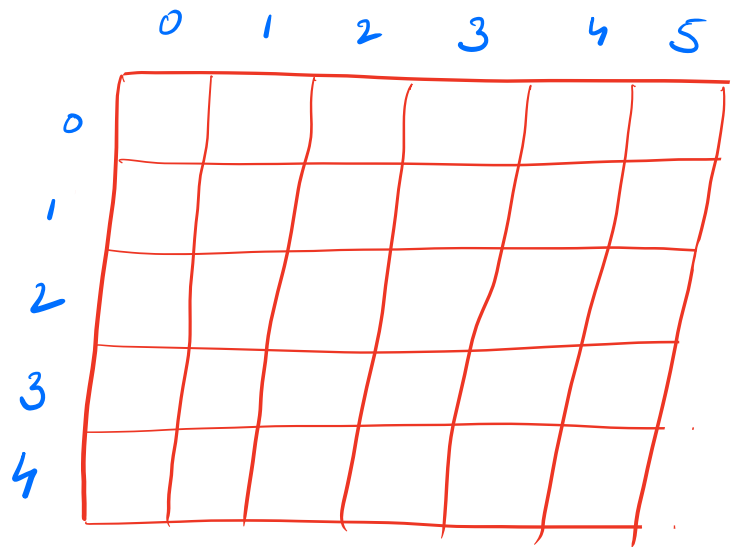


int arr[5][6]

↑     ↑  
rows cols

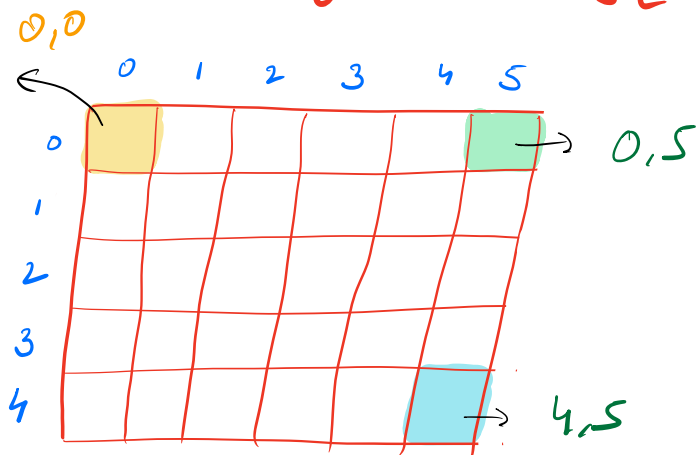
5x6 matrix



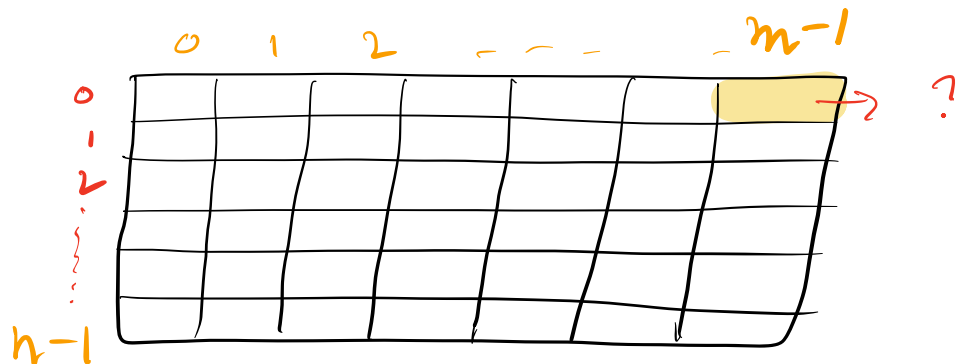
# How to access?

arr[i]     (1D case)

arr[row-no][col-no]     (2D)



NxM  
matrix



Q1 Given arr [N][M], print row-wise sum

Eg

	0	1	2	3	
0	1	2	3	4	→ 10
1	5	6	7	8	→ 26
2	9	1	1	2	→ 13

```
for (i = 0; i < n; i++) {  
    // at the ith row  
    int sum = 0  
    for (j = 0; j < m; j++) {  
        sum += a[i][j]  
    }  
    print(sum)  
}
```

TC:  $O(nm)$

SC:  $O(1)$

Q2 Given arr [N][M], find maximum column sum

Eg

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	1	1	2

↓ ↓ ↓ ↓

15 9 11 14

row  $\rightarrow i$   
col  $\rightarrow j$

ans = 15

```
int max_col_sum = INT_MIN
for (j=0 ; j<m ; j++ ) {
    // at jth col
    int sum = 0
    for (i=0 ; i<n ; i++ ) {
        sum += a[i][j]
    }
    max_col_sum = max (sum, max_col_sum)
}
return max_col_sum
```

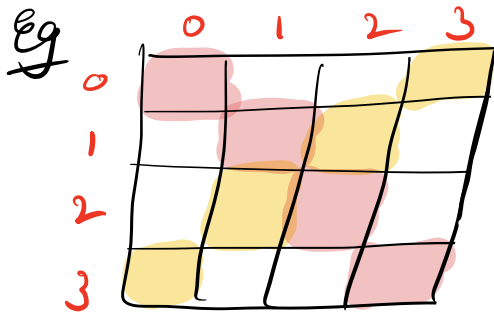
Tc:  $O(nm)$       Sc:  $O(1)$

- Why  $a[j][i]$  is not used generally?

$i \rightarrow$  row-no

$j \rightarrow$  col-no

Q3 Given arr[N][N], print diagonals



● → L-R  
● → R-L

L-R

a[0][0]  
a[1][1]  
a[2][2]  
a[3][3]

Obs: row\_no & col\_no same

```
for(i=0; i<n; i++) {  
    print(arr[i][i])  
}
```

OR

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

R-L

arr[0][3]

arr[1][2]

arr[2][1]

arr[3][0]

+1 { 0  
+1 { 1  
+1 { 2  
+1 { 3

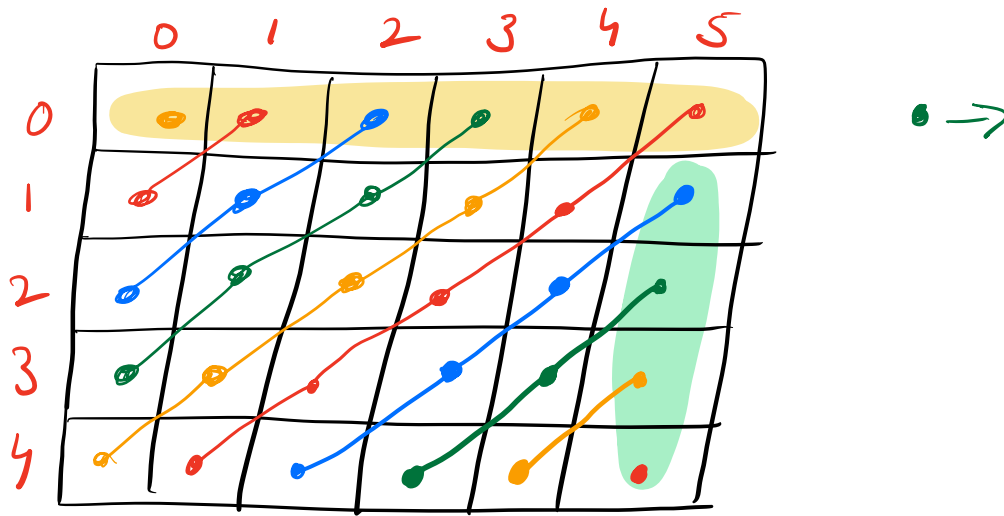
3 -1  
2 -1  
1 -1  
0 -1

```
int i = 0      j = n-1
while (i < n && j >= 0) {
    print (arr[i][j])
    i++
    j--
}
```

TC:  $O(n)$

SC:  $O(1)$

Q4 Given  $arr[N][M]$ , print all R-L diagonals



- 1) Print all diagonals starting at row 0
- 2) Print all diagonals starting at col  $m-1$   
(don't include 0,  $m-1$ )

for (j=0 ; j < M ; j++) d

int x = 0

int y = j

while (x < n && y > 0) d

print (a[n][y])

x++

y--

y

y

j=0

x=0 y=0

a[0][0]

x=1 y=-1

break

j=1

x=0 y=1

a[0][1]

x=1 y=0

a[1][0]

x=2 y=-1

break



$j=2$

$x=0$        $y=2$

$a[0][2]$

$x=1$        $y=1$

$a[1][1]$

$x=2$        $y=0$

$a[2][0]$

$x=3$        $y=-1$

*break*

```

for (i = 1 ; i < N ; i++) {
    int x = i          y = m-1
    while (x < n && y >= 0) {
        print (a[x][y])
        x++
        y--
    }
}

```

$i = 1$

$x = 1$        $y = 5$

1, 5

2, 4

3, 3

4, 2

5, 1

break

TC:  $O(nm)$

SC:  $O(1)$

---

$$i=2$$

$$x=2$$

$$y=5$$

2

5

3

4

4

3

5

2

y break

Q5 Given  $arr[N][N]$ , find the transpose of this matrix

inplace  $\rightarrow$  SC:  $O(1)$

$arr[N][N]$  needs to be updated

Transpose  $\Rightarrow i^{th}$  row  $\rightarrow i^{th}$  col

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

	0	1	2	3
0	1	5	9	13
1	2	6	10	14
2	3	7	11	15
3	4	8	12	16

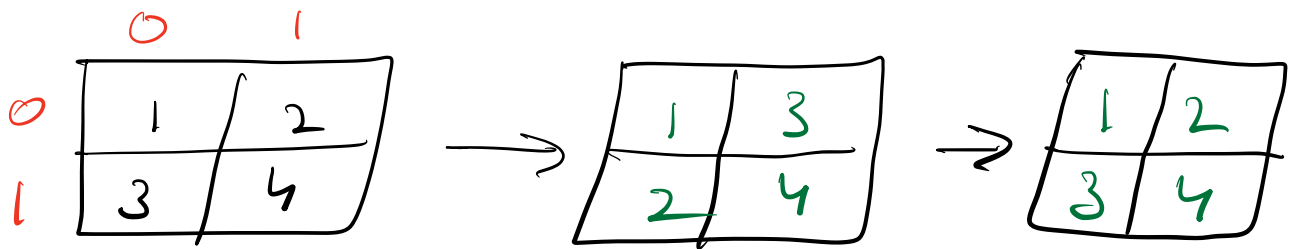
Pattern  $\Rightarrow arr[i][j] \Rightarrow arr[j][i]$

Solution  $\Rightarrow$  swap  $arr[i][j]$  with  $arr[j][i]$

```

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

```



How to avoid? only run on upper half.

```

for (i=0; i<N; i++) {
    for (j=i+1; j<N; j++) {
        swap(arr[i][j], arr[j][i])
    }
}

```


1 2 3      ⇒      1 4 7  
4 5 6      2 5 8  
7 8 9      3 6 9

TC:  $O(n^2)$

SC:  $O(1)$

Q6 Rotate by 90° clockwise


	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16



	0	1	2	3
0	13	9	5	1
1	14	10	6	2
2	15	11	7	3
3	16	12	8	4

Hint: first take transpose

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16



	0	1	2	3
0	1	5	9	13
1	2	6	10	14
2	3	7	11	15
3	4	8	12	16

What is the pattern now?

Ans: reverse every row.

$O(n^2)$   $\left[ \begin{array}{l} \text{Transpose (arr)} \\ \text{for } (i=0; i < n; i++) \{ \\ \quad \text{reverse (arr[i])} \\ \} \end{array} \right.$

TC:  $O(n^2)$  SC:  $O(1)$

{done}



—

—



