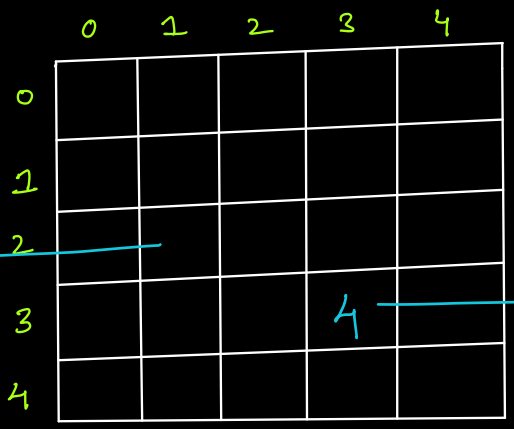
2D Arrays [BASICS]

int mat [5] [5]

columns

rows

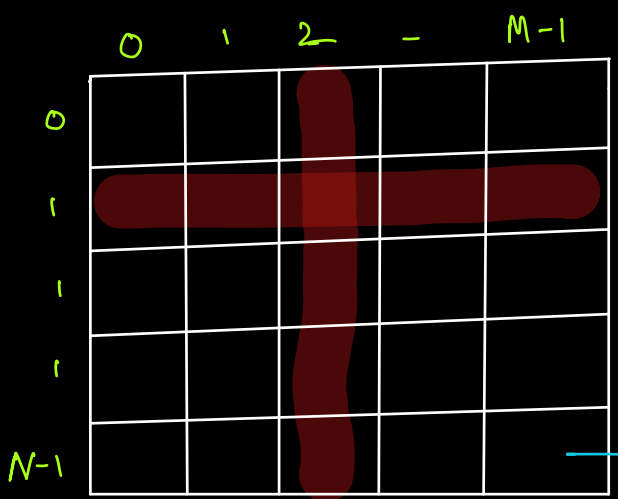


mat [2] [1]

mat [3] [3]

O(1)

int mat [N] [M]



Keep i = 1
j : [0, M-1]
mat [i] [j]

mat [n-1] [n-1]

Keep j = 2
i : [0, N-1]

Q1) Given a mat[N][M], print row-wise sum.

mat[3][4]

	0	1	2	3	
0	3	8	9	2	22
1	1	2	3	6	12
2	4	10	11	17	42

```
int sum => 0
for (int i=0 ; i<n ; i++) {
    sum => 0
    for (int j=0 ; j<m ; j++) {
        sum = sum + mat[i][j];
    }
    print sum;
}
```

Tc: O(mn)
Sc: O(1)

Q2) Given mat[N][M], find max colum sum.

	0	1	2	3
0	3	8	9	2
1	1	2	3	6
2	4	10	11	17
	8	20	23	25

```
int sum => 0
int max_Sum = INT.MIN;
for (int j=0 ; j<m ; j++) {
    sum => 0
    for (int i=0 ; i<n ; i++) {
        sum = sum + mat[i][j];
    }
    max_Sum = max(max_Sum, sum);
}
print max_Sum
```

Tc: O(mn)
Sc: O(1)

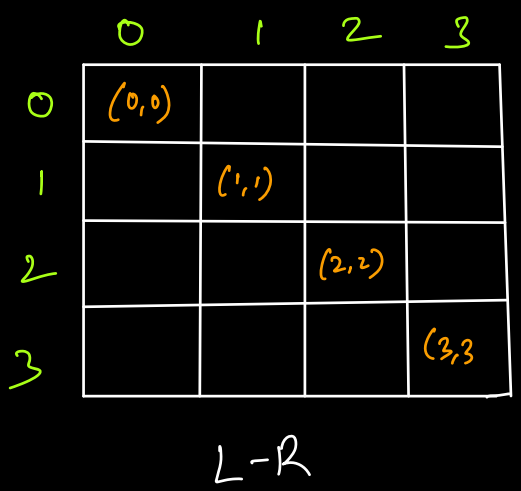
Q3 Given a mat [N] [N], print diagonals.

Case 1 : Left → Right

Case 2 : Right → Left

CASE 1

mat [4] [4]




```
for (int i=0; i<n ; i++) {
    for (int j=0 ; j<m ; j++) {
        if (i==j)
            print mat [i][j]
    }
}
```

3

3

Tc : O(n^2)

sc : O(1)

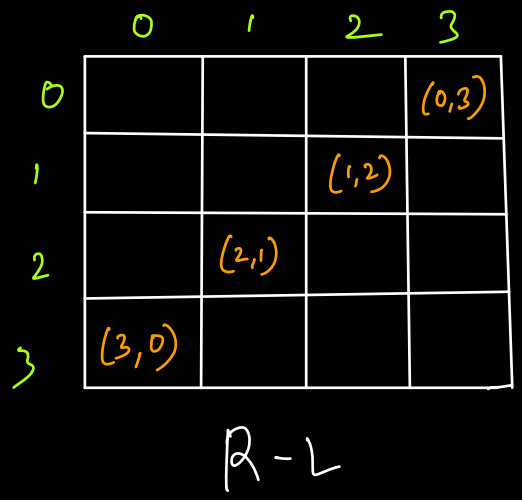
Optimized solution.

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

Tc : O(n)

sc : O(1)

CASE2



R-L

i	j
0	3
1	2
2	1
3	0
i	(n-i-1)

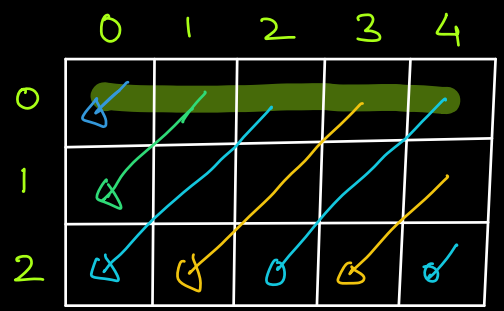
```
i=0, j= n-1
while (i<n) {
    print mat[i][j];
    i++;
    j--;
}
```

```
i=0,
while (i<n) {
    print mat[i][n-i-1]
    i++;
}
```

TC: O(N)
SC: O(1)

Diagonals in a Rectangle

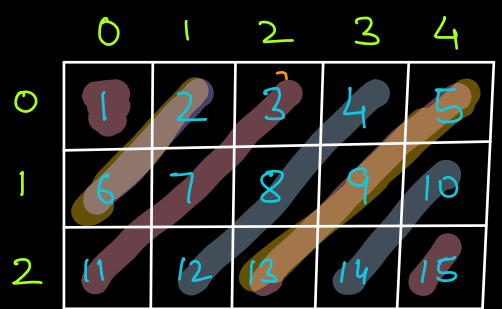
Row = n
Columns = m



no of diagonals $= (m + n - 1)$

Q4 Given `mat[N][M]`, print all diagonals going R-L.

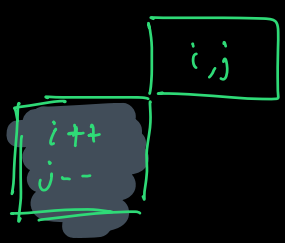
mat [3] [5]

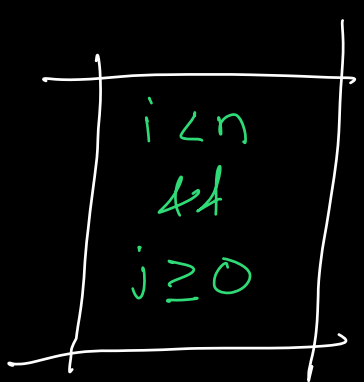


row	colum
0	4
1	3
2	2
3	1

row	colum
0	1
1	0
2	-1

[1]
[2 6]
[3, 7 11]
[4 8 12]
[5 9 13]
[10 14]
[15]





Terminating condition.

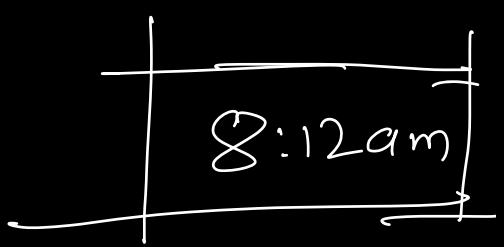
Pseudo code

```
i = 0
for (int j = 0 ; j < m ; j++) {
        int row = i
        int col = j
        while (row < n && col >= 0) {
                print mat[row][col];
                row++; col--;
        }
}

j = m - 1
for (int i = 1 ; i < n ; i++) {
        int row = i
        int col = j
        while (row < n && col >= 0) {
                print mat[row][col];
                row++; col--;
        }
}
```

Tc: O(mn)

Sc: O(1)



Qs Given a mat[N][N], find the transpose inplace. Given input mat[][] should get updated. No extra space has to be used.

SC: O(1)

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

Original

	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

Transpose

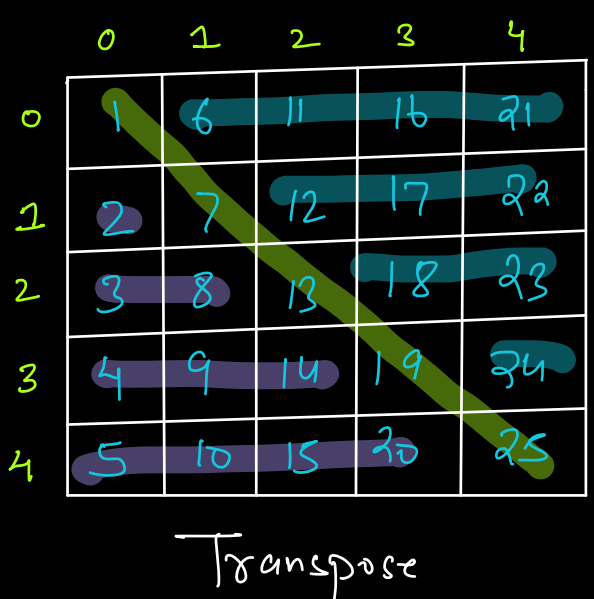
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Swap [mat[i][j] , mat[j][i]]

Pseudo Code :

```
for (int i=0 ; i<n ; i++) {
    for (int j=0 ; j<n ; j++) {
        if (i!=j)
            swap (mat[i][j], mat[j][i]);
    }
}
```

We get original array in the end.

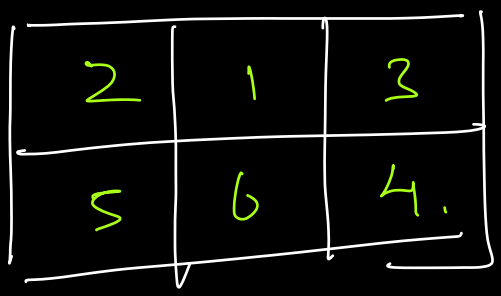


Transpose

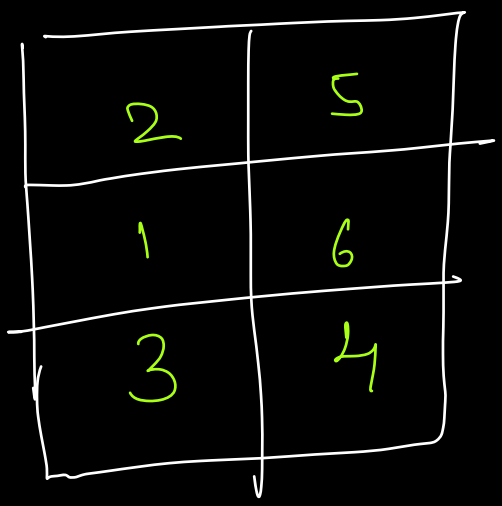
TODO:

Iterate on upper/lower triangle and swap.

What if the matrix is a rectangle.



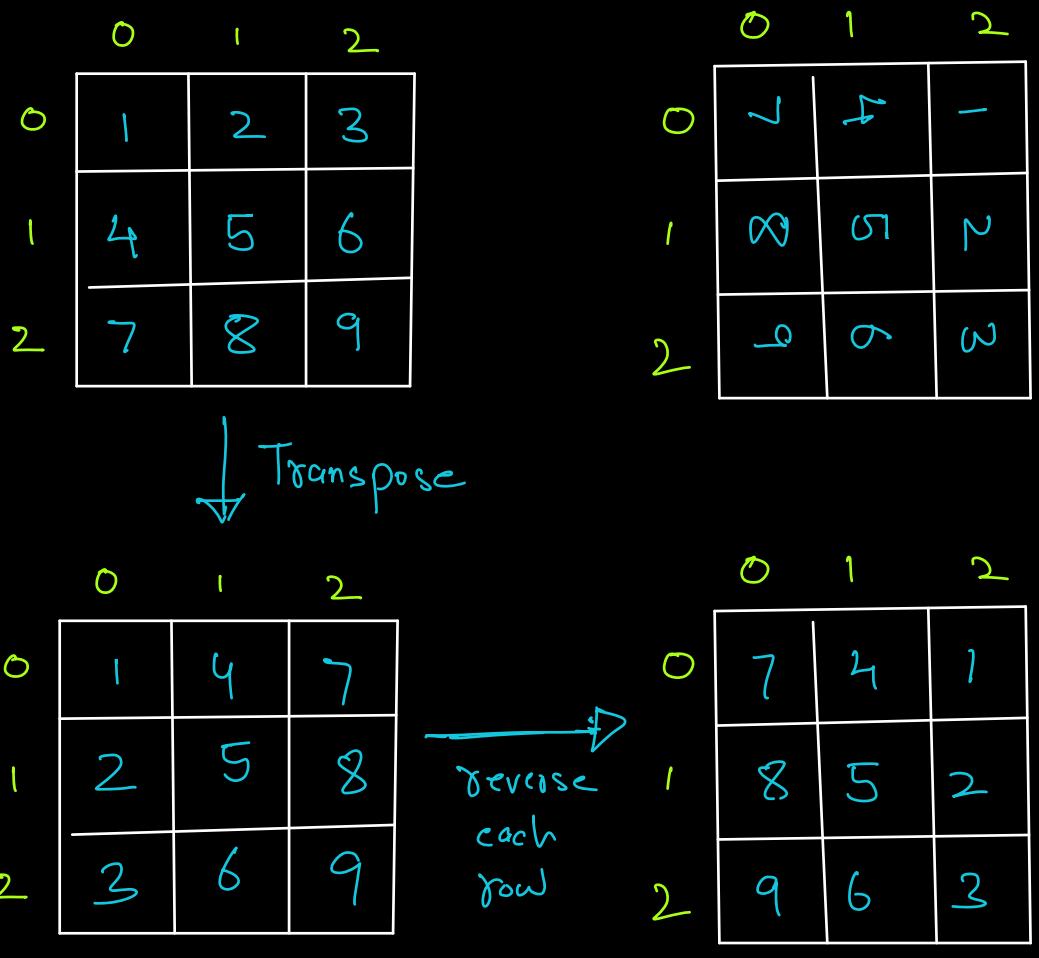
2 x 3



3 x 2

Cannot be solved in O(1) space.

Q6) Given a mat[N][N], rotate by 90° clockwise. Inplace. SC: O(1)



Step 1: Transpose : Tc: $O(n^2)$

Step 2: Reverse each row : Tc: $O(n^2)$

Tc: $O(n^2)$ Sc: O(1)

(Inplace)