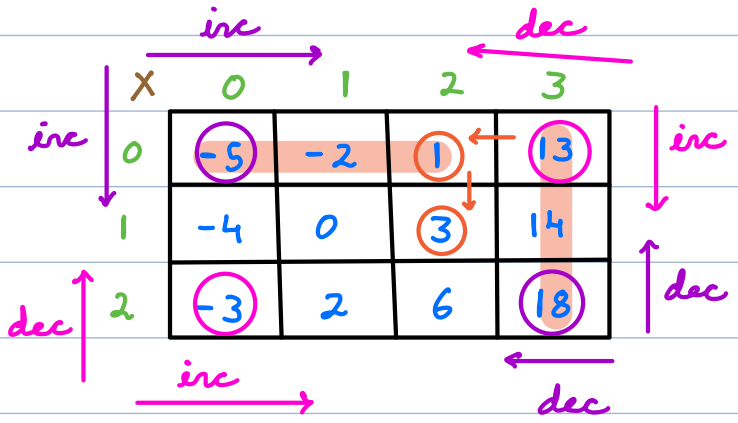
Q → Given a row wise & column wise sorted matrix.
Find out whether element K is present or not.

A =

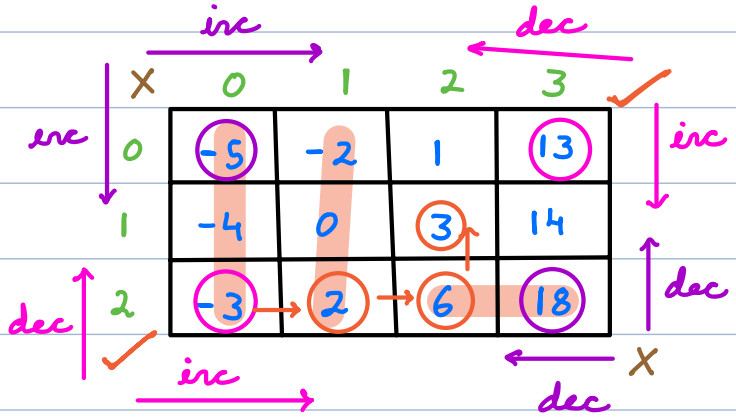
	0	1	2	3
0	-5	-2	1	13
1	-4	0	3	14
2	-3	2	6	18

K		Ans
13	→	true
2	→	true
15	→	false

Bruteforce → check all cells. TC = O(N*M) SC = O(1)



K = 3 → true



K = 3 → true

// top right N → # rows M → # col

```
r = 0    c = M-1
while (r < N && c >= 0) {
    if (A[r][c] == K)  return true
    else if (A[r][c] < K)  r++
```

```
    else  c--
}
```

TC = O(N+M) SC = O(1)

```
return false
```

Q → Given a binary (0/1) sorted (rows) matrix of size N*M.
Find the row with max # 1's.

	0	1	2	3
0	0	0	1	1
1	0	0	0	1
→ 2	0	1	1	1

Ans = 2

Bruteforce → ∀rows, count the # 1's.

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

TC = O(N)

	0	1	2	3	4	5
0	0	0	0	0	1	1
→ 1	0	0	1	1	1	1
2	0	0	0	0	0	0
3	0	0	0	1	1	1
4	0	0	1	1	1	1

r = 0 1 2 3 4 5

c = 5 4 3 2 1

ans = -1 0 0 1 ✓

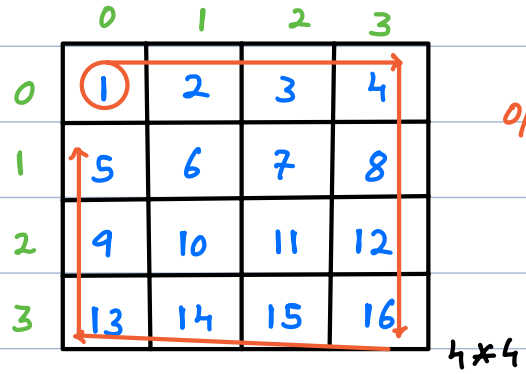
ans = -1

TC = O(N+M) SC = O(1)

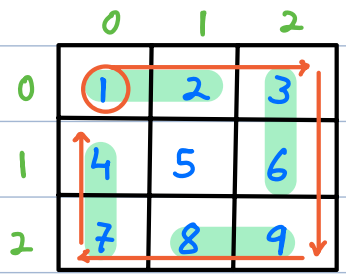
```
// top right
r = 0    c = M-1
while ( r < N && c >= 0 ) {
    while ( c >= 0 && A[r][c] == 1 ) {
        c--
        ans = r
```

```
    }
    r++
}
return ans
```

Q→ Given a square matrix (N*N).
Print boundary elements in clockwise direction.



o/p → 1 2 3 4 8 12 16 15 14 13 9 5



o/p → 1 2 3 6 9 8 7 4

```
r = 0
for c → 0 to (N-2) {
    print (A[r][c])
}
c = N-1
for r → 0 to (N-2) {
    print (A[r][c])
}
r = N-1
```

```
for c → (N-1) to 1 {
    print (A[r][c])
}

c = 0
for r → (N-1) to 1 {
    print (A[r][c])
}
```

TC = O(4N) → O(N)

SC = O(1)

Q → Given an integer N, fill the N*N matrix with elements 1 to N^2 in spiral order (clock wise), starting from (0,0).

N=4

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

4*4

N=5

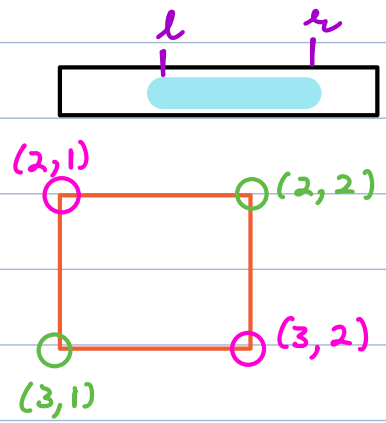
1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Code → HW TC = O(N^2) SC = O(1)

Submatrix

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

eg.



✓ top left & bottom right OR
top right & bottom left

Q → Given a matrix, find sum of all possible submatrix sum.

	0	1	2
0	4	9	6
1	5	-1	2

(0, 0)

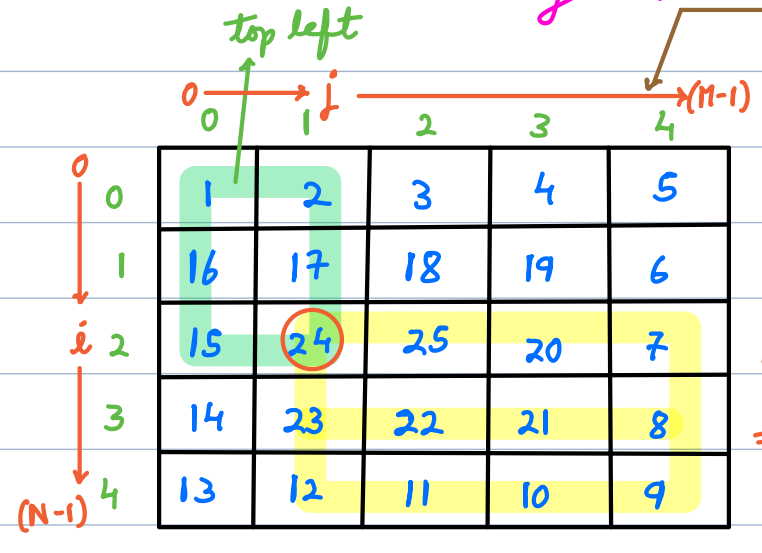
4 → (0+1) * (0+1) * (2-0) * (3-0)
= 1 * 1 * 2 * 3 = 6

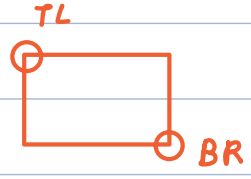
[4] 4 | [4 9] 13 | [4; 5] 9 | [4 9; 5 -1] 17
[9] 9 | [9 6] 15 | [9; -1] 8 | [9 6; -1 2] 16
[6] 6 | [5 -1] 4 | [6; 2] 8 | [4 9 6; 5 -1 2] 25
[5] 5 | [-1 2] 1
[-1] -1 | [4 9 6] 19
[2] 2 | [5 -1 2] 6

Ans = 166

Contribution Technique → Ans = $\sum_i \sum_j$ contribution of A[i][j]

A[i][j] * (# submatrix A[i][j] is a part of)





top left → (i+1) * (j+1)
bottom right → (N-i) * (M-j)

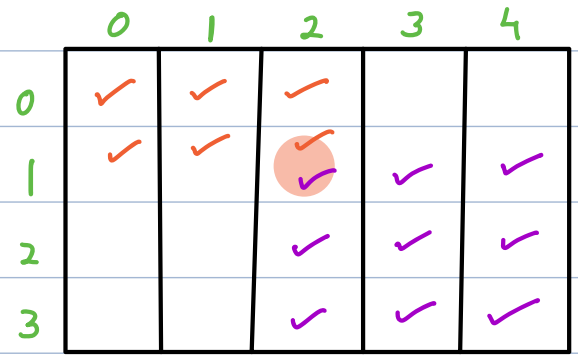
[i N-1] → N-x-i+x = N-i

Submatrix = (i+1) * (j+1) * (N-i) * (M-j)

$$Ans = \sum_{i} \sum_{j} A[i][j] * (i+1) * (j+1) * (N-i) * (M-j)$$

```
ans = 0
for i → 0 to (N-1) {
    for j → 0 to (M-1) {
        ans += A[i][j] * (i+1) * (j+1) * (N-i) * (M-j)
    }
} return ans
```

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



6 * 9 = 54