- INFINE

```
maxsum = INT_MIN / mat[0][0] *

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

ans = 23

TC: O(N*M)

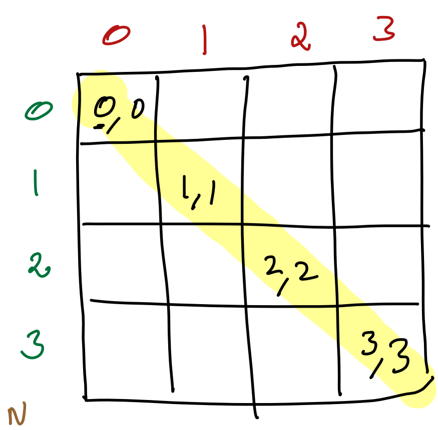
SC: O(1)

30) Given a mat[N][N] print diagnols → Left to Right

↳ Right to left

[0, - N-1]

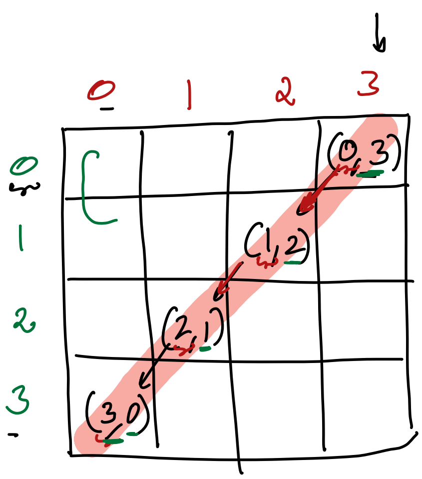
→ mat[4][4] N




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

TC: O(N)

SC: O(1)




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

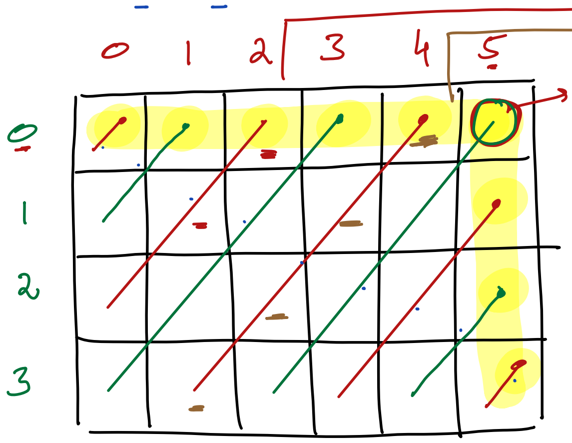
↳ dec

TC: O(N)

SC: O(1)

48) Given mat[N][M] print all diagonls going R-L

mat[4][6] → Diagonals : 9

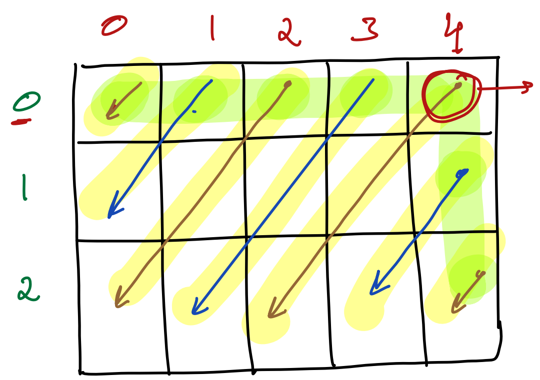


i j

(0, 4)
+1 ↓ ↓ -1
(1, 3)
+1 ↓ ↓ -1
(2, 2)
+1 ↓ ↓ -1
(3, 1)
last row

(0, 2)
↓
(1, 1)
↓
(2, 0)
0th column

mat[3][5] : Diagnols → 7



TC: O(N*M)

Note: R-L diagnols can start from

0th row or M-1th Col

M cells N cells

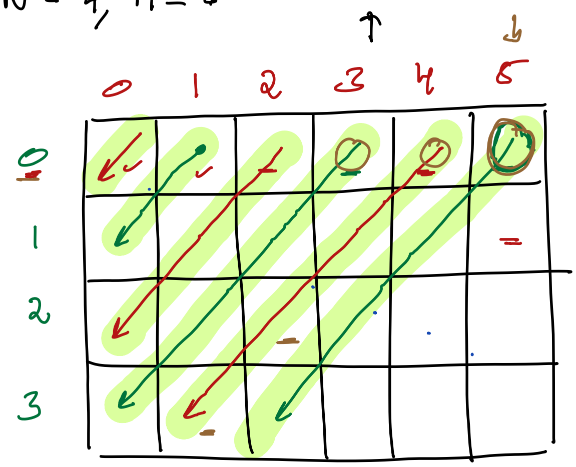
→ $(N+M-1) * \min(N, M)$

Total Diagnols = N+M -1

This comes in both 0th row & M-1th col.

// Print all diagonals starting 0^{th} row

N = 4, M = 6

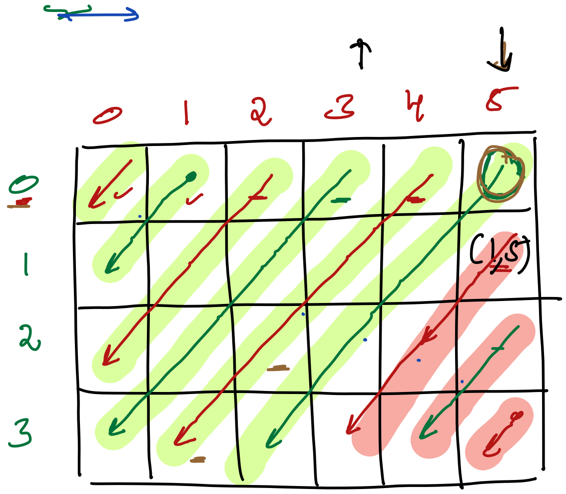



```
j = 0; j < M; j++) {        → cols

    (x, y) = (0, j)
    ↳ Take we start diagonal at (x, y)

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

// Print all diagonals starting at $(M-1)^{th}$ col




```
i = 1 ; i < N ; i++) {      → rows

    (x, y) = (i, M-1)

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

10:37PM

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

5Q) Given a mat[N][N] find the transpose inplace

1) Given input mat[][] should update, SC: O(1)

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

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] → After [4 x 3] we need a matrix

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

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

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] ⇄ mat[1][0]
mat[2][0] ⇄ mat[0][2]
mat[4][0] ⇄ mat[0][4]
mat[3][1] ⇄ mat[1][3]

mat[i][j] ⇄ mat[j][i]

mat[i][i] ⇄ mat[i][i]

mat[2][2] ⇄ mat[2][2]

Pseudo Code

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

i, j

0 3 → mat[0][3] ⇄ mat[3][0]

3 0 → mat[3][0] ⇄ mat[0][3]

Will give same matrix

	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

Iterate in upper triangular matrix & do swap

(TODO) ✓

Iterate in lower triangular matrix & do swap.

(TODO) ✓

68) Given a mat[N][N] rotate 90° Clockwise, SC: $O(1)$

	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

Rotate 90°

→ Transpose + Reverse Each Row

$O(N^2)$ $N * O(N) : O(N^2)$

→ Overall TC: $O(N^2)$

→ Space Complexity : $O(1)$

rotate

Transpose

	0	1	2	3	4
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

reverse

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

⇒ mat[3][2] → After rotating : 2*3

1	2
3	4
5	6

5	3	1
6	4	2

3 Given a mat[N][N] print all boundaries in clockwise

	0	1	2	3	4
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

output

Print 0^{th} row : [0,0] → [0 3] : 4 E

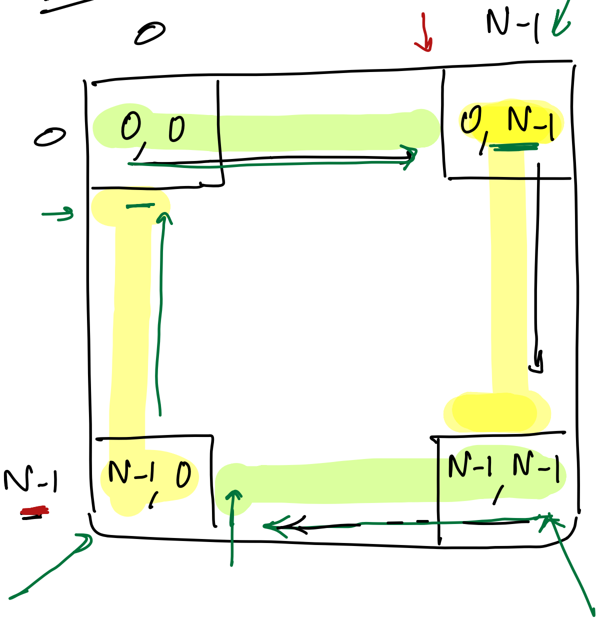
Print $N-1^{th}$ col : [0,4] → [3 4] : 4 E

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

Print 0^{th} col : [4,0] → [1,0] : 4 E

[a b] = b - a + 1

N*N Matrix



[0, 0] → [0, N-2] : N-1 Elements

[0, N-1] → [N-2, N-1] : N-1 Elements

[N-1, N-1] → [N-1, 1] : N-1 Elements

[N-1, 0] → [1, 0] : N-1 Elements

→ Border Printing

$i, j = (0, 0)$

```
k = 1 ; k < N ; k++) {        → k is used for iteration
                                 iterating N-1 times
    print (mat[i][j])
    j++ //
}
```

$// \ i, j = (0, N-1)$

```
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-- ; // N-1
}
```

$// \ i, j = (N-1, 0)$

```
k = 1 ; k < N ; k++) {
    print (mat[i][j]
    i-- ;
}
```

$// \ (i, j) = 0, 0$

$i, j = (0, 0)$ → Spiral Printing

TC: $O(N^2)$

SC: $O(1)$

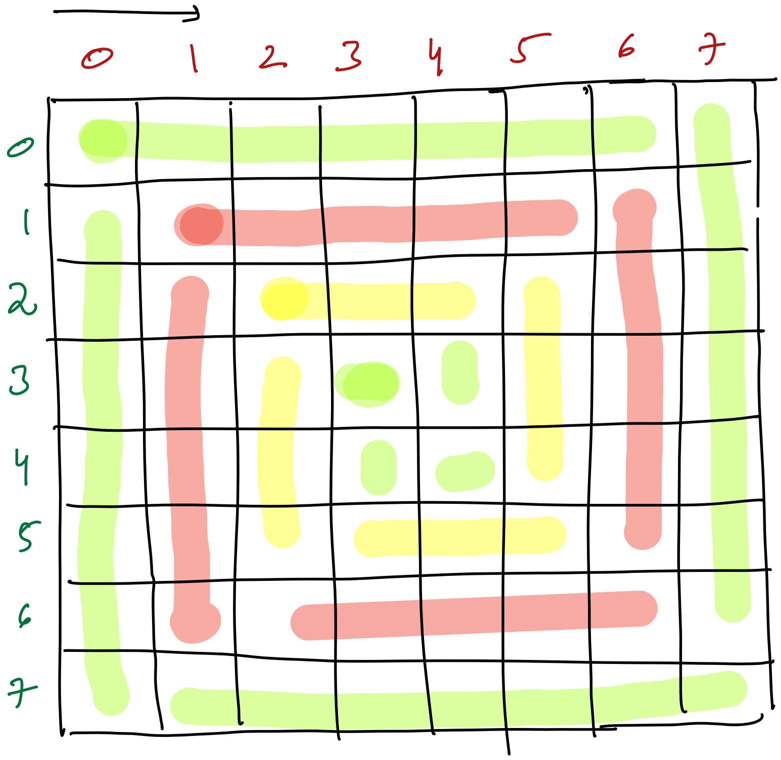
```
while( N > 1) {
    k = 1 ; k < N ; k++) {
        print(mat[i][j])
        j++ //
    }
    k = 1 ; k < N; k++) {
        print(mat[i][j])
        i++ //
    }
    k = 1 ; k < N; k++) {
        print(mat[i][j])
        j--; // N-1
    }
    k = 1; k < N; k++) {
        print(mat[i][j]
        i--;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

	0	1	2	3	4
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

i, j	N	i, j	i++,	j++,	N=N-2
0, 0	5	0, 0	1,	1,	3
1, 1	3	1, 1	2,	2	1
2, 2					

① → we are missing center element

mat[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
4, 4	0				

// If Rectangular matrix : TODO