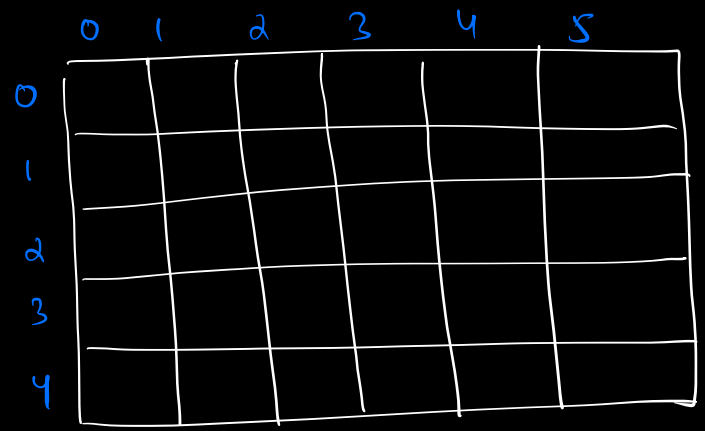
NOTE: Max Subarray Sum → TLE → (why) → next class.

int mat[5][6]

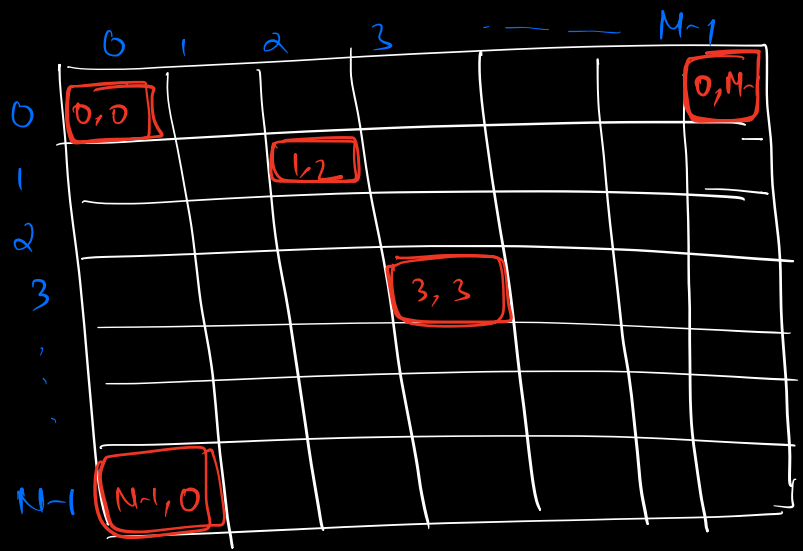
↓

2D array / matrix having 5 rows and 6 columns of type int.



int mat[N][M]

↓ rows ↓ columns



arr[i]

mat[r][c]

↓

mat[1][2]

mat[3][3]

mat[0][M-1]

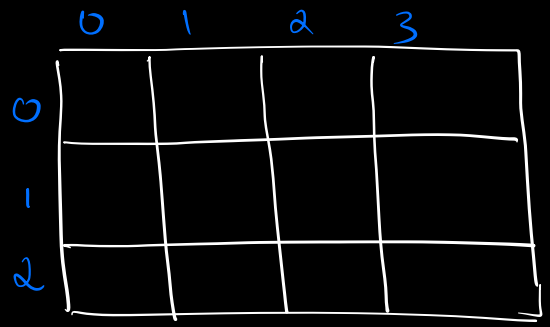
mat[0][0]

mat[N-1][0]

mat[3][4]

// iterate on a row:- [1]

```
for(j=0; j<4; j++){
    print(mat[1][j])
}
```



* iterate on a column [2]

```
for(i=0; i<3; i++) {
    print(mat[i][2])
}
```

{ i → row
  j → column

Q 1. 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

O/P => 22 12 42

pseudo

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

TC => O(N*M)
SC => O(1)

Q 2. Given a mat[N][M], find maxm column sum

mat[3][4] ⇒

	0	1	2	3
0	3	8	9	2
1	1	2	3	6
2	4	10	11	18
	↓ 8	↓ 20	↓ 23	↓ 26

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

O/P = 26

	0	1	2
0	10	11	12
1	-1	6	2
	↓ 9	↓ (17)	↓ 14

j	i		
0	0	10	9
0	1	-1	
1	0	11	17
1	1	6	
2	0	12	14
2	1	2	

Q 3. Given a matrix[N][N], print its diagonals [L-R, R-L] } Corner to corner only

mat[4][4] → 2d matrix

	0	1	2	3
0	0,0			0,3
1		1,1	1,2	
2		2,1	2,2	
3	3,0			3,3

// print L-R

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

TC ⇒ O(N)
SC ⇒ O(1)

// print R-L.

	i	j	
i++	0	3	j--
i++	1	2	j--
i++	2	1	j--
i++	3	0	j--

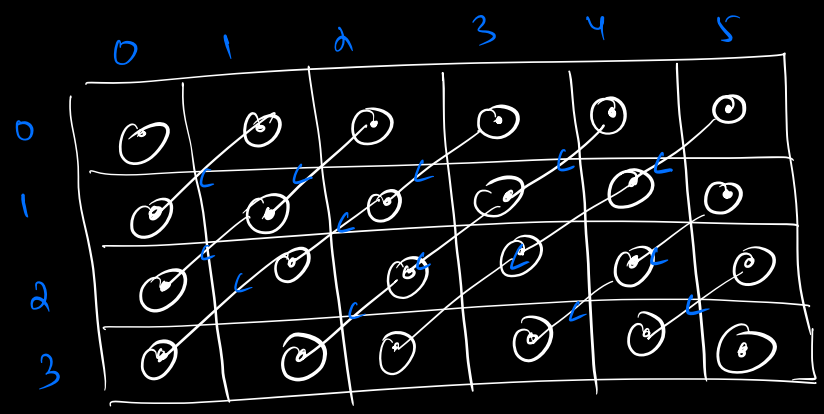
```
i=0; j=N-1;
while(i<N && j>=0) {
    print(mat[i][j])
    i++
    j--
}
```

TC ⇒ O(N)
SC ⇒ O(1)

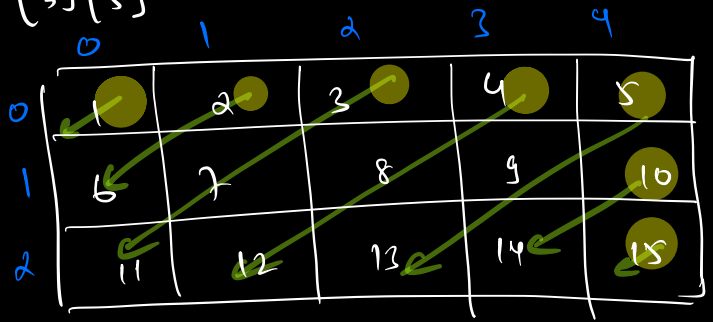
Overall; TC ⇒ O(N+N) ⇒ O(N)
SC ⇒ O(1)

Q4. Given a mat[N][M]. print all diagonals TR-BL R-L

mat[4][6]



ex ⇒ mat[3][5]



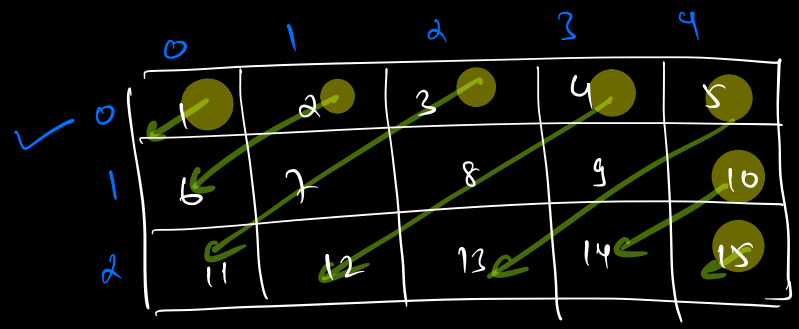
Op

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

Obs

1) diagonals either start from 0th row or M-1th colm

2) TR calls is applicable for both 0th & M-1th colm, take care, so no repetition occurs



0,0 0,1 0,2

1,0 1,1

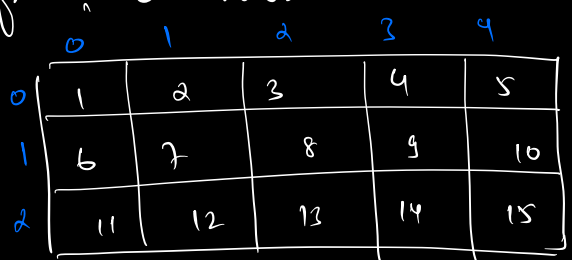
2,0

2,1

rows → [0 N-1]

col → [0 M-1]

S1 ⇒ print all diagonals from 0th row:-




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

S2 ⇒ print all diagonals from $(M-1)^{th}$ colm.

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

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15

O/P = 10 14 15

TC ⇒ O(N x M)

SC ⇒ O(1)

Q5. Given a matrix [N][N], find the transpose inplace.

* transpose ⇒ replace ⇒ rows → colm

// mat[3][3]

1	2	3
4	5	6
7	8	9

⇒ transpose

1	4	7
2	5	8
3	6	9

* inplace → SC O(1)
update the input mat.

mat[2][3]

1	2	3
4	5	6

2x3

⇒

1	4
2	5
3	6

3x2

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

→ transpose

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

Obs: 1) diagonals are same.

2) mat[0][1] ⇄ mat[1][0]

mat[3][2] ⇄ mat[2][3]

mat[i][j] ⇄ mat[j][i].

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

wrong

Eg ⇒ when reached ⇒ [0, 3]

swap ((0,3), (3,0))

when reached → [3,0]

swap ([3,0], [0,3])

either iterate on upper triangle & swap

or, " " " lower " " "

TO-DO ⇒ pseudo code

TC ⇒ $O(N^2)$

SC ⇒ $O(1)$

Q6. Given a sq. matrix, rotate it 90° in clockwise form IR as reference [in place].

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

IR

→ Rotate.

↓ transpose

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

→ reverse rows

same

	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

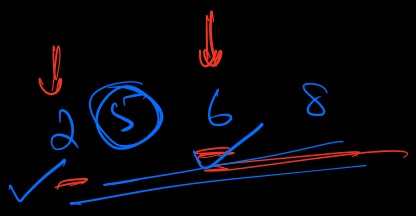
IR

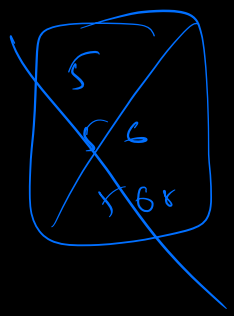
S1 ⇒ transpose → $O(N^2)$

S2 ⇒ reverse each row of transpose → $O(N^2)$

TC ⇒ $O(N^2)$

SC ⇒ $O(1)$





6
6 8
3 4 5

4
4 5

2
2 4
2 3 4
2 3 4 5

5 2 4 6 length