SCALER

2D Arrays

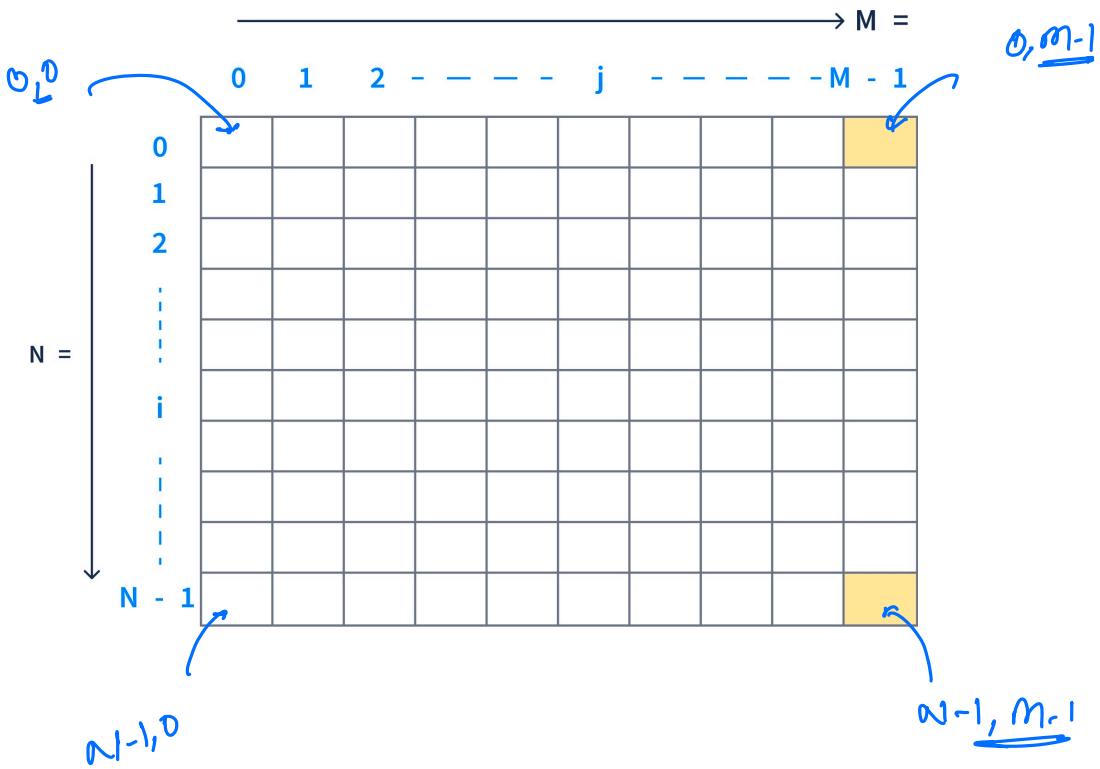
TABLE OF CONTENTS


```
int arr[N];

int[] arr = new int[N];
```

2-D Array

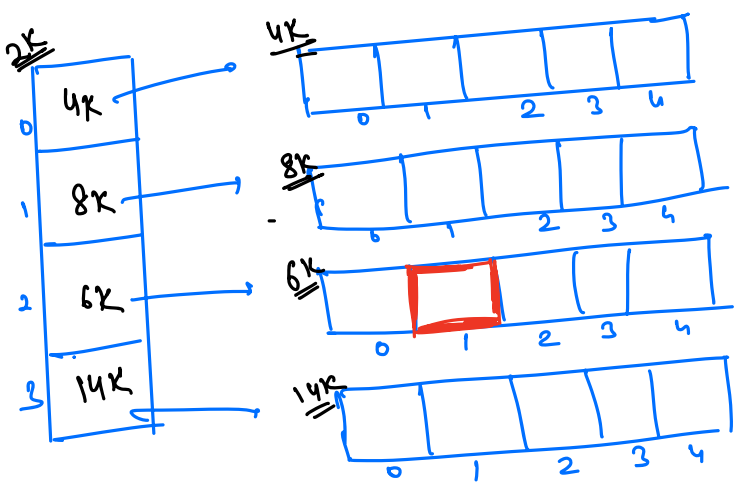
```
int arr[N][m];

int[][] arr = new int[N][m];

arr = [[0] * m for _ in range(N)]
```

- int arr[N][M] , total number of elements → N x m



How is it actually stored?

int[][] arr = new int[4][5];



arr[2][1] → 8K[2][1] → 6K[1] ⇒ T.C for accessing any element in 2-D array ⇒ O(1)

< Question > : Given arr[N][M], print row-wise sum.

- arr [3] [4] →

	0	1	2	3	
0	10	2	7	3	→ 22
1	9	5	-1	8	→ 21
2	3	11	15	20	→ 49

< / > Code

```
void printRow-wise sum(int arr[N][M]){
    for( i = 0; i < N; i++){
        int sum = 0;
        for( j=0; j < m; j++){
        {       sum += arr[i][j];
        }
        print(sum);
    }
}
```

Time → 1 sec.
Space → 128 mB / 256 mB

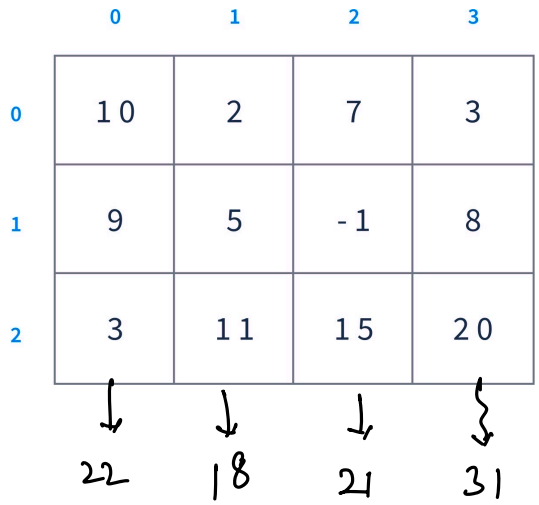
→ 128×10^6 B
→ 1.28×10^8 B.

4 B → 1 int.
1 B → 1/4 int.
1.28×10^8 B → $\frac{1}{4} \times 32 \times 10^6$
$\Rightarrow 3.2 \times 10^7$

T.C → O(N x m)
S.C → O(1)

< *Question* > : Given arr[N][M], print column-wise sum.

- arr [3] [4] →



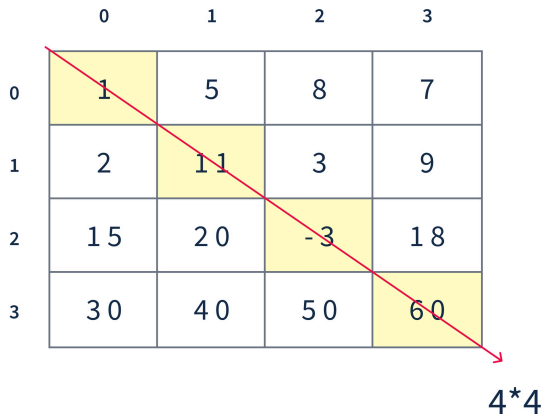
< / > *Code*

```
void printColumn-wise sum(int arr[N][M]){
    for( j = 0; j < m; j++){
        sum = 0;
        for( i = 0; i < n; i++){
            sum += arr[i][j];
        }
        print(sum);
    }
}
```

T.C → O(N x m)
S.C → O(1)

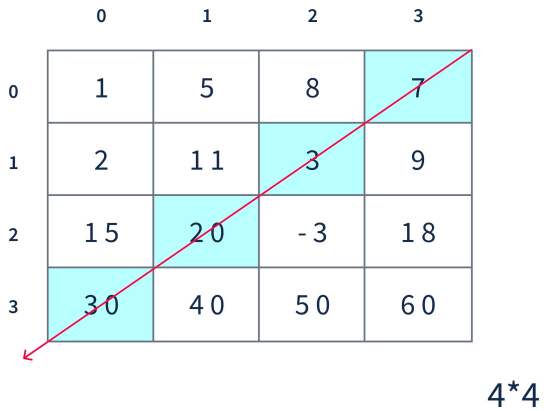
square matrix.

< Question > : Given arr[N][N]. Print diagonals.



	0	1	2	3
0	1	5	8	7
1	2	11	3	9
2	15	20	-3	18
3	30	40	50	60

4*4



	0	1	2	3
0	1	5	8	7
1	2	11	3	9
2	15	20	-3	18
3	30	40	50	60

4*4

o/p. 1 11 -3 60

0,0
↓
1,1
↓
2,2
↓
3,3

o/p → 7 3 20 30

0,3
↓
1,2
↓
2,1
↓
3,0

Code →

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

T.C → O(N)
S.C → O(1)

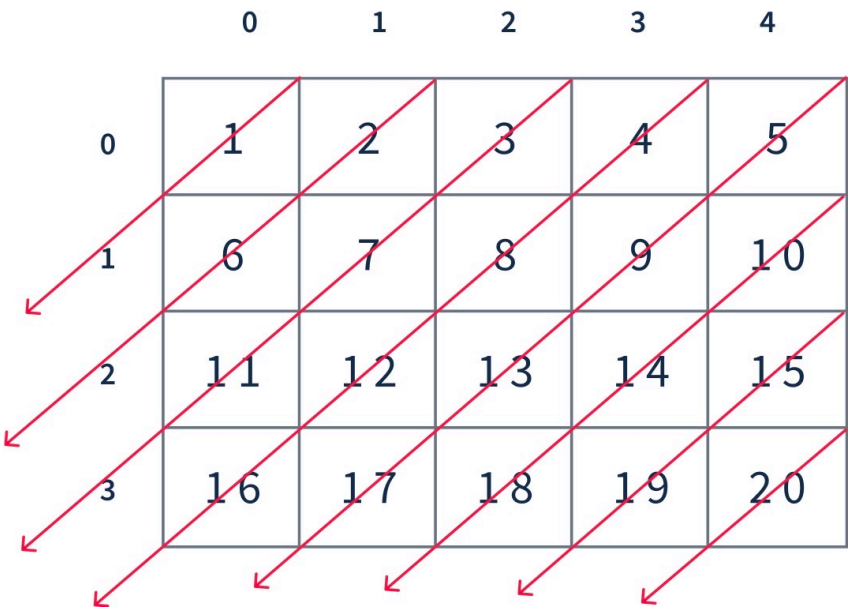
#code.

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

T.C → O(N)
S.C → O(1)

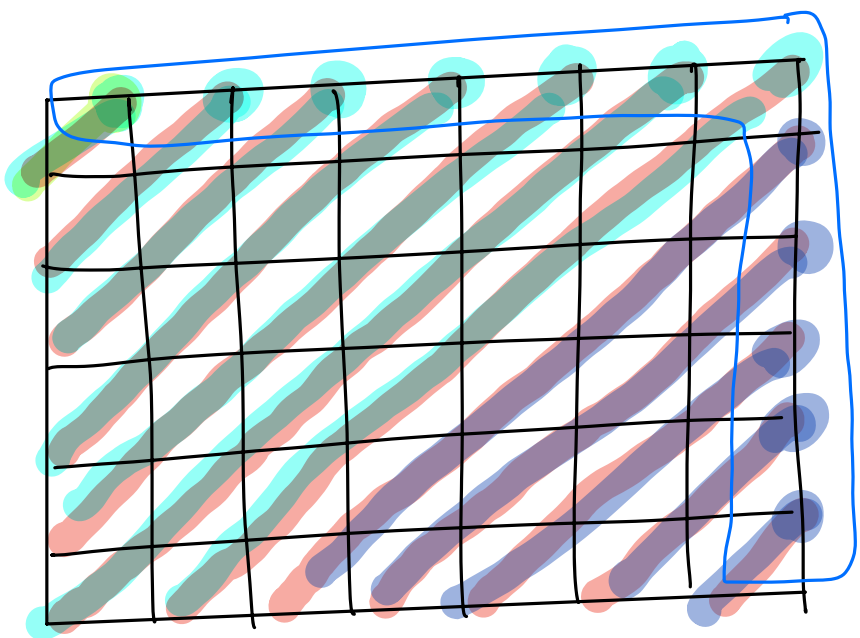
< *Question* > : Given arr[N][M]. Print all the elements diagonally from right to left.

- arr [4] [5] →



Expected output:

1
2, 6
3, 7, 11
4, 8, 12, 16
5, 9, 13, 17
10, 14, 18
15, 19
20

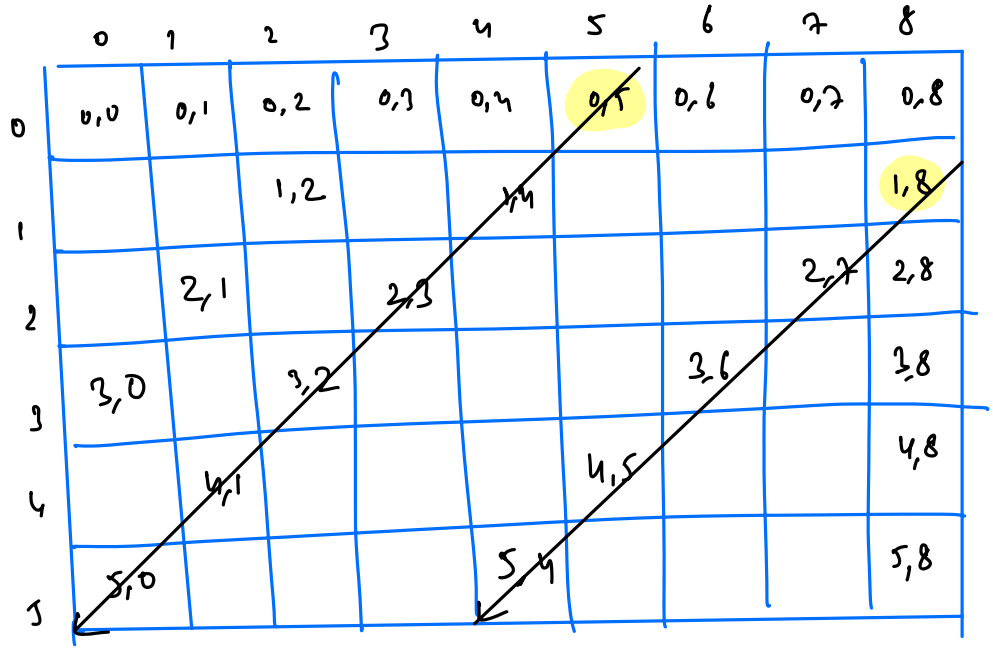


Total no. of diagonals

$n + (N-1) \Rightarrow n+N-1$

no. of diagonals starting from 0th row

no. of diagonals starting from last column.



① - print all diagonals starting from 0^{th} row.

② - print all diagonals starting from $m-1^{th}$ column.

< / > *Code*

```
void printAllDiagonals(int arr[N][M]){
    //print the diagonals starting from 0th row
        for( col=0; col<m; col++){
            i=0, j=col;
            while(i<N && j>=0 ){
                print(arr[i][j]);
                i++;
                j--;
            }
        }
    //print the diagonals starting from last column
        for(row=1; row<N; row++){
            i=row, j=m-1
            while(i<N && j>=0 ){
                print(arr[i][j];
                i++;
                j--;
            }
        }
}
```

T.C → O(N×m)
S.C → O(1)

Transpose of a Square Matrix

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

Transpose →

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

0,1 ↔ 1,0
0,2 ↔ 2,0
0,3 ↔ 3,0

1,2 ↔ 2,1
1,3 ↔ 3,1

2,3 ↔ 3,2

#Code :-

```
for( i = 0; i < N; i++){
    for( j = i+1; j<N; j++){
        //swap arr[i][j] with arr[j][i]
        int temp = arr[i][j];
        arr[i][j] = arr[j][i];
        arr[j][i] = temp;
    }
}
```

T.C → O(N²)
S.C → O(1)

A →

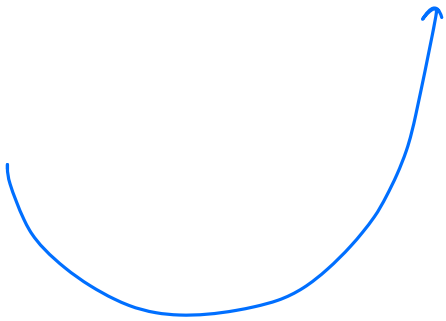
2	3
7	12
11	4
5	8
9	6

5 × 2

A^T →

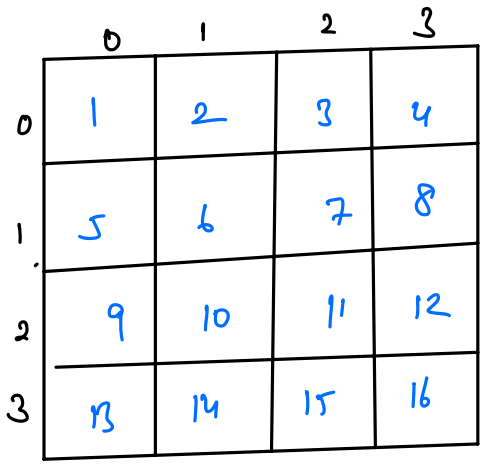
2	7	11	5	9
3	12	4	8	6

2 × 5

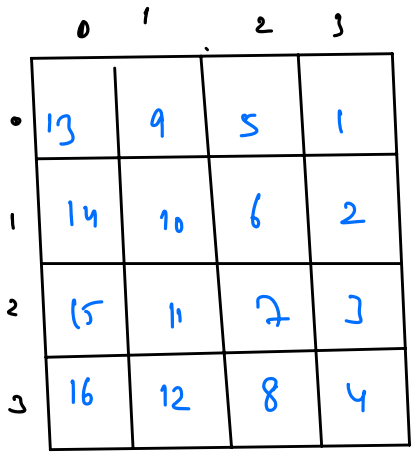


Need to create another 2D array.

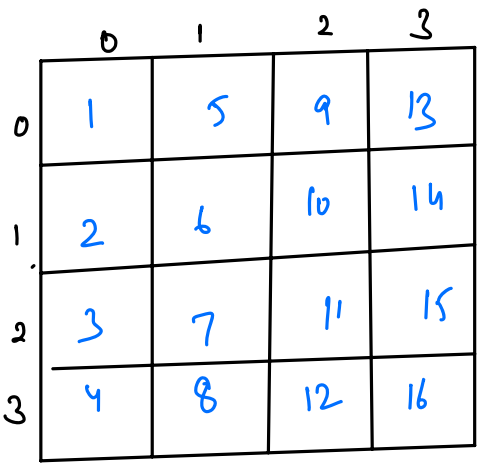
Rotate a mat[N][N]



Rotate by 90° in clockwise



transpose



?

Reverse the elements of every row.

#code →

```
for( i = 0; i < N; i++){
    for( j = i+1; j < N; j++){
        //swap arr[i][j] with arr[j][i]
        int temp = arr[i][j];
        arr[i][j] = arr[j][i];
        arr[j][i] = temp;
    }
}
```

} transpose of given matrix

```
for( i = 0; i < N; i++){
    reverse( arr[i]);
}
```

} Reverse every row.

T.C → $O(N^2)$
S.C → $O(1)$

```
void reverse ( int[] a) {
    l = 0 , r = a.length - 1;
    while( l < r) {
        int temp = arr[l];
        arr[l] = arr[r];
        arr[r] = temp;
        l++;
        r--;
    }
}
```

arr[] = [1, 12, 10, 3, 14, 10, 5, 4, 9, 20] B = 8

ans = 2.

① Find no. of elements $\leq B$. → (x)

② Explore all the subarrays of size x & in every subarray find how many swaps are required to bring all elements $\leq B$ in that subarray.

for every window of size x, find the no. of elements > B
using sliding window technique.