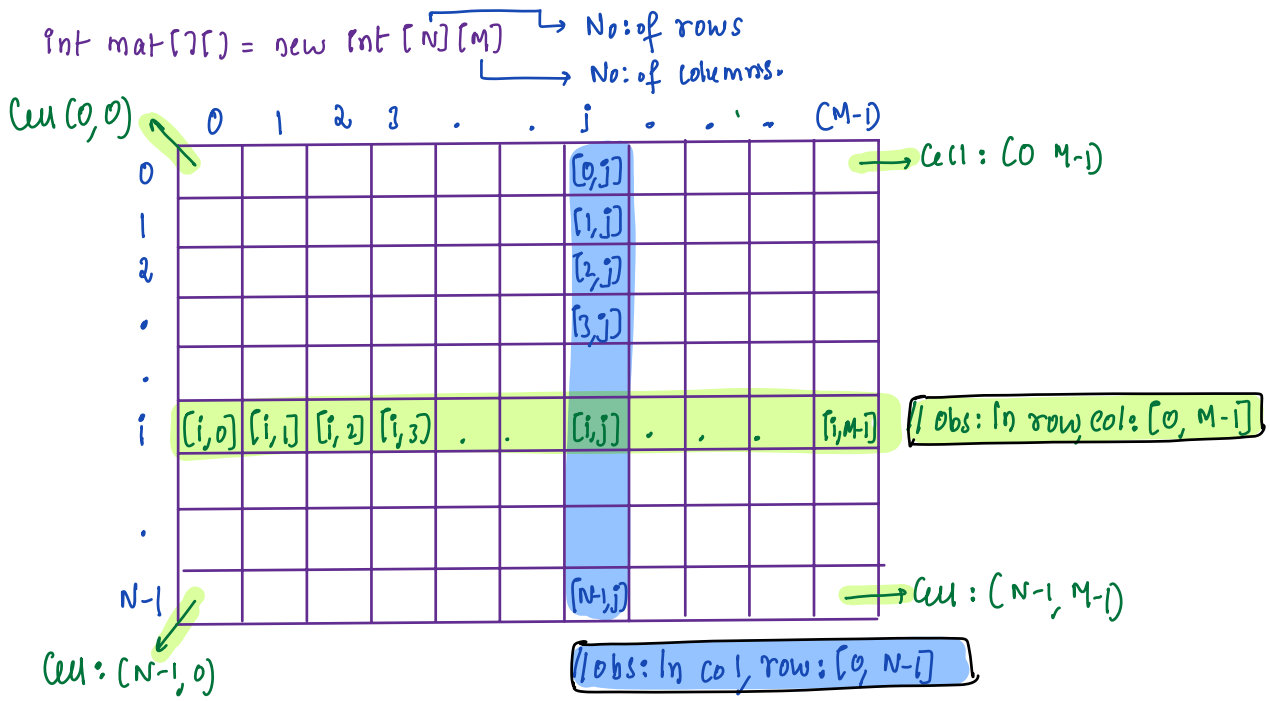
Todays Content:

1. Boundary Printing
2. Spiral Printing
3. Search for k
4. Count No: of 1's
5. Matrix Multiplication : if Time permits

2D Matrice



Q2: Given mat[N][N], print boundary in clockwise direction.

Constraints:

$1 <= N * N <= 10^6$

$-10^6 <= mat[i][j] <= 10^6$

Ex1: Mat[5][5]

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Ex2: mat[3][3]

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

O/p: 1 2 3 6 9 8 7 4

O/p: 1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6

Ideas:

mat[5][5]

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Steps: (4 = N-1 (N=5))

1. 0^{th} Row, 4 Steps From L-R
2. Last Col, 4 Steps From T-D
3. Last row, 4 Steps from R-L
4. 0^{th} Col, 4 Steps from D-T

mat[6][6]

	0	1	2	3	4	5
0	1	2	3	4	5	6
1	7	8	9	10	11	12
2	13	14	15	16	17	18
3	19	20	21	22	23	24
4	25	26	27	28	29	30
5	31	32	33	34	35	36

Steps: (5 = N-1 (N=6))

1. 0^{th} Row, 5 Steps From L-R
2. Last Col, 5 Steps From T-D
3. Last row, 5 Steps from R-L
4. 0^{th} Col, 5 Steps from D-T

```
void printBoundaryElements(int mat[][]){
    int N = mat.length;
    int i=0, j=0;

    for(int k=1; k<N; k++) // L-R
    {
        print(mat[i][j])
        j=j+1
    }

    for(int k=1; k<N; k++) // T-D
    {
        print(mat[i][j])
        i=i+1
    }

    for(int k=1; k<N; k++) // R-L
    {
        print(mat[i][j])
        j=j-1
    }

    for(int k=1; k<N; k++) // D-T
    {
        print(mat[i][j])
        i--;
    }
    if(N==1){ print(mat[i][j]) }
}
```

Eg: N=5, mat[5][5]

i=0, j=0	k<N	print( )	i	j
	1<5	mat[0][0]	0	1
	2<5	mat[0][1]	0	2
	3<5	mat[0][2]	0	3
	4<5	mat[0][3]	0	4
	5<5	Stop		

i=0, j=4	k<N	print( )	i	j
	1<5	mat[0][4]	1	4
	2<5	mat[1][4]	2	4
	3<5	mat[2][4]	3	4
	4<5	mat[3][4]	4	4
	5<5	Stop		

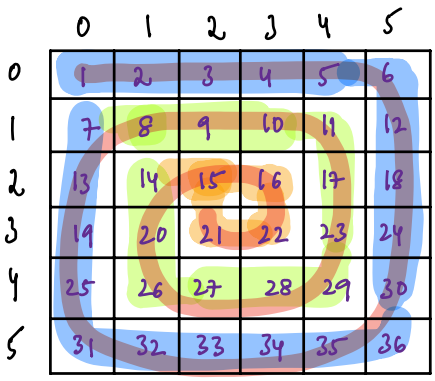
i=4, j=4	k<N	print(	i	j
	1<5	mat[4][4]	4	3
	2<5	mat[4][3]	4	2
	3<5	mat[4][2]	4	1
	4<5	mat[4][1]	4	0
	5<5	Stop		

i=4, j=0	k<N	print( )	i	j
	1<5	mat[4][0]	3	0
	2<5	mat[3][0]	2	0
	3<5	mat[2][0]	1	0
	4<5	mat[1][0]	0	0
	5<5	Stop	Final (i,j) = (0,0)	

Edge: N=1 mat[1][1] = 0 | 8 | output: 8

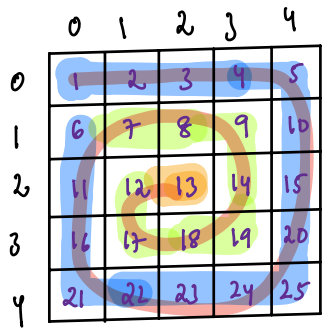
Spiral Printing

mat[6][6]



i	j	N	Iterations	After Loop i, j
0	0	6	N-1, 5	0, 0 ↻
↓+1	→+1	↓-2		
1	1	4	N-1, 3	1, 1 ↻
↓+1	→+1	↓-2		
2	2	2	N-1, 1	2, 2 ↻
↓+1	→+1	↓-2		
3	3	0 Stop		

mat[5][5]



i	j	N	Iterations	After Loop i, j
0	0	5	N-1, 4	(0, 0) → (1, 1) ↻
↓+1	→+1	↓-2		
1	1	3	N-1, 2	(1, 1) → (2, 2) ↻
↓+1	→+1	↓-2		
2	2	1	N-1, 0 Stop,	

if(N==1) { print(mat[i][j]) }

```
void SpiralPrinting(int mat[][]){   TC: O(N^2)   SC: O(1)
    int N = mat.length;
    int i=0, j=0;
    while(N>1){
        for(int k=1; k<N; k++){ // L-R
            print(mat[i][j])
            j = j+1
        }
        for(int k=1; k<N; k++){ // T-D
            print(mat[i][j])
            i = i+1
        }
        for(int k=1; k<N; k++){ // R-L
            print(mat[i][j])
            j = j-1
        }
        for(int k=1; k<N; k++){ // D-T
            print(mat[i][j])
            i--;
        }
        i = i+1; j = j+1; N = N-2
    }
    if(N==1){ print(mat[i][j]) }
}
```

Qs: Given rowwise columnwise sorted Matrix[N][M], find k?

Constraints:

$1 <= N, M <= 10^3$

$-10^6 <= mat[i][j] <= 10^6$

Ex: mat[6][6]

k = 12 ✓ k = 18 ✓ k = 21 ✗

	0	1	2	3	4	5	k = 12
0	-10	-5	-2	2	4	7	< 12 *
1	-7	-4	-1	3	6	9	< 12 *
2	-2	3	5	7	11	14	> 12, can be, search it in row *
3	3	6	8	11	14	17	> 12, can be, search it in row *
4	7	11	12	15	19	20	> 12, can be, search it in row ✓
5	10	14	18	20	24	29	*

Idea:

1. Iterate on mat[][] search for k

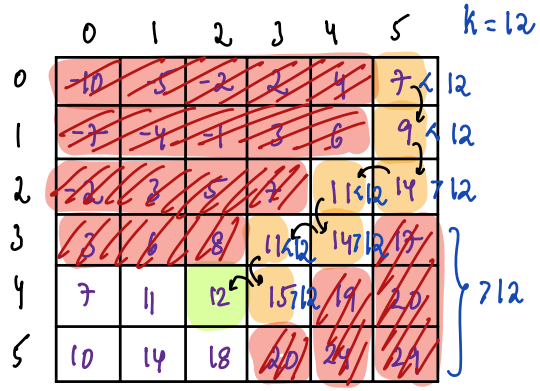
   TC: O(N*M) SC: O(1)

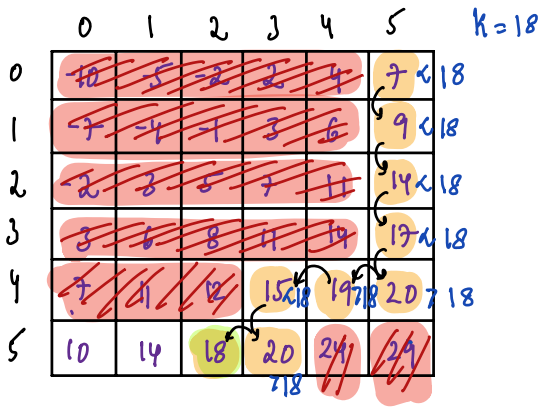
2. Compare k with last element in each row, if k can be present iterate & search on row.

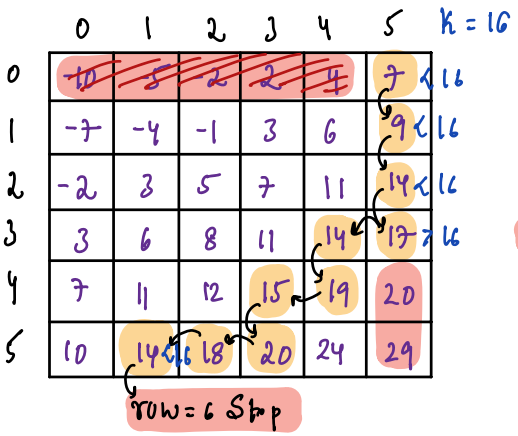
   Worst Case: We will search every row.

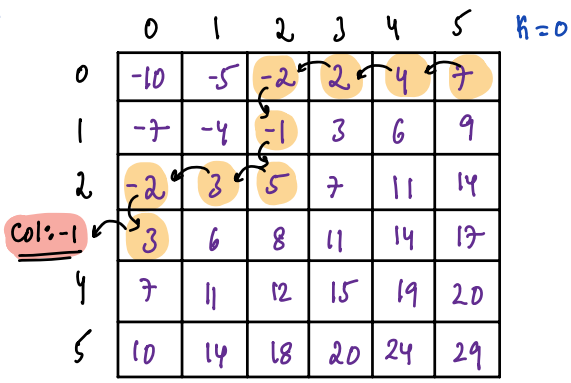
   TC: O(N*M) SC: O(1)

Ideas:





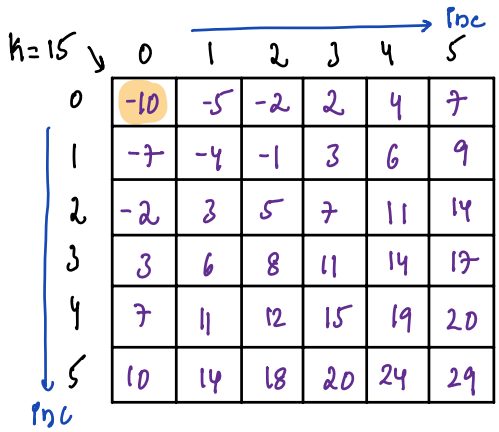





```
boolean matSearch (int mat[][], int k){
    int N = mat.length;
    int M = mat[0].length;
    int i=0, j= M-1
    while( i<N && j>=0){
        if(mat[i][j]==k){
            return True;
        }
        else if(mat[i][j] < k){ // skip row go down
            i = i+1
        }
        else { // skip col go left
            j = j-1
        }
    }
    return false;
}
```

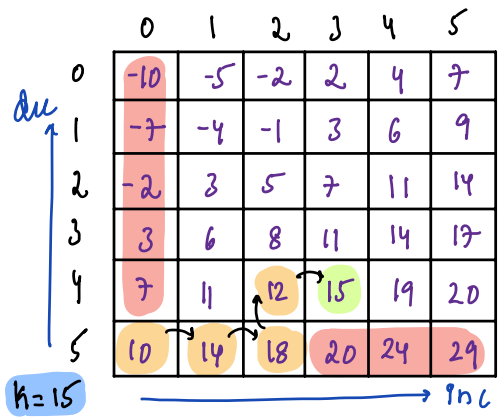
TC: Total Rows: N
Total Cols: M
In each iteration:
We skip 1 row or 1 col
Total Iterations: N+M

Will same approach work from other corners as well?



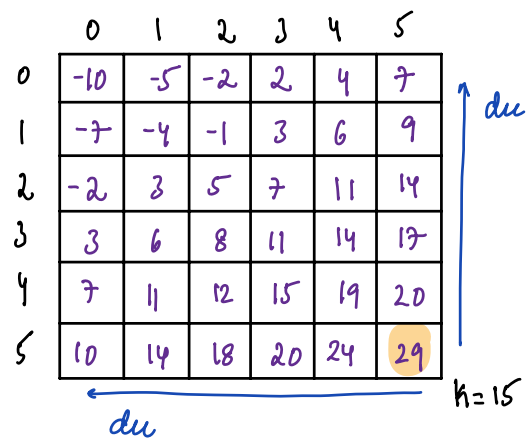
	0	1	2	3	4	5
0	-10	-5	-2	2	4	7
1	-7	-4	-1	3	6	9
2	-2	3	5	7	11	14
3	3	6	8	11	14	17
4	7	11	12	15	19	20
5	10	14	18	20	24	29

Top left ✗



	0	1	2	3	4	5
0	-10	-5	-2	2	4	7
1	-7	-4	-1	3	6	9
2	-2	3	5	7	11	14
3	3	6	8	11	14	17
4	7	11	12	15	19	20
5	10	14	18	20	24	29

Bottom left ✓



	0	1	2	3	4	5
0	-10	-5	-2	2	4	7
1	-7	-4	-1	3	6	9
2	-2	3	5	7	11	14
3	3	6	8	11	14	17
4	7	11	12	15	19	20
5	10	14	18	20	24	29

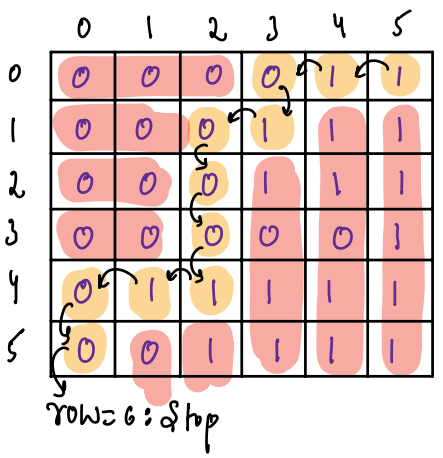
38) Given a binary 2D matrix mat[N][M] every row is sorted.

Find smallest column number which contains atleast single 1

If No such column exist: Return M

Q: Return Max no: of 1's present in a row :

Eg:

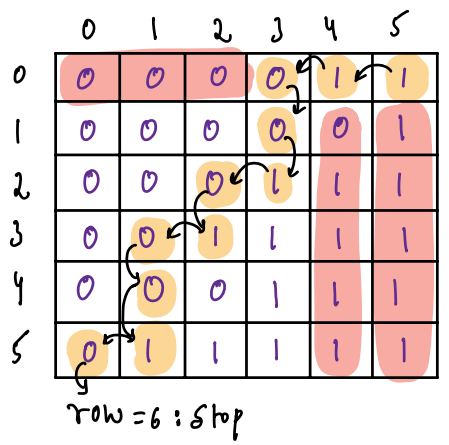


ans = 5 → 4 → 3 → 2 → 1

Q: Max 1's present in a row :

: return M-ans = 6-1 = 5

Eg2:



ans = M = 6 → 5 → 4 → 3 → 2 → 1

Q: Max 1's present in a row :

: return M-ans = 6-1 = 5

Ideas:

1. Iterate on all columns & return 1st column which contains 1

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

2. ans = M;
   i=0, j = M-1;

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

```
while ( i<N && j>=0){
    if(mat[i][j]==1) { ans = j; j--;}
    else { i++;}
}
return ans; // smallest col number contains 1
return M-ans; // Max 1's present in a row.
```