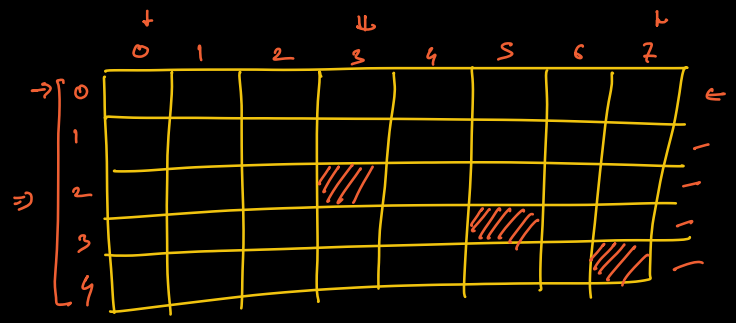
- Arrays
- Subarrays
- Prefix Sum
- Carry forward
- Sliding window
- Contribution tech.



=> 2D Matrix

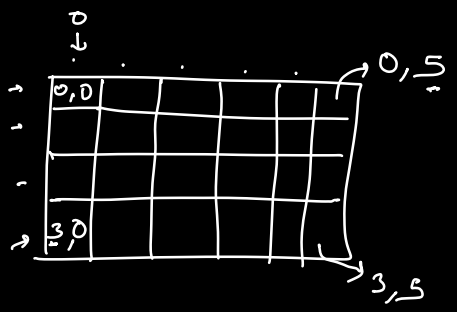
No. of cells = 5 x 8

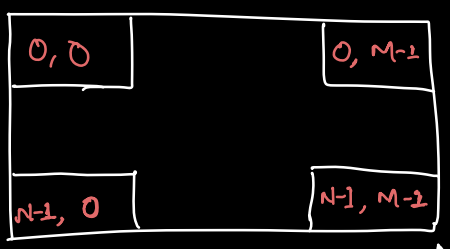
= 40

int mat [4] [6]

No. Rows [0-3]

No. of col [0-5]





NxM

Q Given a 2-D array of size N×M. Print all the values row-by-row.

```
for (i=0; i<N; i++) {
    for (j=0; j<M; j++) {
        Print (A[i][j]);
    }
}
```

TC: O(NM)

Q Given a matrix. Print sums of all the cols.

3	1	6	→
8	2	5	→
0	-1	3	→
↓	↓	↓	
11	2	14	

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

Q Given two 2-D matrics. Return the sum.

A : 3, 2, 8
1, 6, 9
7, 5, 4

B : -1, 2, 2
-2, 6, 1
3, 4, 8

A + B ⇒ (3-1) (2+2) (8+2)
(1-2) (6+6) (9+1)
(7+3) (5+4) (4+8)

Dimentions of the two matrics have to be the same for adelition/subtraction

```
[Sum[N][M] ⇐ New space
for(i=0; i<N; i++){
    for(j=0; j<M; j++){
        Sum[i][j] = A[i][j] + B[i][j];
    }
}
```

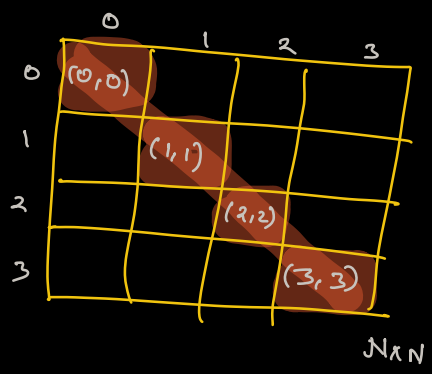
TC : O(N*M)
SC : O(N*M)

Space Complexity ⇒ Total Space used in the program

Input + Extra / Auxiliary space

Q Given a square matrix. Print all the diagonal elements. (L to R)

```
for(i=0; i<N; i++){
    Print(a[i][i]);
}
```



TC: O(N)

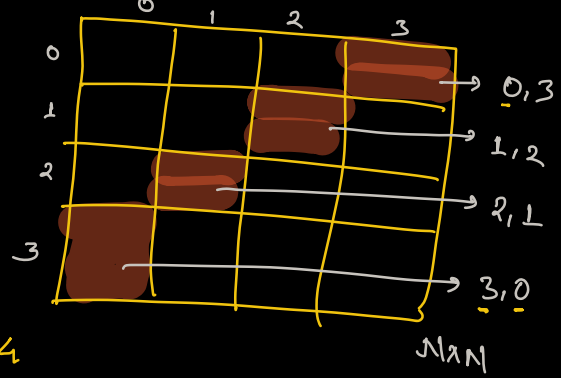
Diagonal (R-to-L)

Row 0 → (N-1)

Col N-1 → 0

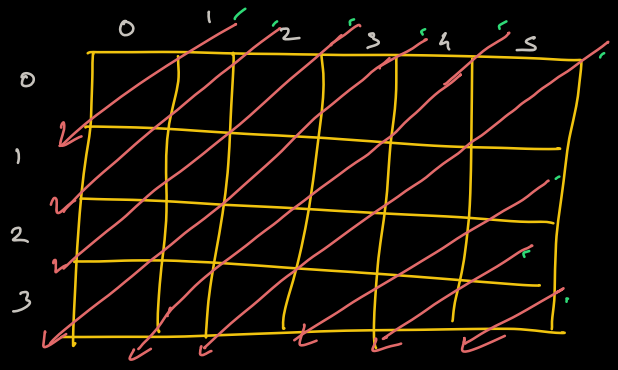
$i + j = N-1$

$\Rightarrow \boxed{j = N-i-1}$

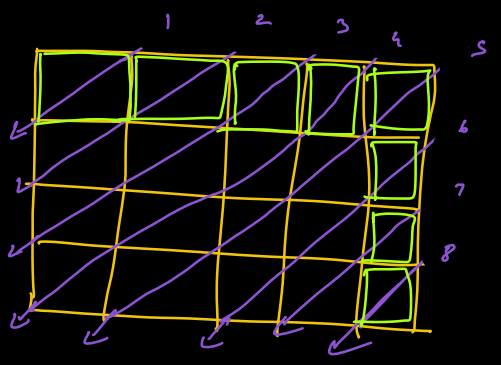


```
i = 0 ; j = N-1;
While ( i<N && j>=0 ){
    Print ( arr[i][j]);
    i++;
    j--;
}
```

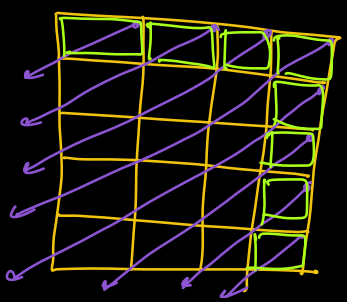
Q Diagonals in a rect. matrix

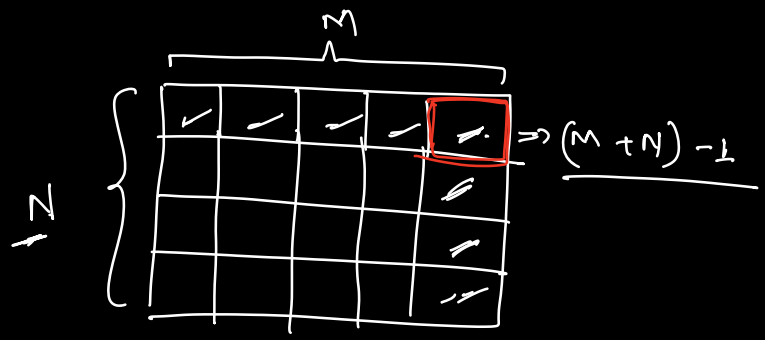


Quiz



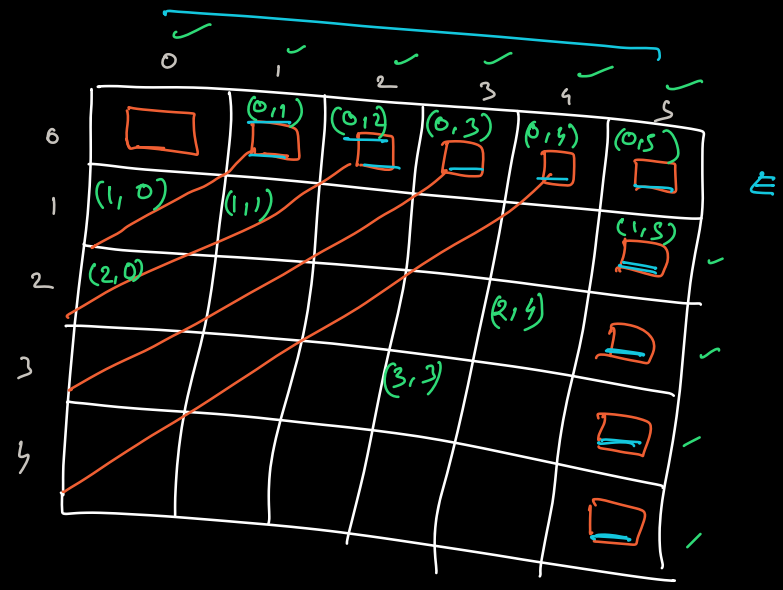
⇒ 8





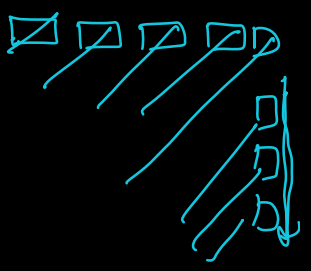
No of diagonals = N + M - 1

(R to L)
OR
(L to R)

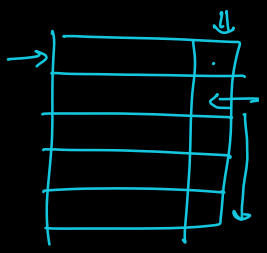


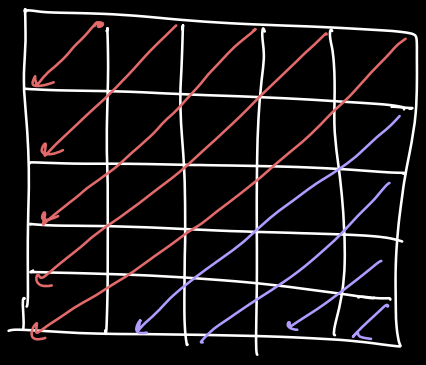

```
Print Diagonal ( Si, Sj, N, M) {
    i = Si;  j = Sj;
    while ( i < N && j >= 0) {
        Print ( a[i][j]);
        i++ ; j--;
    }
}
```

```
// for the first row
for (j=0 ; j<M ; j++) {
        PrintDiagonal(0, j, N, M);
}
```



```
// for the last col.
for (i=1; i<N ;  i++  ) {
        PrintDiagonal (i, M-1, N, M);
}
```





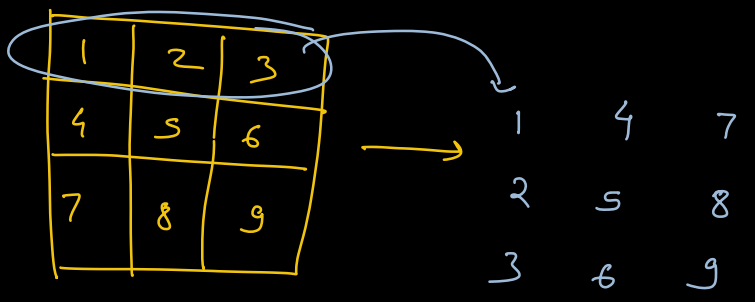
N, M

Total No of cells = N*M

TC = O(N*M)

Break till 10:35 p

Q Given a square matrix. Convert it to transpose.
Without using any extra space.

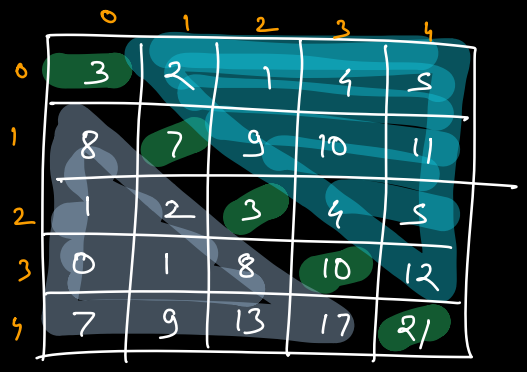


1, 4, 6
3, 7, 2
8, 9, 5

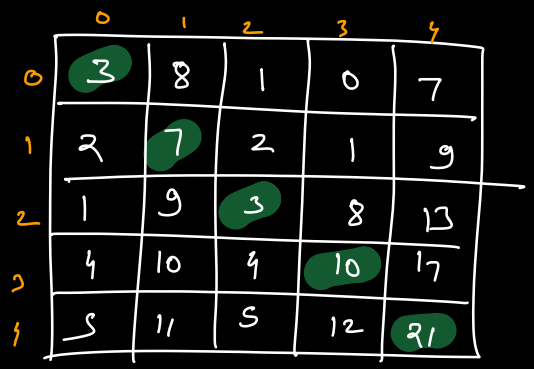
→

1 3 8
4 7 9
6 2 5

0th row → 0th col
1st row → 1st col
⋮
(N-1) row → (N-1) col



	0	1	2	3	4
0	3	2	1	4	5
1	8	7	9	10	11
2	1	2	3	4	5
3	0	1	8	10	12
4	7	9	13	17	21



	0	1	2	3	4
0	3	8	1	0	7
1	2	7	2	1	9
2	1	9	3	8	13
3	4	10	4	10	17
4	5	11	5	12	21

(0,0) ⟶ (0,0)

(0,1) ⟶ (1,0)

(0,2) ⟶ (2,0)

① $i, j \longrightarrow j, i$

② Diagonal elements remain in original position.

```
for(i=0; i< N; i++) {
    for( j=i+1; j<N; j++) {
        //Swap A[i][j] with A[j][i]
    }
}
```

i	j
0	3
3	0

TC : $O(N^2)$

SC (Extra) : $O(1)$

HW : Create transpose by iterating over the lower triangle.

Q Given a square matrix. (N*N)

Amazon MS Adobe

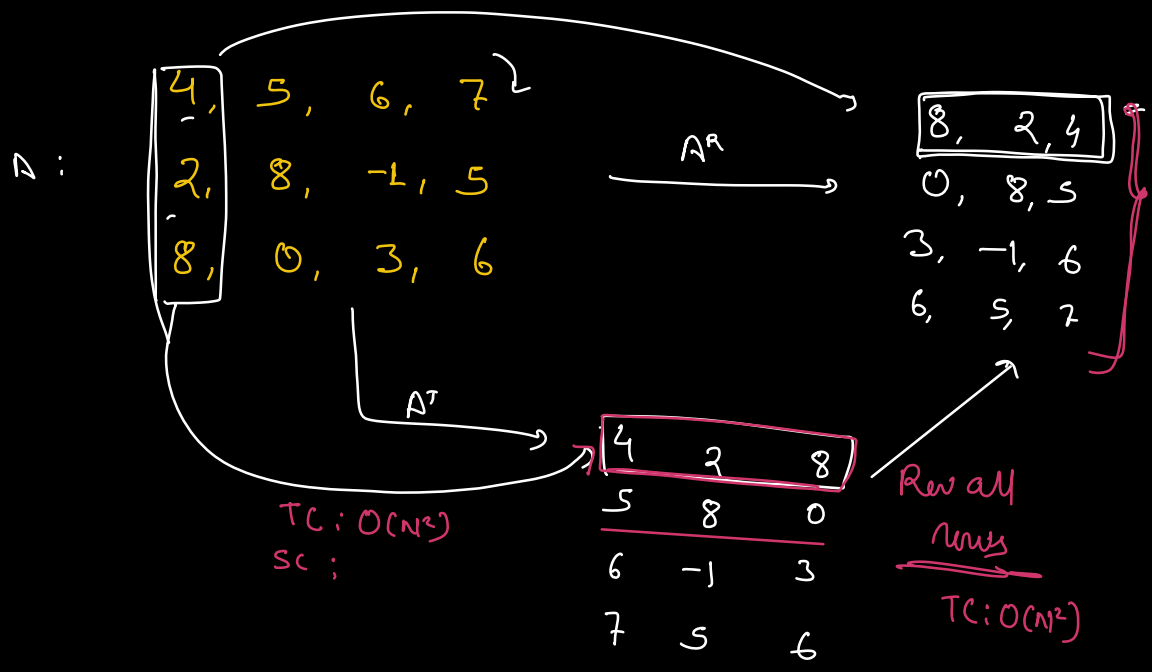
Rotate it by 90° in clockwise direction. [Without extra space]

90°

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

⟹

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



→ Write matrix on a paper

→ Rotate the paper by 90° Clockwise.

TC : $\underbrace{O(N^2)}_{A^T} + \underbrace{O(N^2)}_{\text{Rev of rows}} \Rightarrow O(N^2)$

SC : $\underbrace{O(1)}_{\text{(Extra)}}$

OSC = $O(N^2)$

① $A \longrightarrow A^T$ $\Rightarrow O(N^2)$ | $O(1)$

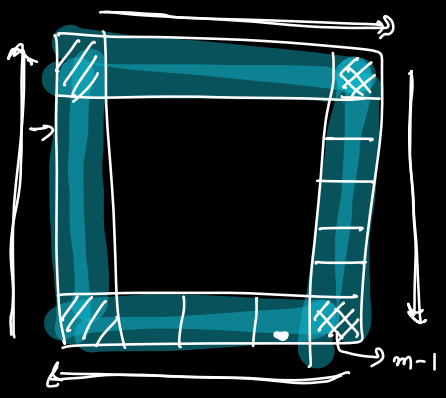
② Rev all rows of A^T $\Rightarrow O(N^2)$ | $O(1)$

$\Downarrow$

$TC : O(N^2)$

Mat $\longrightarrow$ N x M

A : N x M $\qquad$ $A^T \Rightarrow$ M x N



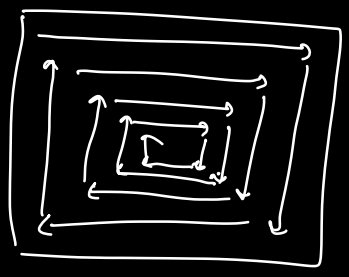
① $r \to 0$, fn ($j = 0 \to M$)

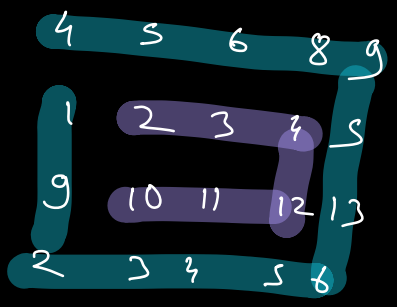
② $c \to M-1$ fn ($i = 1 \to N$)

③ $r \to N-1$ fn ($j = m-2 \to 0$)

④ $c \to 0$ fn ($i = N-2 \to 1$)

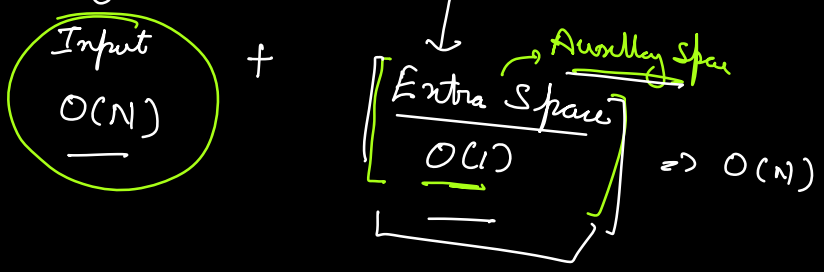
HW: Print Matrix in Spiral order





Space Complexity

→ Total Space used by the program



int Max(A[]){

}

SC of Binary Search : O(1)

↑ Extra Space