2D Matrix

→ Arrays
→ Prefix Sum
→ Carry forward
→ Sliding Window
→ Contribution Techniques
→ Subarrays

2D Matrix Basics

int mat[4][6]

rows Columns

	0	1	2	3	4	5
0						
1						
2					(2,4)	
3			(3,2)			

Number starting from index 0

Access [0 based Indexing]

Row	Column	Data
2nd	5th	A[2][5]
0th	3rd	A[0][3]

1Q Given Mat [N] [M] print all elements row by row.

N rows & M Columns

[0, N-1] [0, M-1]

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

Iterations : N*M

2Q

	0	1	2	3	4	
0	3	-1	2	6	1	: 11
1	3	2	1	7	-3	: 10
2	6	-3	4	-3	-2	: 2
3	10	2	3	-7	3	: 11

Sum row by row

// When we iterate on a particular row : Row number Same

// When we iterate on a particular col : Column nuber Same

Q) Print row wise sum for a matrix [N] [M]

Mat [N] [M]

```
i = 0; i < N; i++) {
    // Get i^th row sum
    s = 0;
    j = 0; j < M; j++) {
        s = s + mat[i][j]
    }
    print(s) // i^th row sum
}
```

```
j = 0; j < M; j++) {
    // Get j^th column sum
    s = 0;
    i = 0; i < N; i++) {
        s = s + mat[i][j]
    }
    print(s) // j^th column sum
}
```

Arithmetic operation on Matrices

Add

mat1 [3][3]

3	-1	6
5	2	6
10	11	12

⟺

mat2[3][2]

-1	6
2	3
4	5

Note: To Add or to Subtract 2 Matrix both their dimension should be same

Note: while +, or -, of 2 matrices if we take i, j from mat1, Even from mat2 we need to take i, j

Mat1[N][M], Mat2[N][M], Mat3[N][M]

Q) Add Mat1 & Mat2 & Store in Mat3

Dimensional check for a matrix is Imp

```
i = 0; i < N; i++){
    j=0; j<M; j++){
        Mat3[i][j] = Mat1[i][j] + Mat2[i][j]
    }
}
```

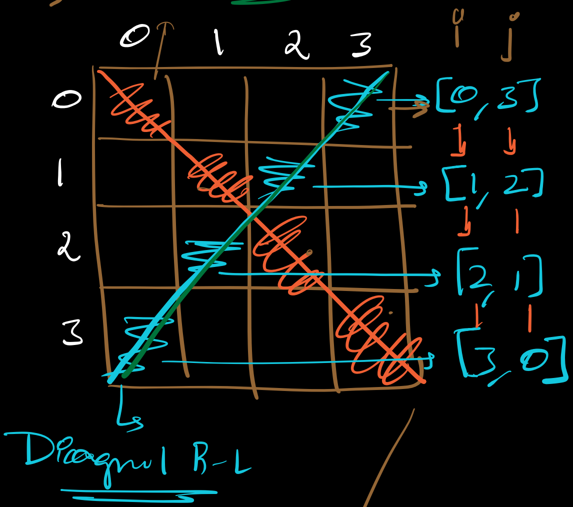
Mat1[N_1][M_1]
Mat2[N_2][M_2]

$N_1 == N_2$

$M_1 == M_2$

For add/ Subtraction

→ 4x4 Square



Observations :

A) i is increasing &
j is decreasing

B) i+j == N-1

C) (i, j) start = [0, N-1]

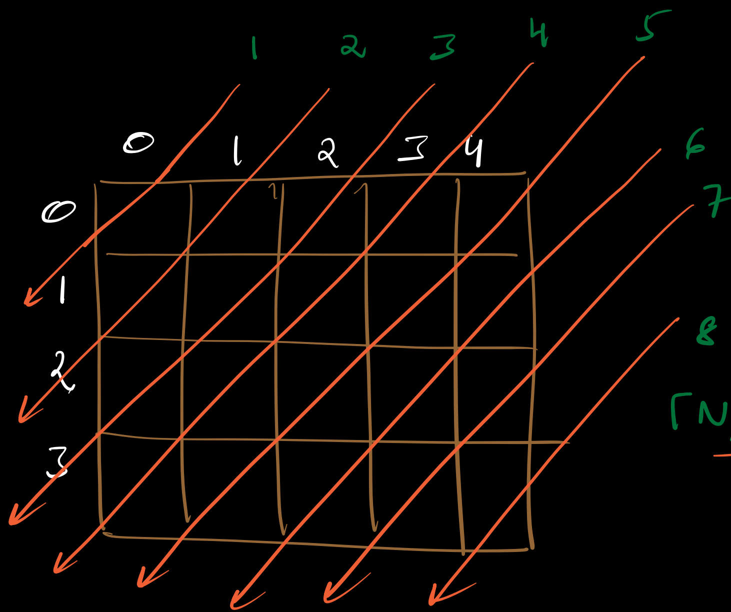
d) (i, j) end = [N-1, 0]

Diagonal R-L

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

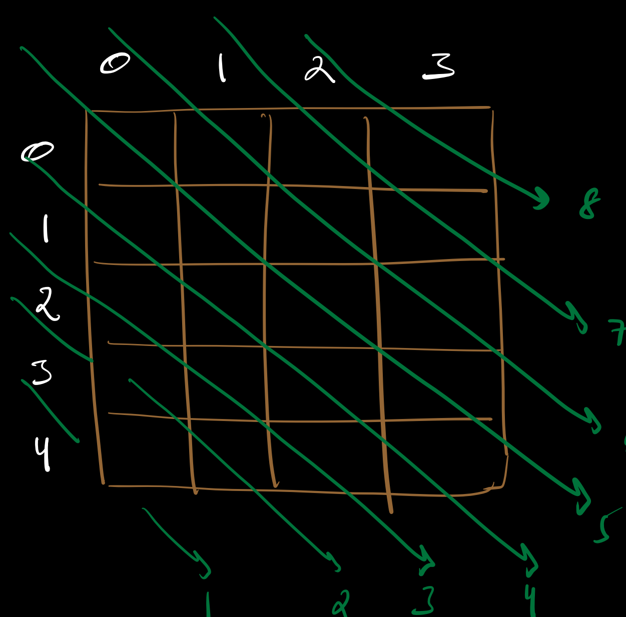
```
i = 0; i < N ; i++){
    print( mat[i][i])
}
```

Diagonal L-R



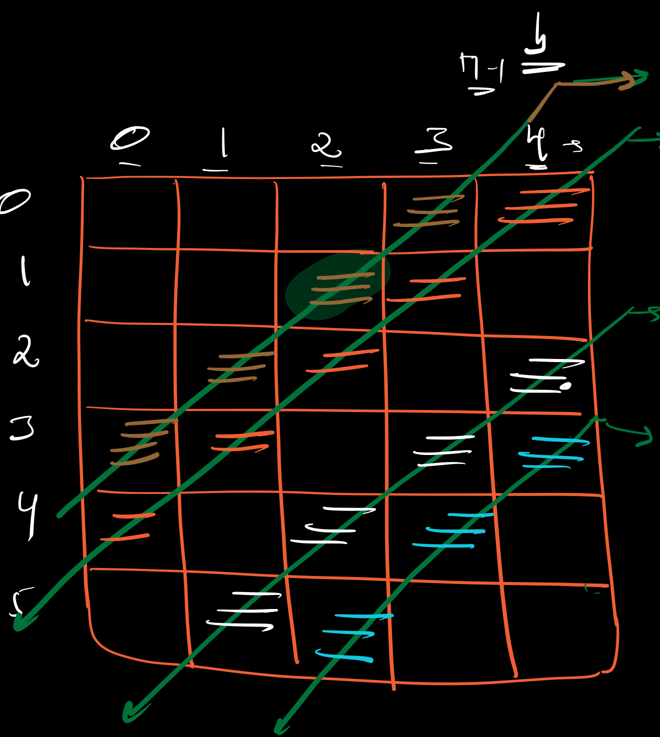
Total R-L = N+M-1

[N, M] = [4, 5]



Total L-R : N+M-1

3Q) Print all diagnols from R-L



i+j sum same

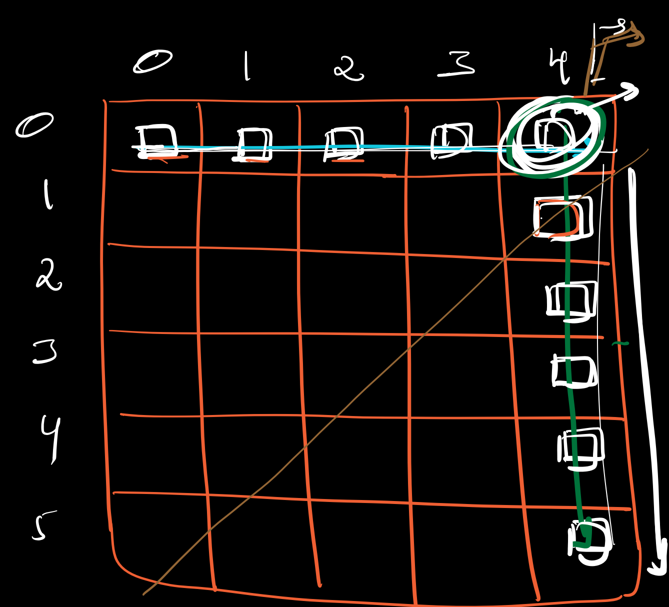
[0 3] [1 2] [2 1] [3 0]:

[0 4] [1 3] [2 2] [3 1] [4 0]

[2, 4] [3 3] [4 2] [5, 1]

[3, 4] → [4, 3] → [5, 2] →

In every right to left diagnol i+j is same.



What all cells can be start of right diagnols

= All first row & last colum elements

M + N − 1

In single row M elements

In single column N elements

{10:30 PM}

// All right diagonal start

a) 0th row } ⟶ iteration of 0th row

```
j = 0; j < M; j++) {
    (s, e) = [0, j]   // start point of diagonal
    while ( s < N && e >= 0)
        print(mat[s][e])
        s++, e--
    }
    print("\n")
}
```

row   column

e

e--

⟶

0

s = row number is continously

Increasing max s value we can

have is N-1

e = colum number, is continously

decreasing, min e value we can

have is 0

b) M-1th Column

We need to start from 1?
Because we already Considered [0, M-1]

```
i = 0; i < N; i++){    [1]

    (s, e) = [i, M-1]

    while ( s < N && e >= 0)
        print(mat[s][e])
        s++, e--
    }
    print("\n")
}
```

Given a Square Matrix

→ Find Transpose of a matrix?

$$\begin{bmatrix} 3 & -2 & 1 & 6 \\ 4 & 3 & 2 & 1 \\ 7 & 2 & 9 & 3 \\ 2 & 1 & 6 & 8 \end{bmatrix} \qquad \underbrace{\begin{bmatrix} 3 & 4 & 7 & 2 \\ -2 & 3 & 2 & 1 \\ 1 & 2 & 9 & 6 \\ 6 & 1 & 3 & 8 \end{bmatrix}}_{\text{Transpose}}$$

$$\underbrace{\begin{bmatrix} 3 & -2 & 1 & 6 \\ 3 & 3 & 6 & 9 \\ 2 & 1 & 7 & 8 \end{bmatrix}}_{3 \times 4} \qquad \underbrace{\begin{bmatrix} 3 & 3 & 2 \\ -2 & 3 & 1 \\ 1 & 6 & 7 \\ 6 & 9 & 8 \end{bmatrix}}_{4 \times 3}$$

0^{th} row $\xrightarrow{\text{After transpose}}$ 0^{th} column

1^{st} row $\longrightarrow$ 1^{st} column

i^{th} row $\longrightarrow$ i^{th} column

Note: In a rectangular matrix, after transpose dimension will change

Ex:

mat[5][5] :

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & -1 & 3 & 2 & 6 & -2 \\
1 & 4 & 1 & 3 & \boxed{5} & 8 \\
2 & -3 & 2 & -6 & 1 & 5 \\
3 & 7 & 3 & 2 & 9 & -8 \\
4 & 4 & \boxed{-3} & 2 & 4 & 3
\end{array}
\xrightarrow{\text{Transpose}}
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & -1 & 4 & -3 & 7 & 4 \\
1 & 3 & 1 & 2 & 3 & \boxed{-3} \\
2 & 2 & 3 & -6 & 2 & 2 \\
3 & 6 & \boxed{5} & 1 & 9 & 4 \\
4 & -2 & 8 & 5 & -8 & 3
\end{array}
$$

An Element mat[i][j] is at mat[j][i]

After transpose

Pseudo Code :

```
i = 0; i < N; i++) {
    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]

wrong because

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & -1 & 3 & 2 & 6 & -2 \\
1 & 4 & 1 & 3 & \boxed{5} & 8 \\
2 & -3 & 2 & -6 & 1 & 5 \\
3 & 7 & 3 & 2 & 9 & -8 \\
4 & 4 & \boxed{-3} & 2 & 4 & 3
\end{array}
$$

→ upper triangular matrix

{ lower triangular matrix }

Todo

Iterate on lower triangular matrix and Do transpose.

```
i = 1; i < N; i++){
        j = 0; j < i; j++){
            swap
              Mat[i][j] & 4
              Mat[j][i]
        }
}
```

// lets iterate upper trian

```
i = 0 ; i < N ; i++){
    j = i+1; j < N; j++){
        swap
          Mat[i][j] & 4
          Mat[j][i]
    }
}
```

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

→ Given a Square Matrix rotate 90°, Clock wise

	0	1	2	3	4
0	-3	2	1	6	4
1	2	3	-1	6	2
2	3	9	8	-1	2
3	3	-2	4	6	1
4	9	-3	1	2	-9

90°

Again 90°, Clock wise rotation

	0	1	2	3	4
0	9	3	3	2	-3
1	-3	-2	9	3	2
2	1	4	8	-1	1
3	2	6	-1	6	6
4	-9	1	2	2	4

Transpose

	0	1	2	3	4
0	-3	2	3	3	9
1	2	3	9	-2	-3
2	1	-1	8	4	1
3	6	6	-1	6	2
4	4	2	2	1	-9

Reverse

	0	1	2	3	4
0	9	3	3	2	-3
1	-3	-2	9	3	2
2	1	4	8	-1	1
3	2	6	-1	6	6
4	-9	1	2	2	4

To Rotate 90° Clockwise

1) Transpose : →
2) Reverse every row →

TC: $O(N^2)$

SC: $O(1)$

→ Given a Matrix [N][M] Print the boundary
↳ Clockwise dire

Mat[4][5]

	0	1	2	3	4
0	-1	3	-2	4	6
1	2	4	-3	2	-6
2	3	-2	4	-1	2
3	1	3	1	2	-1

Printing:

1) 0^{th} row (L-R)
2) $M-1^{th}$ colum (T-D)
3) $N-1^{th}$ row (R-L)
4) 0^{th} colum (D-T)

Mat[N][M]

{ Note: Take care of corner Elements }

4 Steps:

0^{th} row:

```
j = 0; j < M-1; j++) {
        print( mat[i][j])
}
```

→ M-1

// After above loop j = M-1

$M-1^{th}$ col:

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

N-1

// After above loop i = N-1

$\overset{M}{N-1}$ row
(R-L)

```
j = M-1; j>0; j--){
    print(mat[i][j])
}
```
} M-1

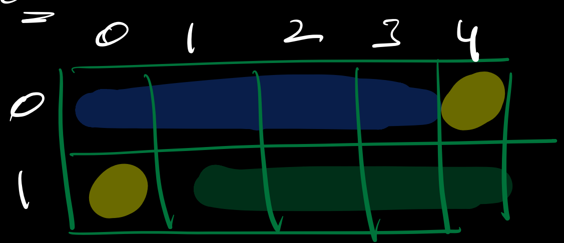
// After above loop j=0

$\overset{M}{0}$ col
(B-T)

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

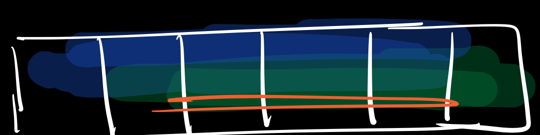
// After above loop i=0, j=0

Ex:



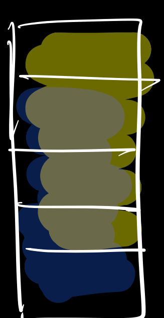
[2*5] ✓

Ex1:



[1*6] → * it won't work

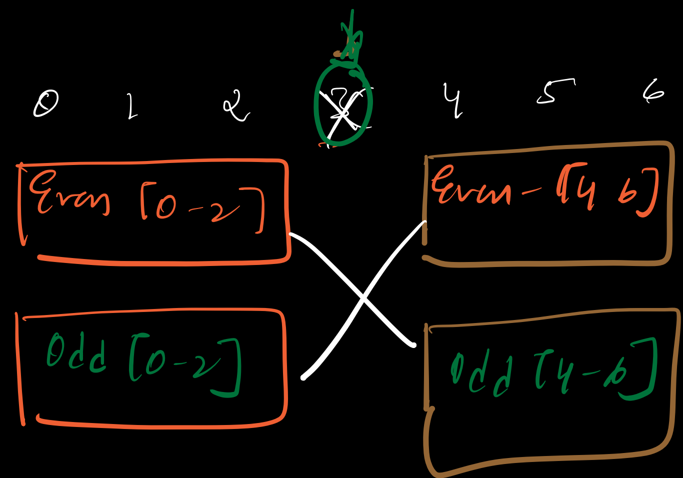
Ex2:



[5*1] → it won't work.
N M

Note: If it's a single row or column, it won't work. (Print entire data directly)

Doubts:



PFE[i] = Sum of all Even indices from [0-i]

PFO[i] = Sum of all Odd indices from [0-i]

	0	1	2	3	4	5
arr[6]:	3	-1	2	6	2	1
PFE[]:	3	3	5	5	7	7
PFO[]:	0	-1	-1	5	5	6

$$\forall_{i=0}^{N-1} \left\{ SumE[0, i-1] + SumO[i+1, N-1] = SumE[i+1, N-1] + SumO[0, i-1] \right\}$$

XOR Prefix value

Pf[i] = Pf[i-1] ^ arr[i]

⇒ If update in array

1pt

array 1pt