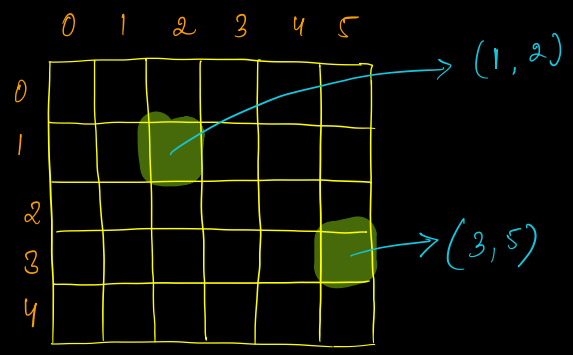
```
int mat[5][6]
```

rows

columns

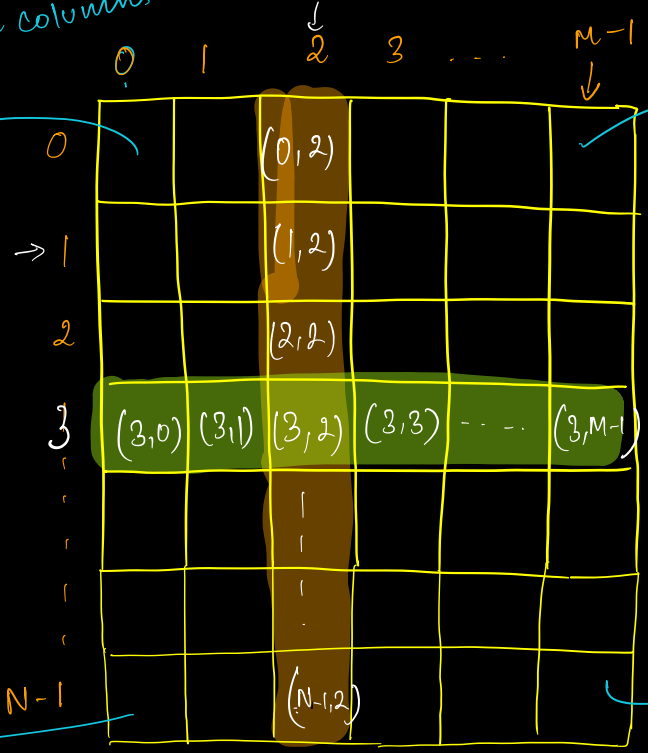



```
int mat[N][M]
```

N rows

M columns

Top left
mat[0][0]



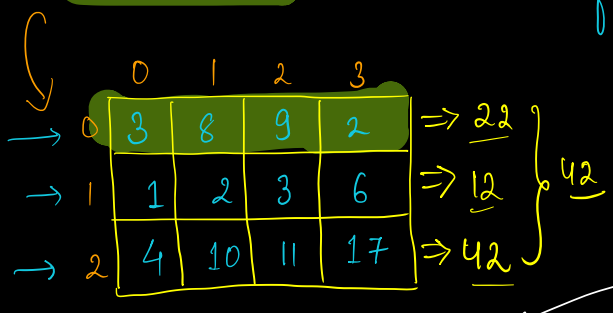
Top Right
mat[0][M-1]

Bottom left
mat[N-1][0]

Bottom Right
mat[N-1][M-1]

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

mat[3][4]

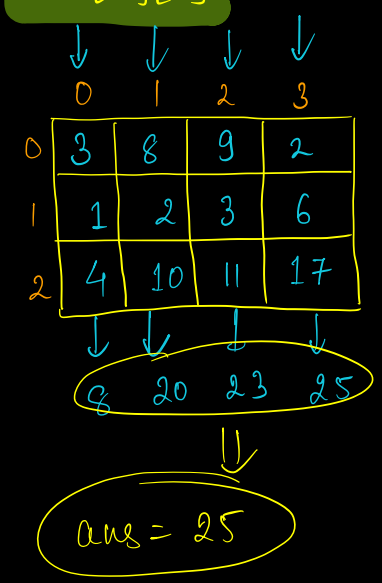



```
maxsum = INT_MIN.
for(i=0 ; i < N ; i++){
    sum = 0
    for(j=0; j< M; j++){
        sum = sum + mat[i][j];
    }
    print(sum);
    maxSum = max(maxsum, sum);
}
return maxsum
```

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

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

mat[3][4]




```
int maxsum = INT_MIN.
for(j=0 ; j< M; j++){
    sum =0
    for(i=0; i<N; i++){
        sum = sum + mat[i][j];
    }
    maxsum = max(maxsum, sum).
}
return maxsum
```

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

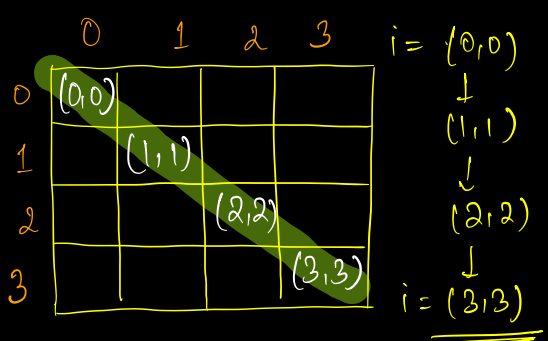
Q3) Given a mat[N][N], Print diagonals.

↳ Square matrix

$i \in [0, N-1]$

$\Rightarrow N-1-0+1 \Rightarrow N$

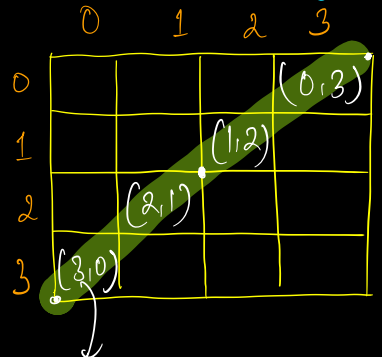
mat[4][4]



i = (0,0) → (1,1) → (2,2) → (3,3)

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

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



(0,3) → (1,2) → (2,1) → (3,0) → (4,-1)

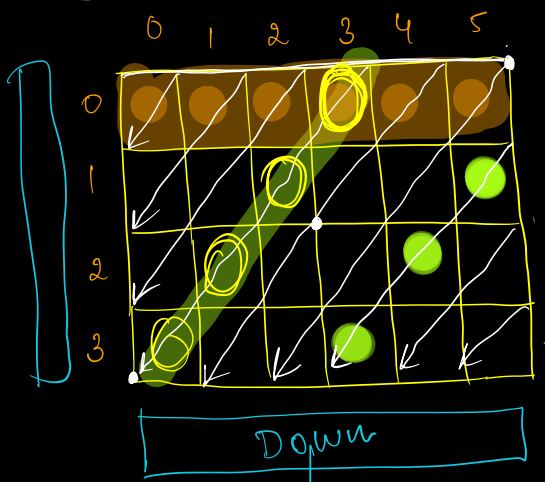
$i=0,\ j=N-1$ → any 1 condition is enough

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

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

Q4) Given mat[N][M], Print all diagonals going R-L.

mat[4][6]



(0,3) → (1,2) → (2,1) → (3,0) → (4,-1) (i+1, j-1)

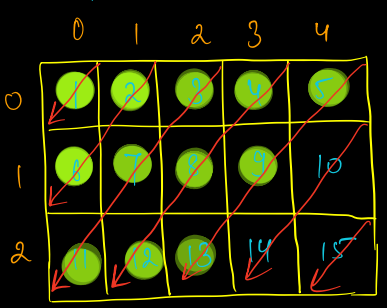
(1,5) → (2,4) → (3,3) → (4,2)

(4,2) → invalid / valid

x	y
0	2

mat[3][5]

Input



Output

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

1

→ 2 6 /n

→ 3 7 11

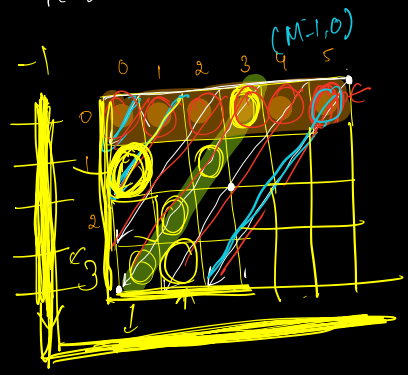
4 8 12

5 9 13

Pseudo Code

Step 1 : Print diagonals starting from 0th row.

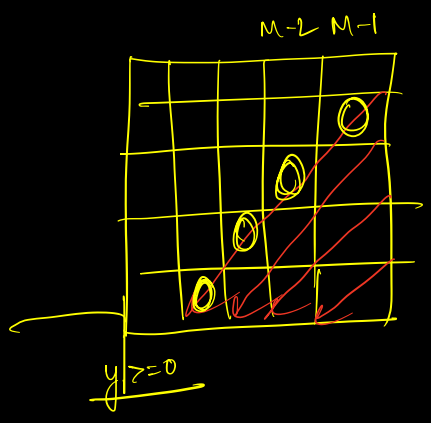
```
for(j=0; j<M; j++){
    (0,0)
    x=0, y=j // (x,y) = start of a diagonal.
    while(x<=N && y>=0){ // checking for boundary.
        print(mat[x][y]);
        x++;
        y--;
    }
    print("\n");
}
```



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

Step 2 Print all diagonals starting from M-1 col.

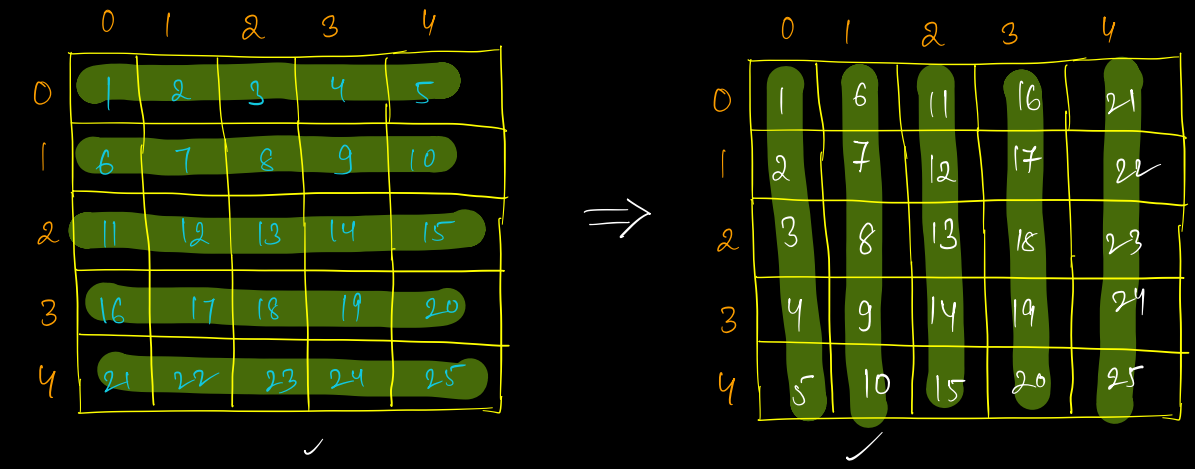
```
for(i=1; i<N; i++){
    x=i, y=M-1 //start of diagonal
    while(x<=N && y>=0){
        print(mat[x][y]);
        x++;
        y--;
    }
    print("\n");
}
```

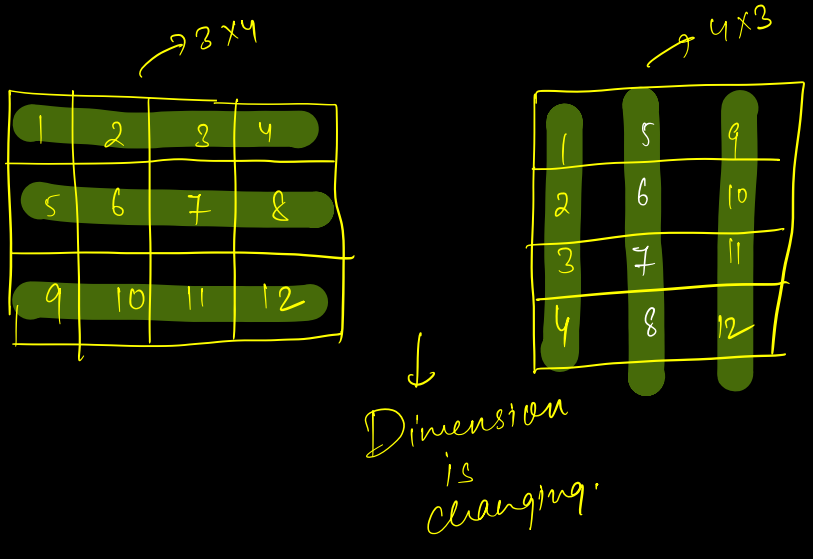


Q5) Given a mat[N][N], find the transpose inplace.

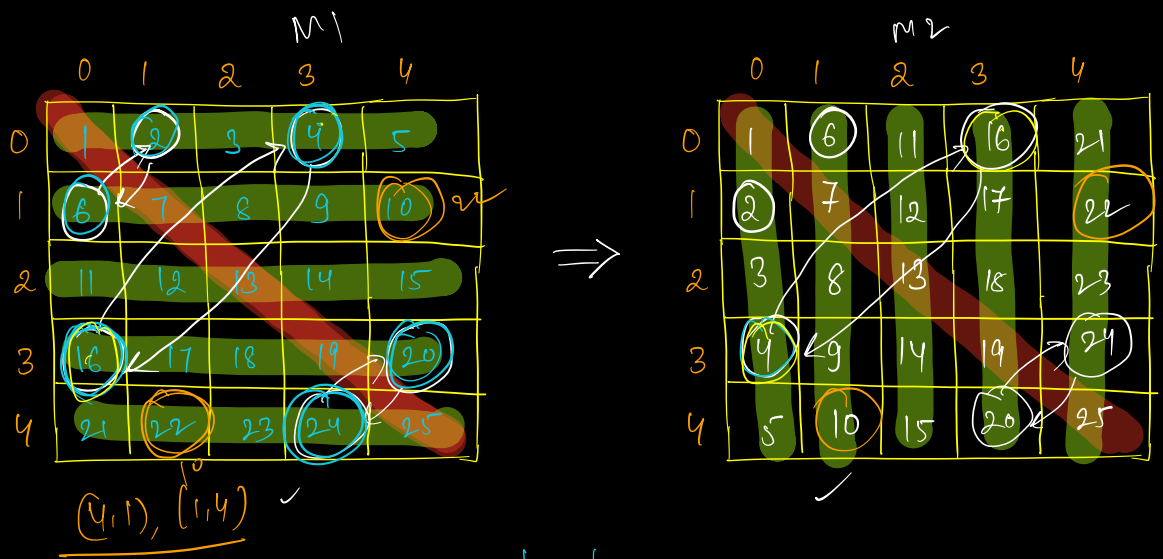
(Given input mat[][] should update)

mat[5][5] → final mat[5][5]





↓ Dimension is changing.



swap (mat[0][1] , mat[1][0])

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

swap(mat[3][4], mat[4][3]);

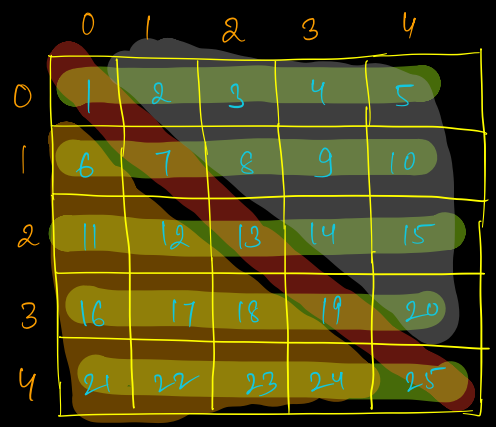
→ (swap (mat[i][j], mat[j][i]);)

→ When we reach (0,3)
swap (mat[0][3], m[3][0]);

→ When we reach (3,0)
swap (mat[3][0], m[0][3])

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

→ Matrix is not changing.



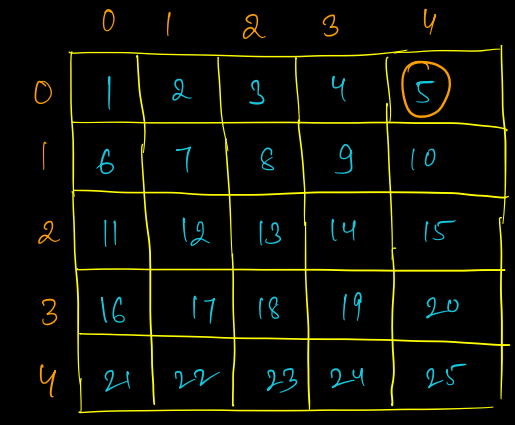
→ iterate in upper triangle & do swap.
↳ TODO

TC : O(N²)
SC : O(1)

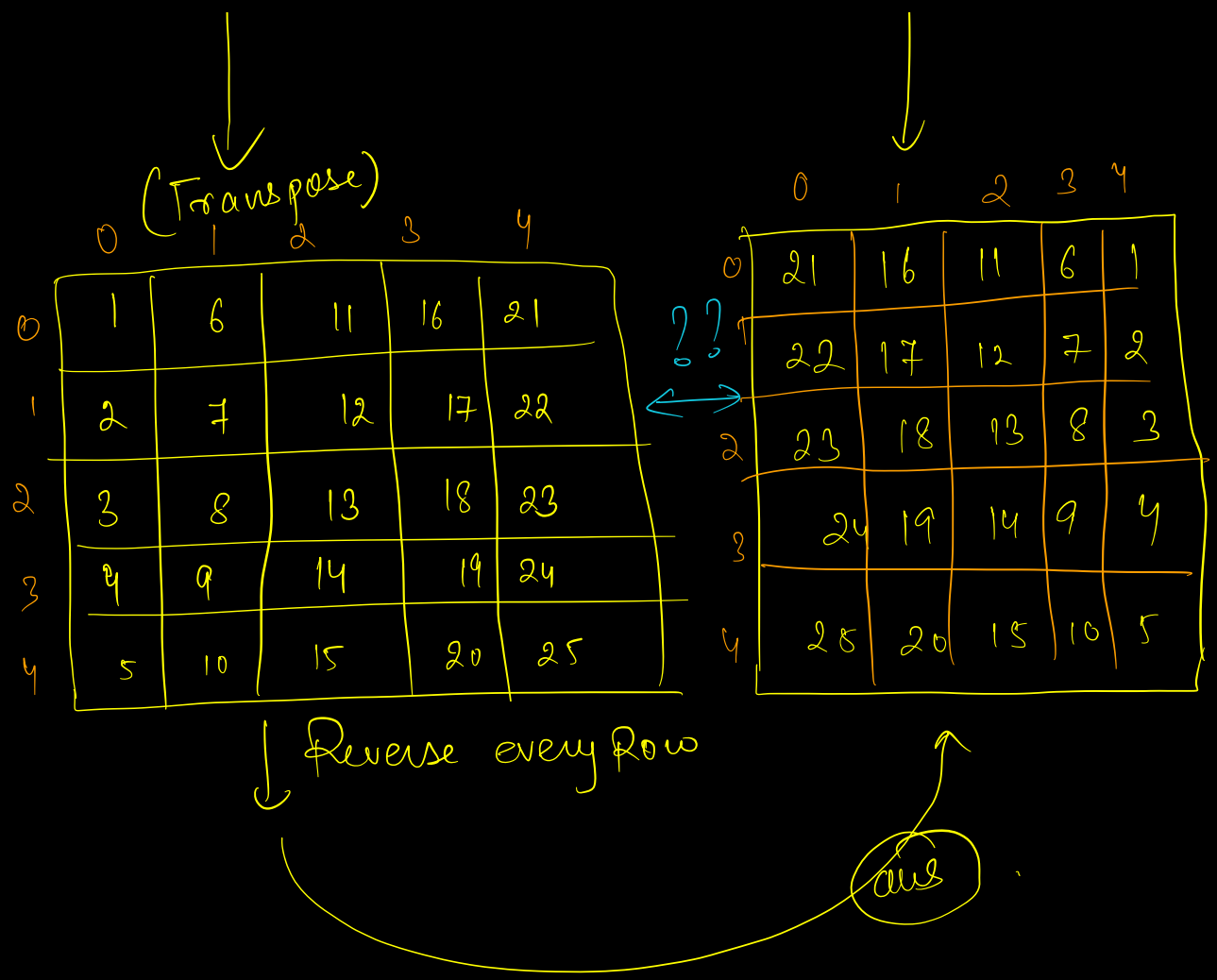
↓ iterate in lower triangle & do swap
↳ TODO

Q6) Given a square matrix. Rotate by 90° in clockwise from Top Right as a reference, inplace.

mat[5][5]



final mat[5][5]



Steps

1) find transpose
2) Reverse each Row

} Rotate 90° using TR as Ref.

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