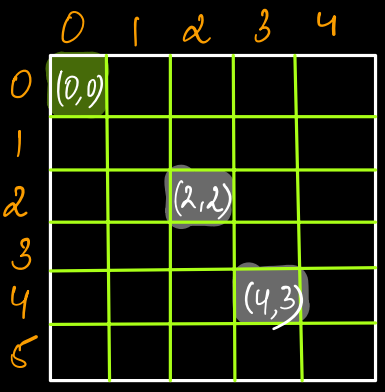
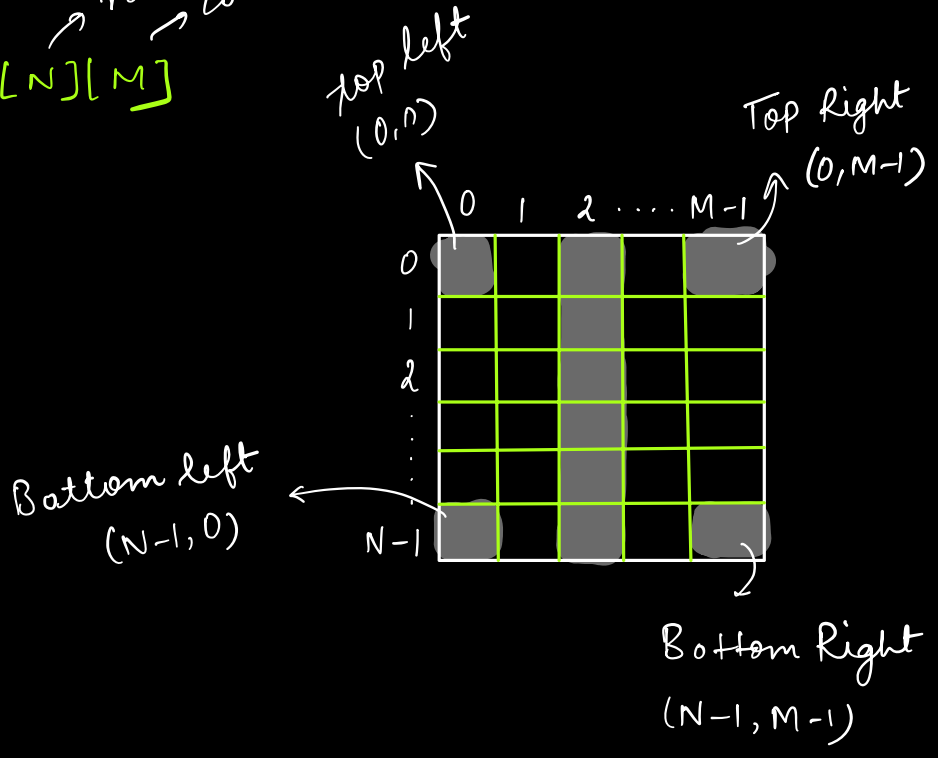
Today's Agenda :-

2D Array

```
int mat[6][5]
    |     |
  rows  columns
```




```
int mat[N][M]
    |     |
  rows  columns
```



$i^{th} \rightarrow arr[i]$

$(i, j) \rightarrow mat[i][j]$

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

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

(row, col)

O/P

10
26
42

Pseudo Code:-

```
for(int row = 0; row < N; row++){
    int sum = 0
    for(int col = 0; col < M; col++){
        sum = sum + mat[row][col]
    }
    print(sum)
}
```

Q2) Given mat[N][M]. print the column-wise sum.

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

O/P: 15 18 21 24

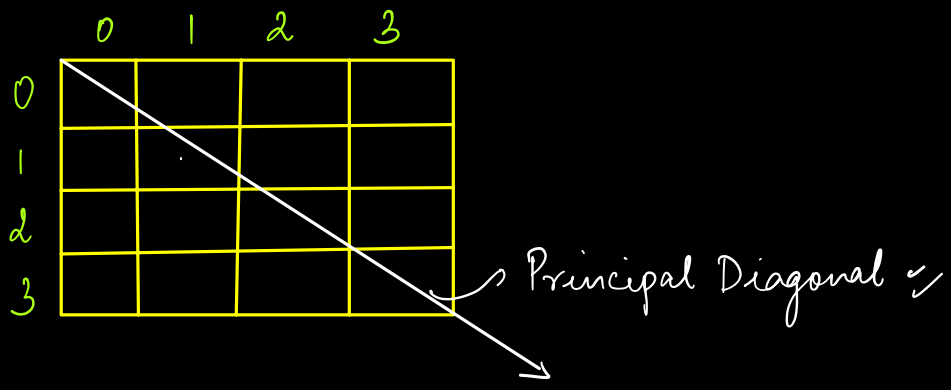
```
for(int col = 0; col < M; col++){
    int sum = 0
    for(int row = 0; row < N; row++){
        sum = sum + mat[row][col]
    }
    print(sum)
}
```

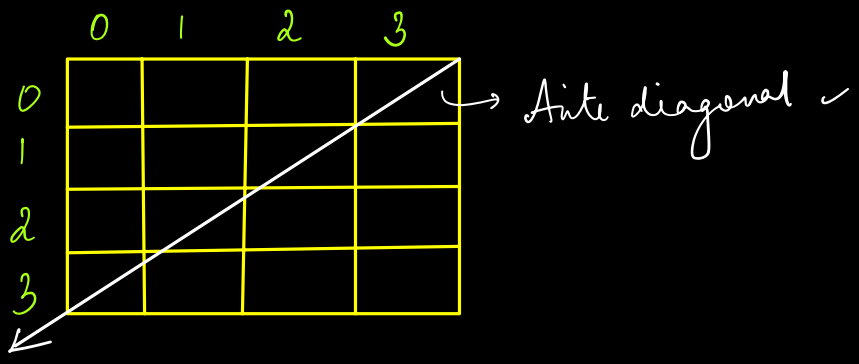
TC : O(N*M)

SC : O(1)

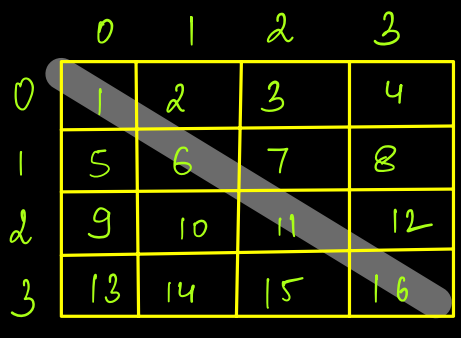
No of main Diagonals in a square matrix ?

↳ Rows = cols (N = M)





Q3) Print principal diagonal ? mat[N][N]



1 6 11 16

(i = j)

0,0 → (1,1) → (2,2) → (3,3)

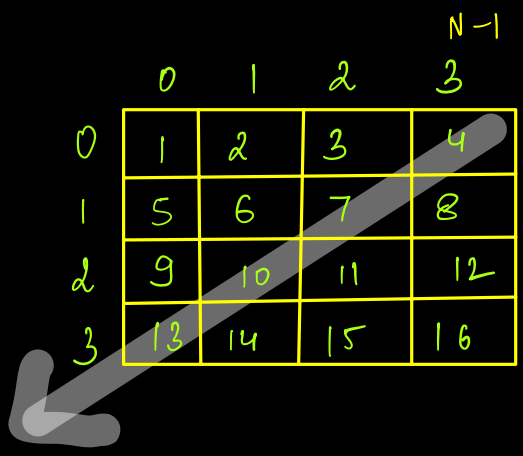
No of rows | col

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

TC : O(N)

SC : O(1)

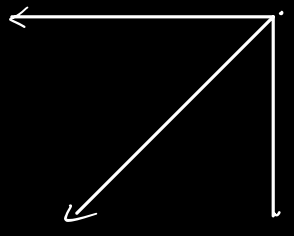
Q4) Print Anti-diagonal ? mat[N][N]



4	7	10	13

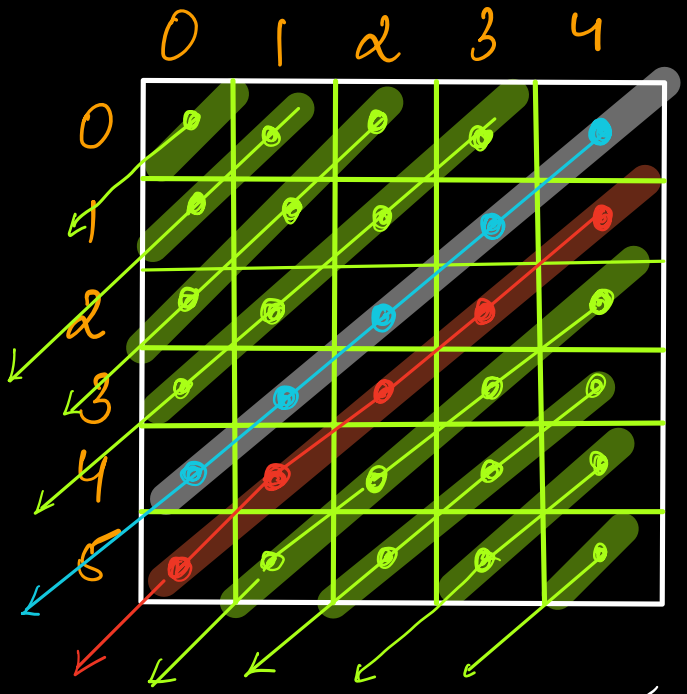
```
i   j
0   3
1   2
2   1
3   0
4   -1
```

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



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

Q5) Given mat[N][M]. Print all the diagonal elements from right to left

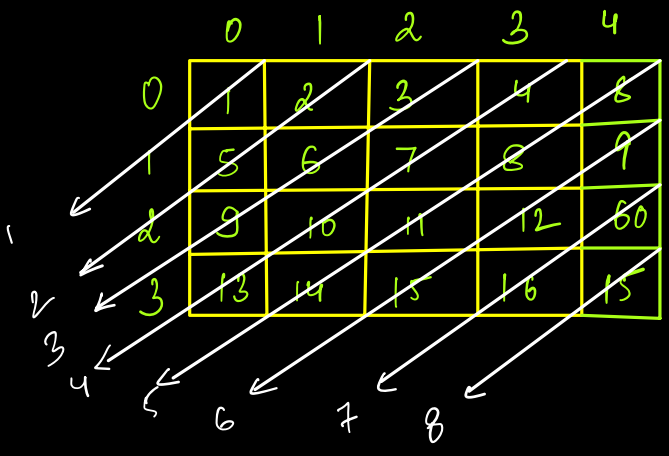


→ (6 * 5) ⇒ 5 + 6 − 1 = 10

diagonals = 10

→ 4 + 5 − 1 = 8

→ (4 * 5) # diagonals = 8



O/P

1

2 5

3 6 9

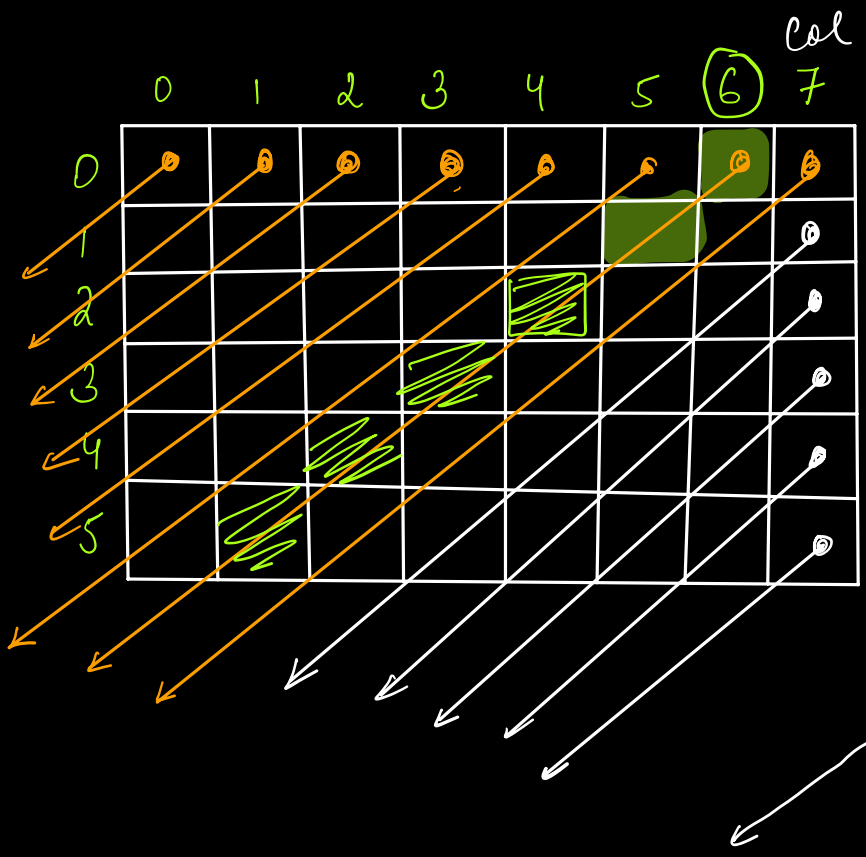
4 7 10 13

8 8 11 14

9 12 15

60 16

15



col

$8 + 5 = 13$

$M + (N-1)$ diagonals

6 * 8
rows col

diagonals from R-L =

$(N+M-1)$

(i++, j--)

i < N && j >= 0

(0,6) → (1,5) → (2,4)
↓
(6,0) ← (5,1) ← (4,2) ← (3,3)

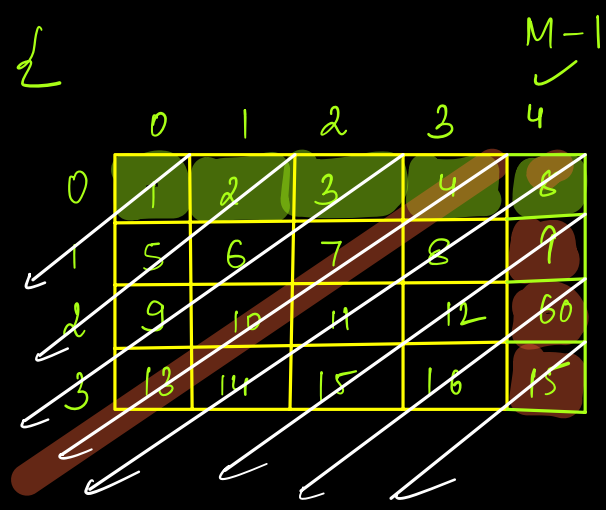
row col

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

Implement 2D dynamic array ✓

Try to solve today's problems

```
for(int col = 0; col < M; col++) {
    //(row, col) is start of a diagonal
    int i = 0 , j = col
    while( i < N && j >= 0 ) {
        print(mat[i][j])
        i++
        j--
    }
}
```



4 7 10 13

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

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

Q6) Given a square 2D matrix mat[N][N]. Find the transpose ?

What is Transpose ?

→ Transpose of a matrix is a new matrix obtained by interchanging the rows & columns of the original 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

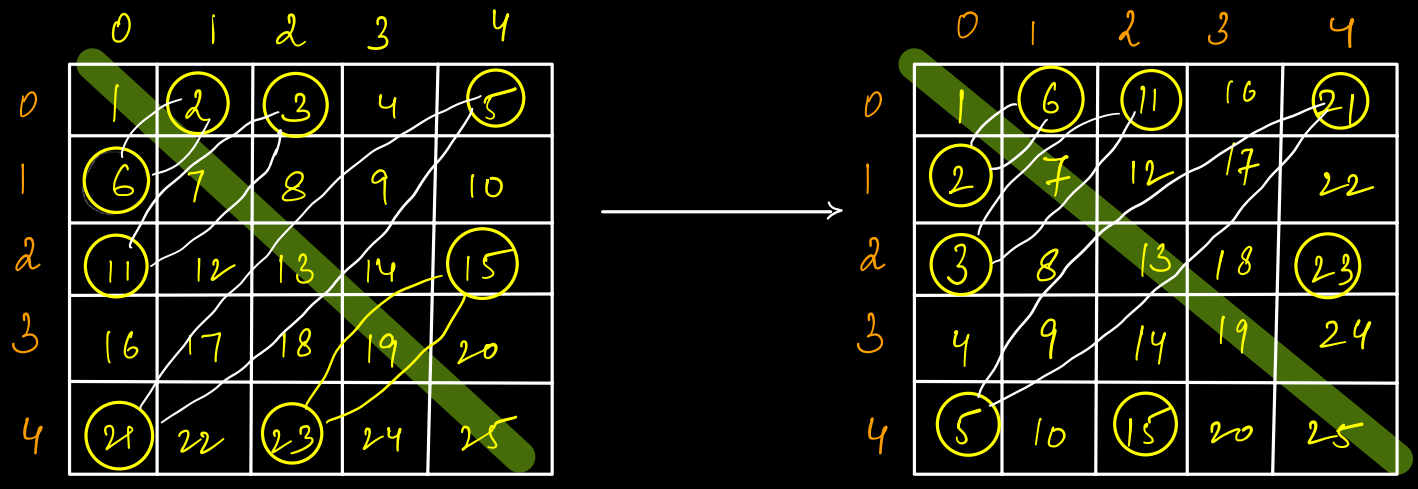
→

	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

	0	1	2	3	4
0	1	2	3	4	8
1	5	6	7	8	9
2	9	10	11	12	60
3	13	14	15	16	15

x → 1, 2, 3, 4, 8

extra space ✓




```
swap(mat[1][0], mat[0][1])
swap(mat[2][0], mat[0][2])
swap(mat[4][0], mat[0][4])
swap(mat[4][2], mat[2][4])
```

```
swap(mat[i][j], mat[j][i])
```

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

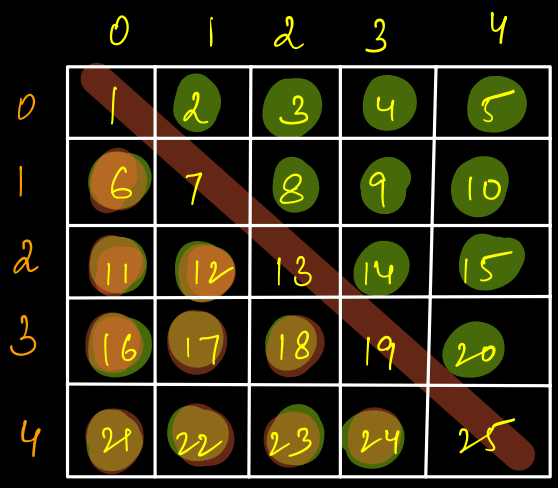
i = 0, j = 3

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

i = 3, j = 0

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

O|P won't even change



iterate only in upper triangle $(i < j)$

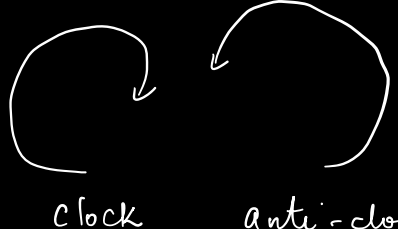
OR

iterate only in lower $(i > j)$

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

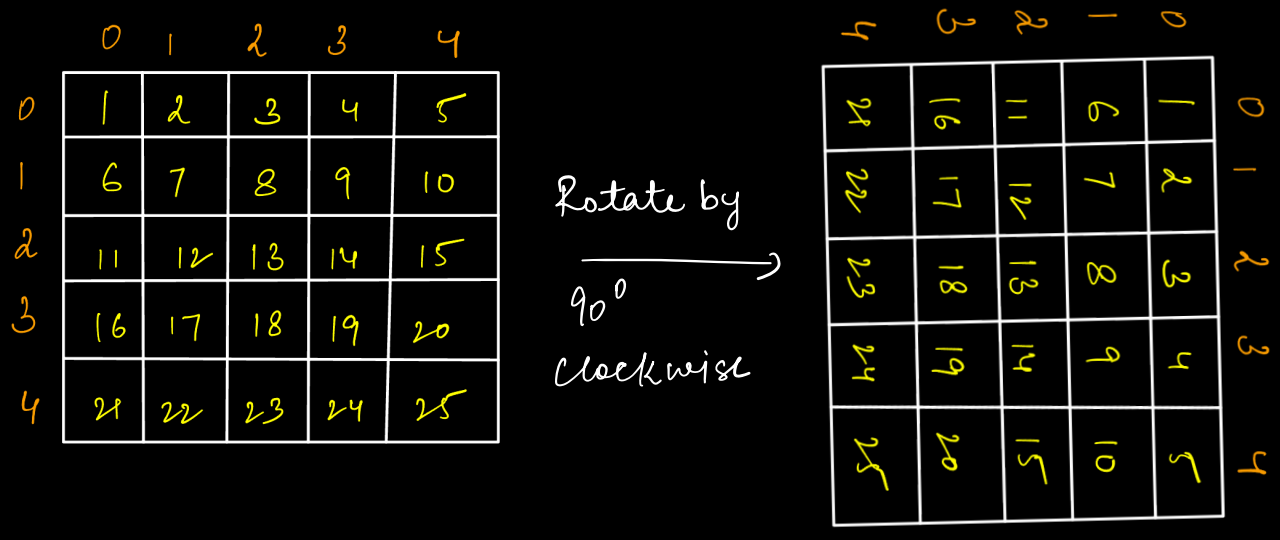
TC : $O(N^2)$
SC : $O(1)$

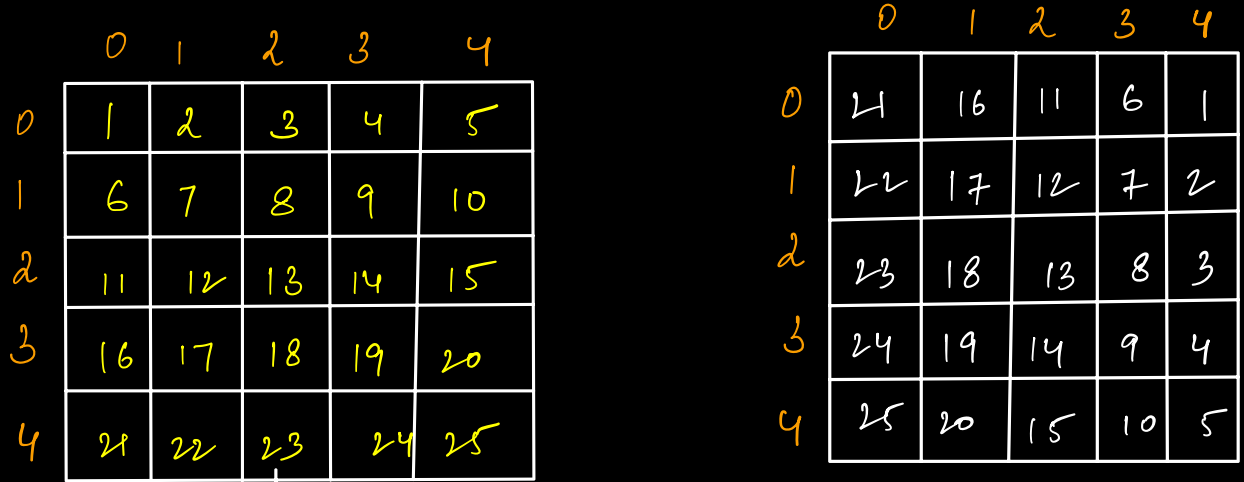
→ Interview Problem

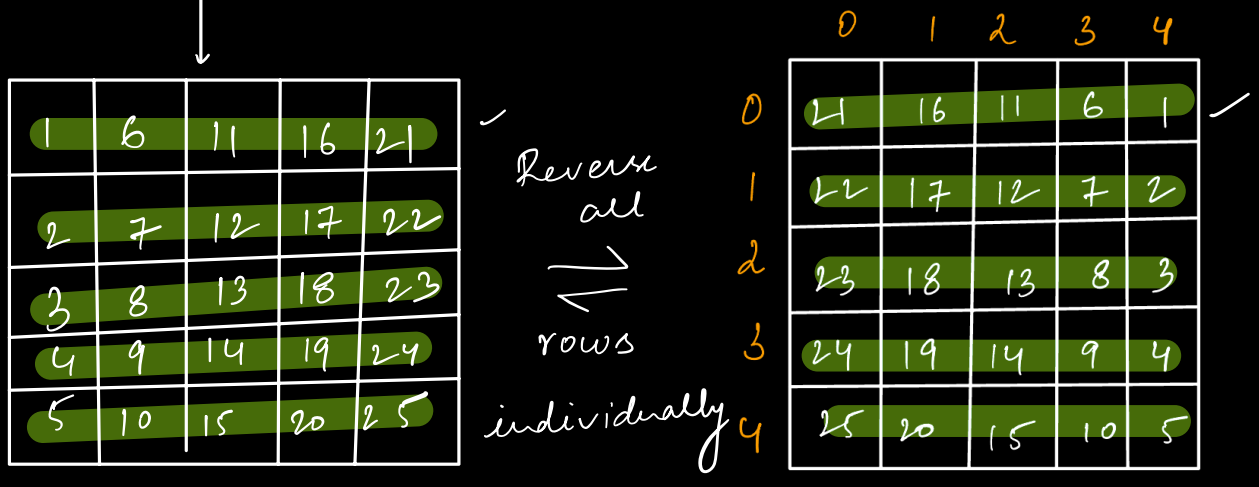


clock anti-clock

Q 7) Given a square matrix mat[N][N]. Rotate by 90° clockwise from top right as a reference, in place. ✓



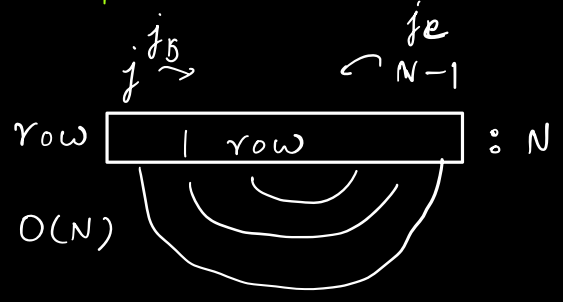




Step 1: Find transpose → N^2

Step 2: Reverse each row → N^2

} Matrix will rotate by 90° clockwise



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

```
reverseRow(int row) {
    int sc = 0, se = M-1
    while (sc < se) {
        swap(mat[row][se],
             mat[row][de])
        sc++
        se--
    }
}
```