Q Given a row-wise & column wise sorted matrix. find whether a given element K is present or not.

M =

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

K = 13 ⇒ True
K = 2 ⇒ True
K = 15 ⇒ false.

Sol^n

1) Brute force

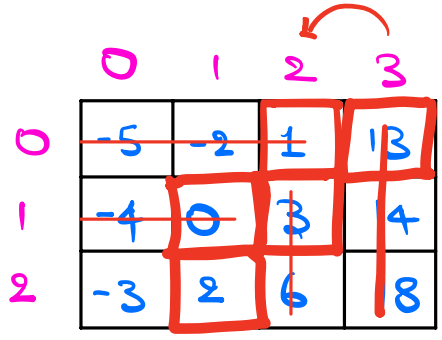
Iterate over the matrix

T.C. = O(N x M)
S.C. = O(1)

2) Optimised Approach

K = 2

M =



(0, 3) ⇒ 13 > 2

↓ (j--)

(0, 2) ⇒ 1 < 2

⇓ (i++)

(1, 2) ⇒ 3 > 2

⇓ (j--)

(1, 1) ⇒ 0 < 2

⇓ (i++)

(2, 1) ⇒ 2 (True)

Code

```
i = 0 , j = M-1
while ( i < N && j >= 0) {
        if (M[i][j] == K) {
            return true;
        }
        else if ( M[i][j] < K) {
            i++;
        }
        else {
```

```
        j--;
    }
}
return false;
```

T.C. = O(M+N)

Given a binary matrix (M[i][j] = {0, 1}) Row-wise sorted. (N×N)

Find the row with the max. no. of 1's
In case of multiple rows, return the one with lower index.

$$M = \begin{bmatrix} 0, & 1, & 1 \\ 0, & 0, & 1 \\ 0, & 1, & 1 \end{bmatrix} \rightarrow 0 \text{ (Ans)}$$

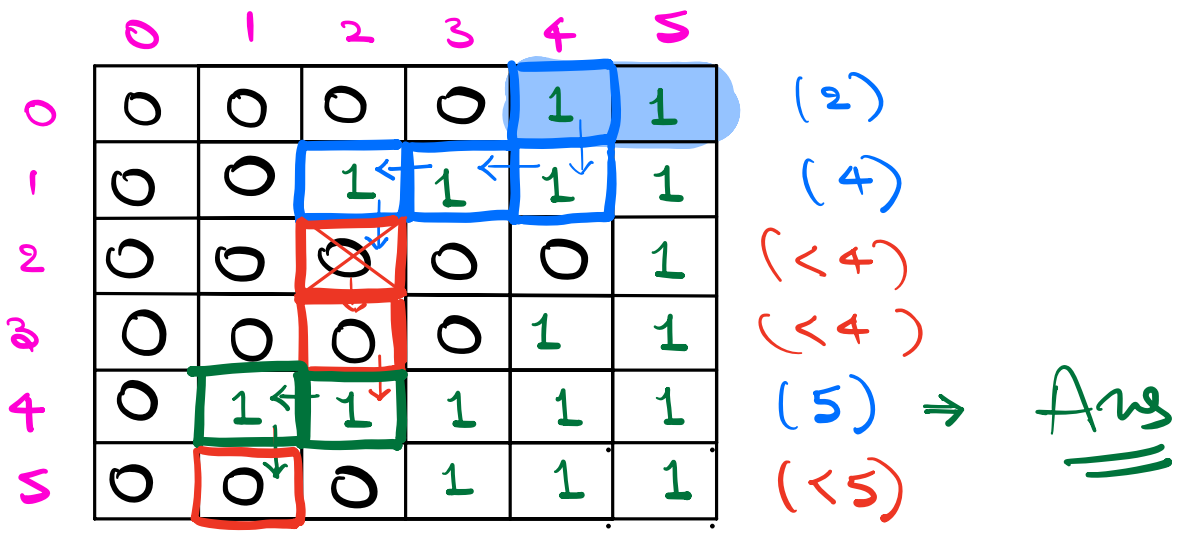
$$M1 = \begin{bmatrix} 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 1 \\ 0, & 0, & 1, & 1 \\ 0, & 1, & 1, & 1 \end{bmatrix} \rightarrow 3 \text{ (Ans)}$$

Sol^n ⇒ Brute force

⇒ Iterate over every row & count the no. of 1's

⇒ Compare the count of each row & take max.

$$T.C. = O(N \times N)$$

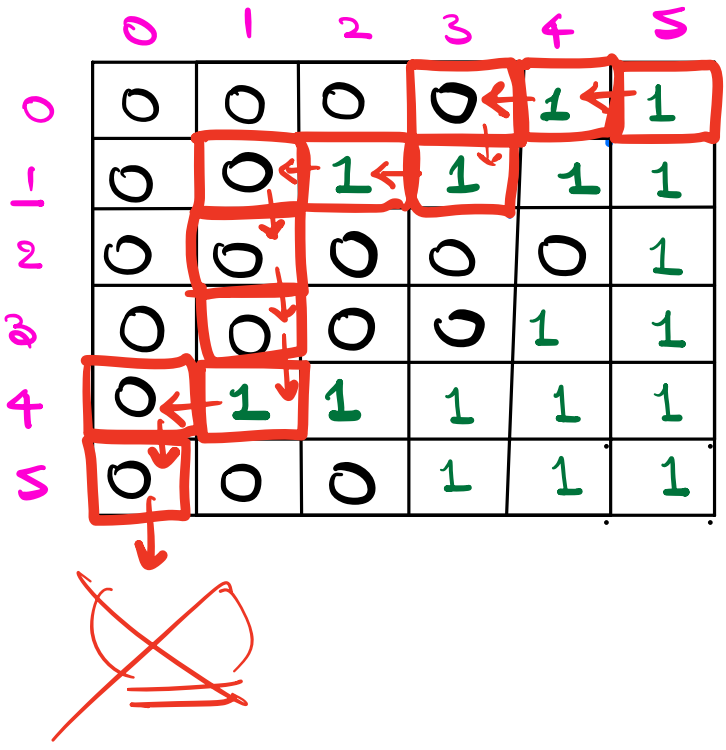


Code

```
i = 0, j = N-1;

ans = 0;
while ( i<N && j>=0) {

    if (M[i][j] == 1 ) {
        j--;
        ans = i;
```

```
        }
        else {
            i++;
        }
    }
}
return ans;
```

ans = 4

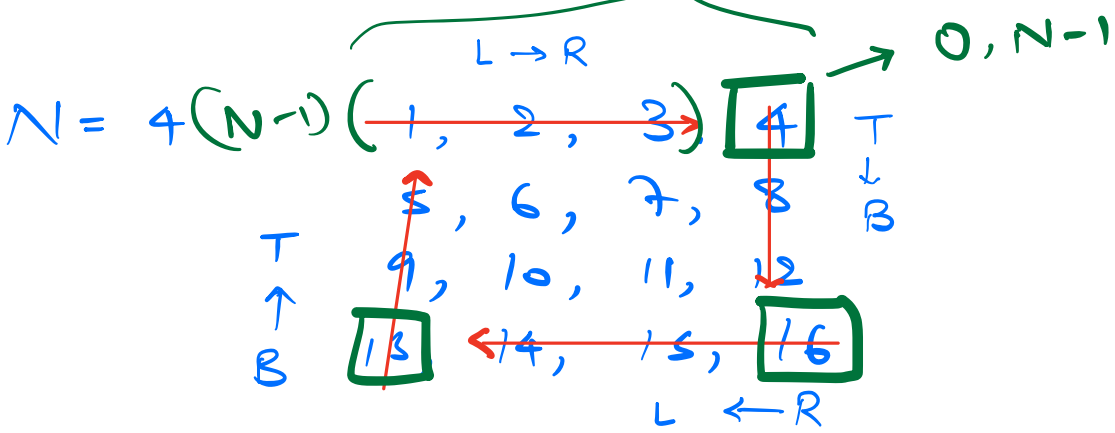


T.C. = O(N+N) = O(N)

Given a matrix of size N x N.

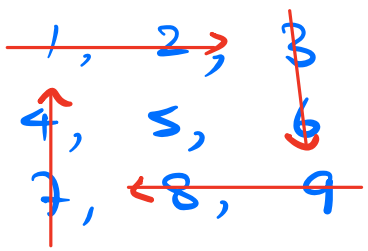
Print the boundary elements in a clock-wise direction starting from (0, 0)

N



O/p : 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5

N = 3



1, 2, 3, 6, 9, 8, 7, 4

Steps

1) Print N-1 elements in 1st row (L → R)
2) Print (N-1) elements in last col (T → B)
3) Print (N-1) elements in last row (R → L)
4) Print N-1 elements in 1st col (B → T)

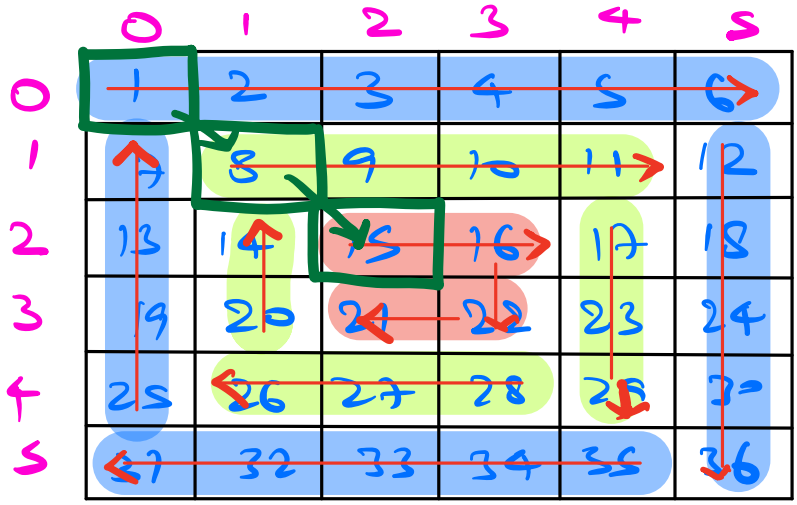
Code

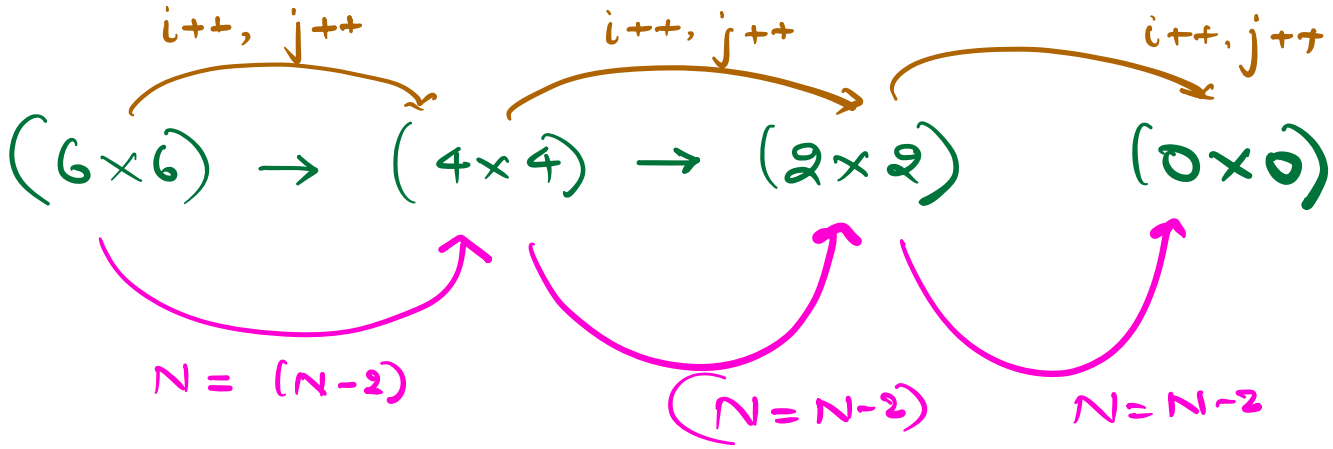
```
void printBoundary ( M[][], N ) {

    i = 0, j = 0;

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            j ++;
    }

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            i ++;
    }

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            j --;
    }

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            i --;
    }

}
```

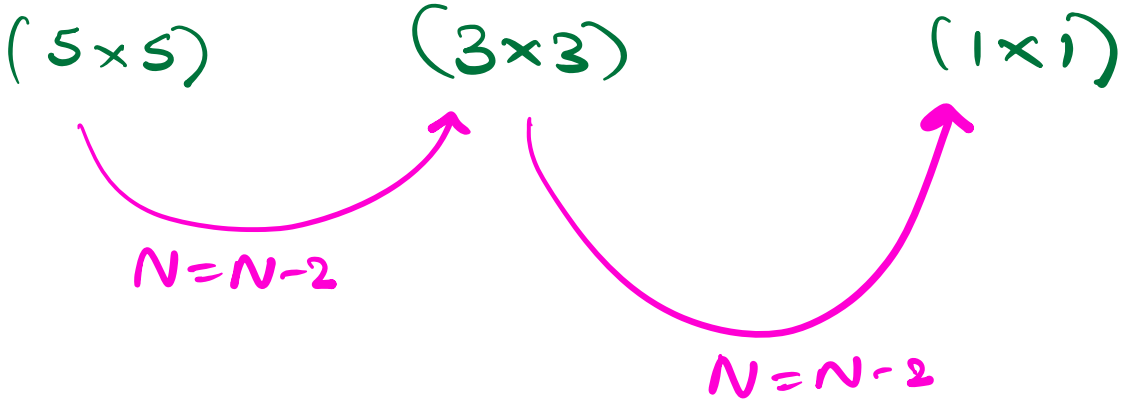
T.C. = O(4xN) = O(N)

Spiral Order Matrix

$N = 6$







$(N >, 2)$

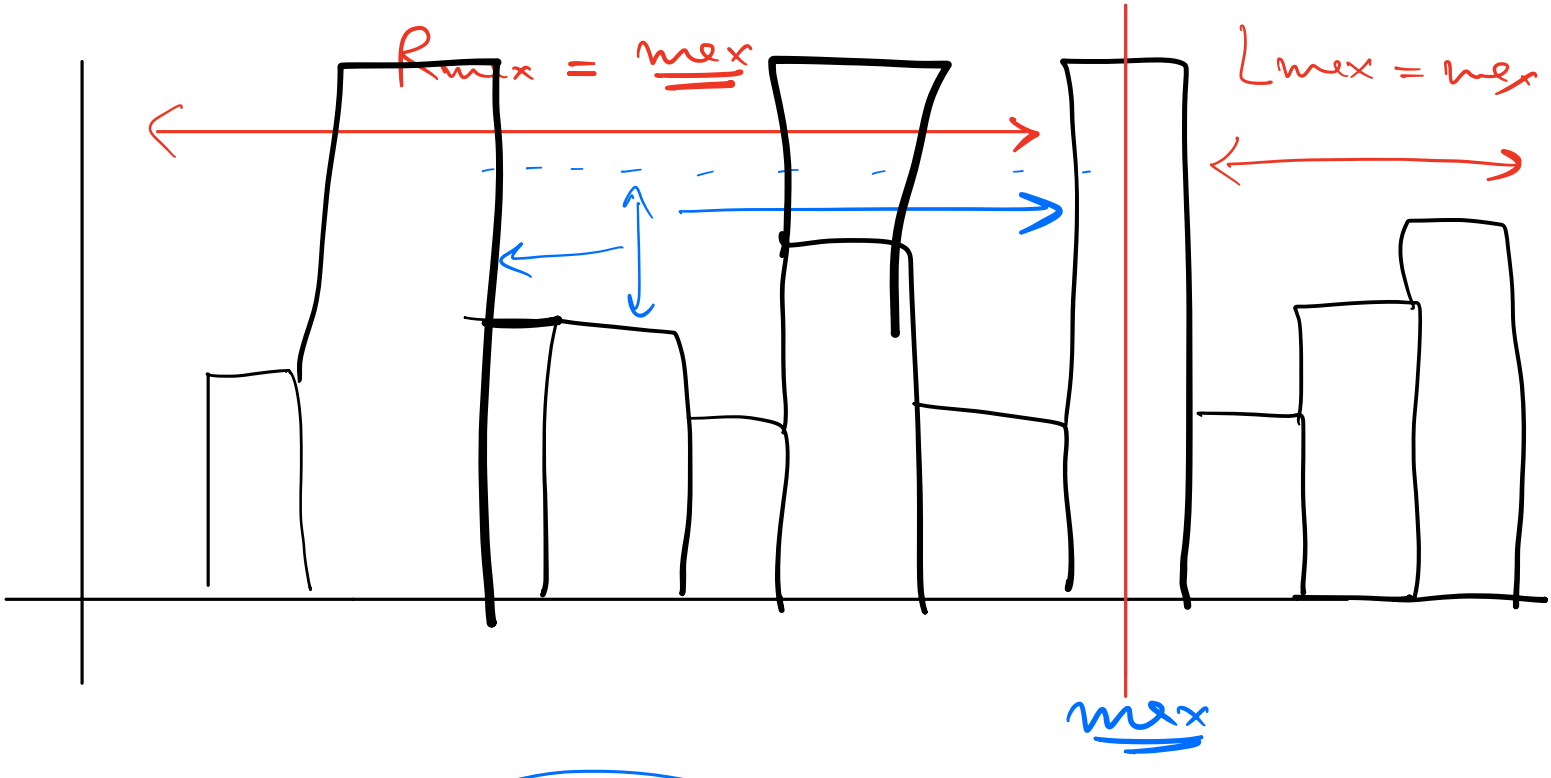
Code

```
void printBoundary ( M[][], N, i, j ) {

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            j ++;
    }

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            i ++;
    }

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            j --;
    }

    for ( count = 0;  count < (N-1) ; count ++) {
            print ( M[i][j] );
            i --;
    }

}
```

```
K = N;
i = 0, j = 0;
while (K >= 2) {
    printBoundary(M, K, i, j);
    K = K - 2;
    i++;
    j++;
}
if (K == 1) {
    print(M[i][j]);
}
```

T.C. = $O(N^2)$

S.C. = $O(1)$

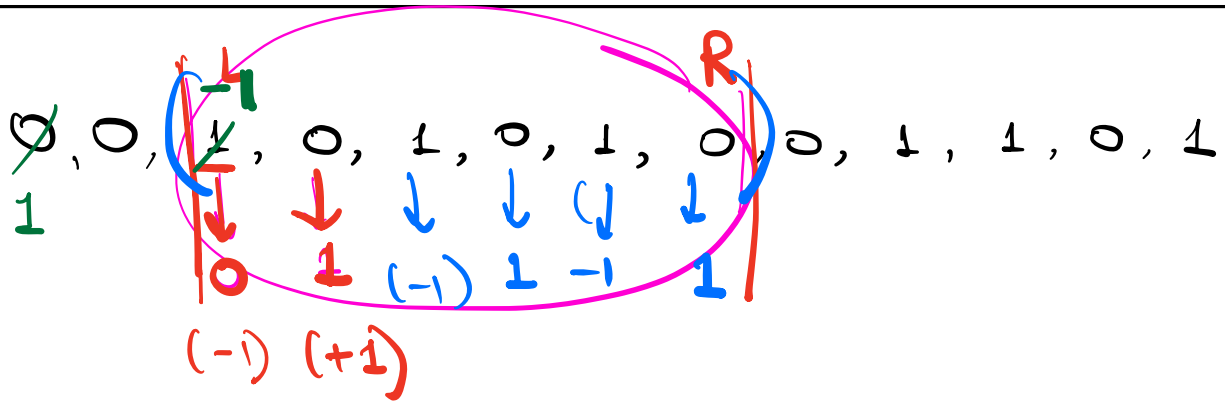


Reattempt 2

Sat (6:00 AM)

⇓

Sund (12:00 PM)



max subseq sum