2D Arrays

Real life?

- Image process → RGB values, pixes (Matlab)
- → Chessboard → 8×8 → '8' 1-D array.
- → Sudoku
- → Tic Tac Toe
- → Minesweep etc.

lots of game (Tetris, snake etc.)

Storage

a[N]

a[N][M] N=3

rows colums

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

M=4

a[i][j] i → row. j → col.

Access → O(1)

Row major [C/C++ etc.]

A[a] &a[0][0] RAM

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

RAM
1
2
3
4
5
6
7
8
9
⋮

M, M, NM

A[i][j]

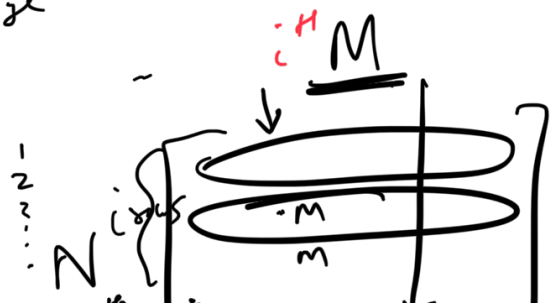
A[2][1]

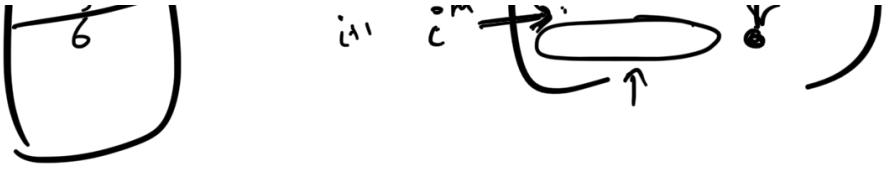
i=2 3rd row

1 2 3 4 5 6 7 8 9

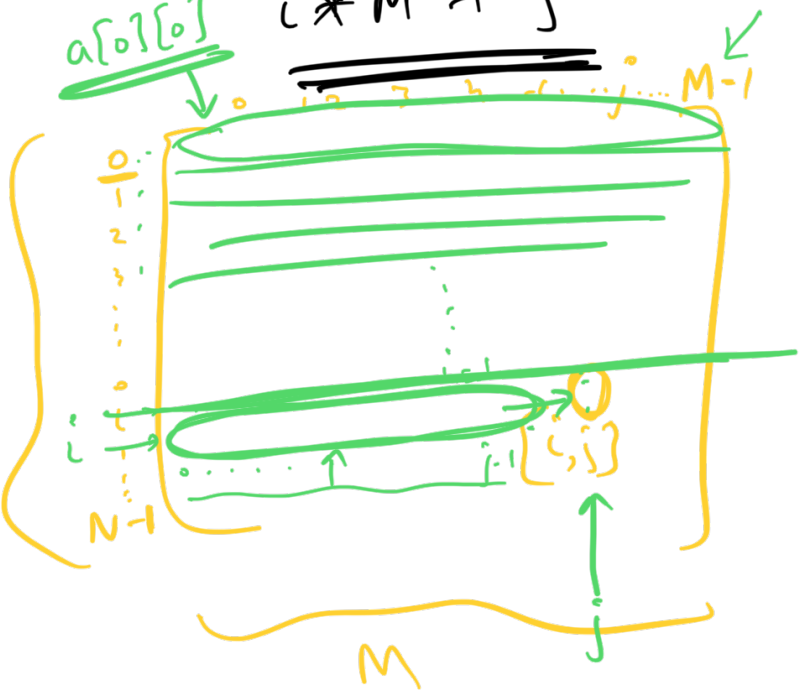
2*3+1=6+1=7

A[6] &a[0]+ 6*size





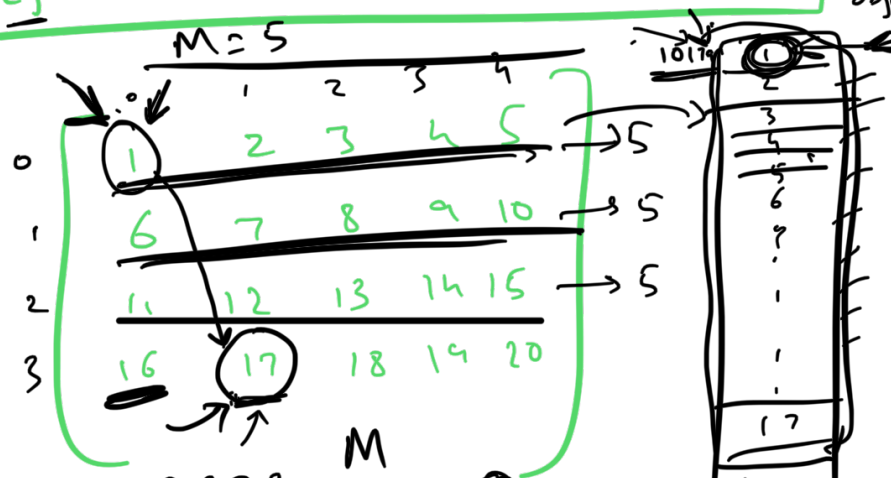
$i * M + j$



row major

$i \text{ rows} \times M + j$

$$\boxed{a[i][j] = \&a[0][0] + (i * M + j) * \text{size of each obj}}$$



$$\&a[3][1] = \&a[0][0] + 3 \times 5 + 1 \times 4 \text{ bytes}$$

$$= \&a[0][0] + 16$$

Column major

[Matlab] R etc.

$$N = \begin{array}{c|ccc} i: & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 2 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

[N][M]

$$j * N + i$$

Contigu.

1
4
7
2
5
8
3
6
9

No row-major/ col-major. (java)

M

N

a[0]

a[1] 0

a[2] 0

a[i][j]

A[100][2]

(Jagged array)

access O(1)

a[i][j] →

Total Memory → $N \times M + N$

Jagged

M

N

a[0] → 0 0 0 0

a[1] →

a[2] →

a[N-1]

a[2][3]

a[8] a → &a[0]

$O(NM + N)$ a[2][3]

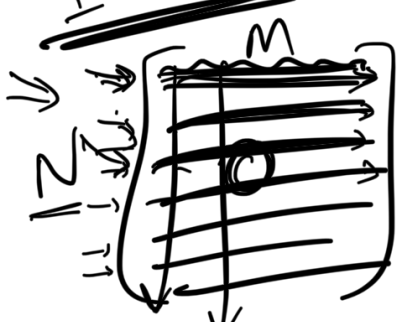
aref[2] ⇒ &aref[0] + 2

&aref a[8]

a[3] ⇒ a + 3*size

&a[3]

Iterate



el → (NM).

```
for (i=0; i<n; i++) → O(n
{
    for (j=0; j<m; j++)
    {
        print a[i][j]
    }
}
```

print al

T.C ⇒ O(N*M)

S.C ⇒ O(N*M)

1	2	3
4	5	6

1 2 3 4 5 — — .

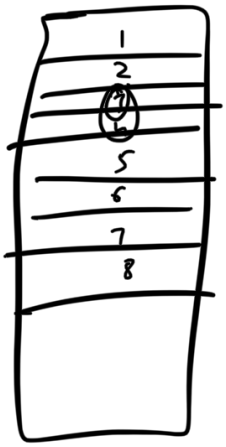
1 4 7 2 5 8

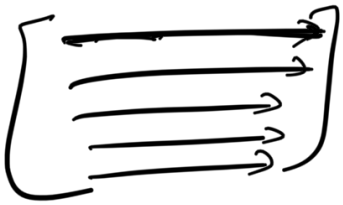
```
for(j=0; j<m; j++)
{
    for(i=0; i<n; i++)
    {
        print a[i][j]
    }
}
```

Locality of reference

- spatial → cache neighboring element
- temporal → same element is accessed later

(row major)





access

(Cache > RAM > HDD)
reg

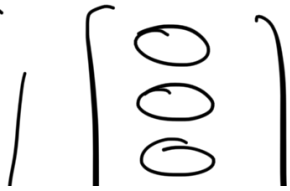
Most voted on stack overflow → Branch prediction (how compiler optimizes!)

(Why sorted array execution unsor

(faster)

a[2][100] ← (spatial locality)

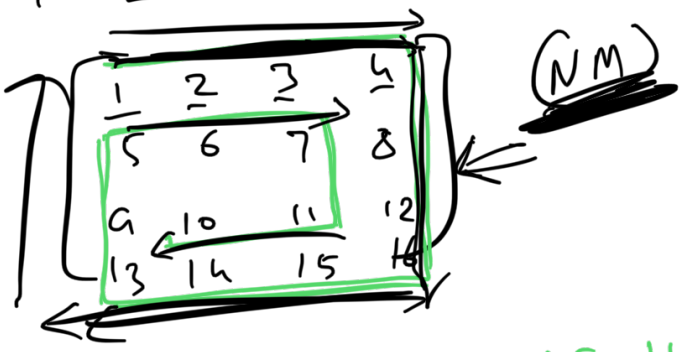
a[100][2]



100

Q.1 print matrix in spiral way!

Amazon
Microsoft
Adobe
FlipKart



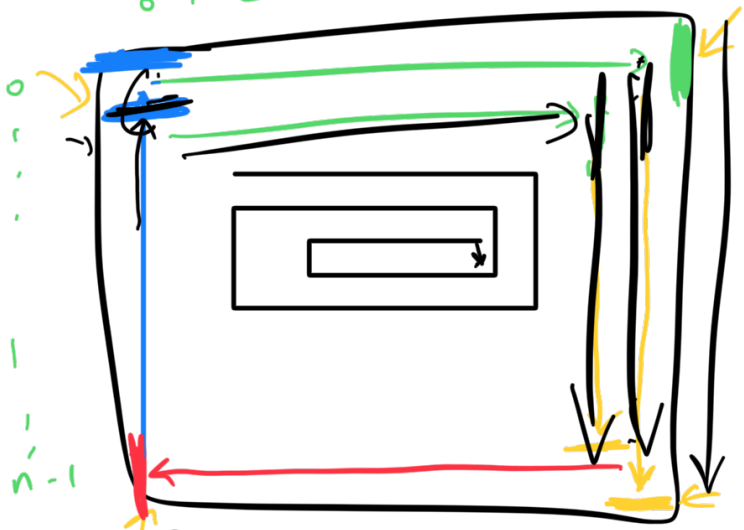
(N M)

(N×M)

Ans

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

iteration is clear?



Top = 0

Right = m-1

Bottom = n-1

Left = 0

While (_)

{ Once 1st row is printed,

Top++

row is printed.

Once last colu is p

Right--;

Once last row is printed;

Bottom--;

Once 1st colum is printed;

Left++.

H.W.

While (Left ≤ Right && Top ≤ Bottom

{

for(i = left; i ≤ right; i++) → (ON)

print a[Top][i]

Top++;

for(i = Top; i ≤ Bottom; i++) → 0

print a[i][right]

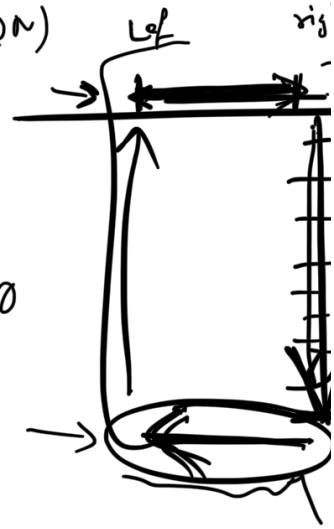
Right--;

for(i = right; i ≥ left; i--)

print a[bottom][i]

bottom--;

}



N = M

N = 10

of 'N' dim

Q.2 Rotate "square" matrix by 90°.

Amazon

1	2	3

→

7	4	1

$$\begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow \begin{bmatrix} 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$$

$\Downarrow$ Transpose

Tra $\rightarrow$ $A[i][j] \leftrightarrow A[j][i]$

reverse

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

$O(N)$

$O(N)$

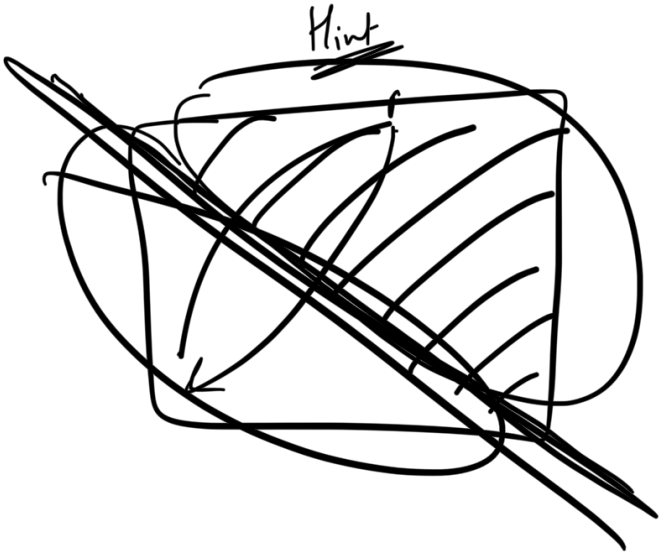
$O(N)$

reverse of each row

$O(N \times M) \Rightarrow O(N^2)$

Brute: new 2-D array $\Rightarrow O(NM)$ extra space

Hint



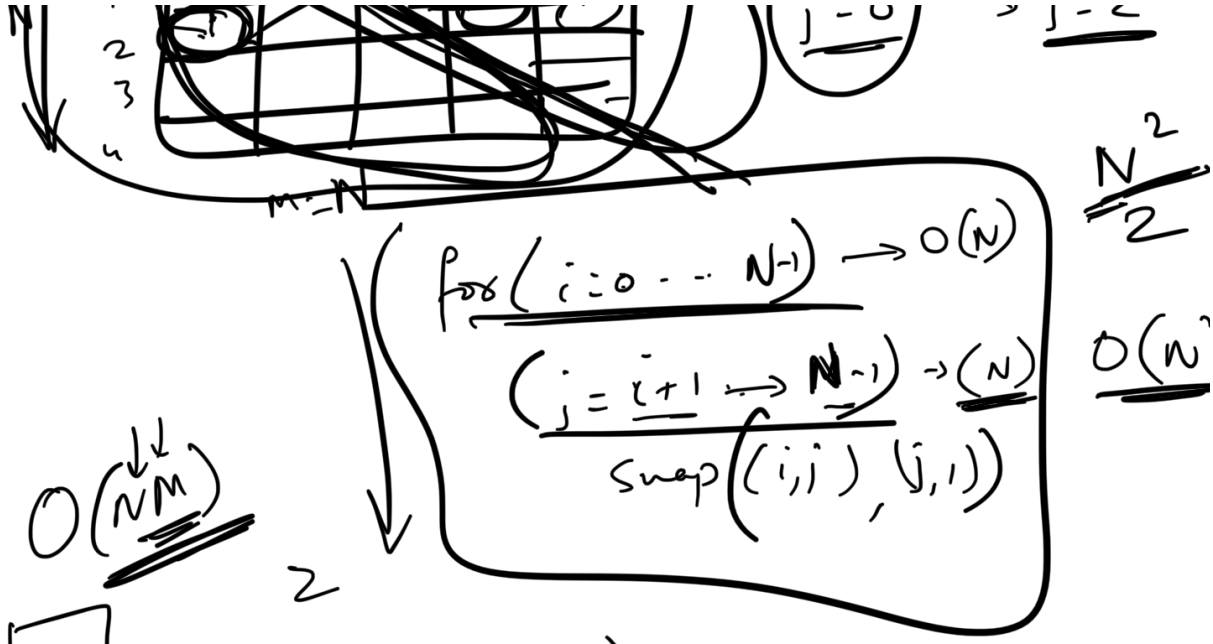
If you travel the entire matrix

↳ you get the original matrix

swap(a[i][j], a[j][i])

$i=2$ $\Rightarrow$ $i=0$

$i=2$

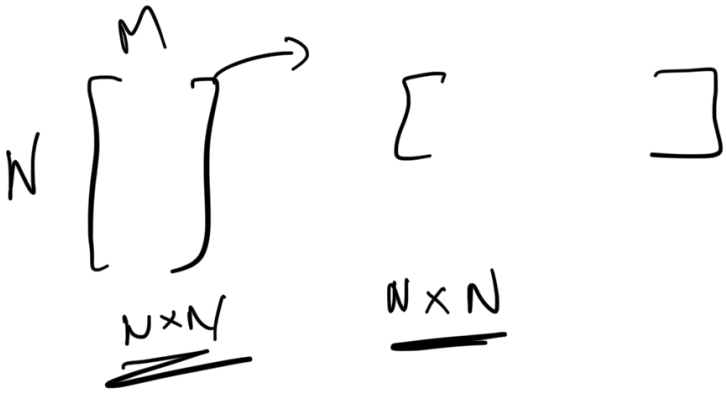


$O(NM)$

N^2

T.C ⇒ $O(N^2) + O(N^2)$

$\Rightarrow O(N^2)$.

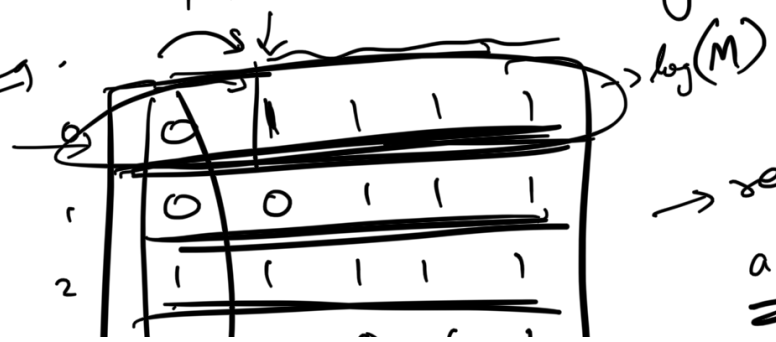


Q.3 Given matrix of 0's & 1's (N×

Amazon ↳ all rows are sorted.

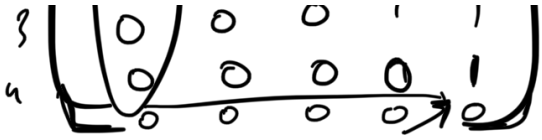
Find row having maximum no. of 1

eg.



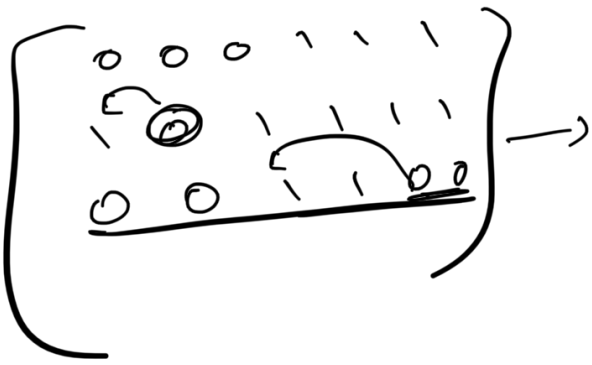
→ return 2;

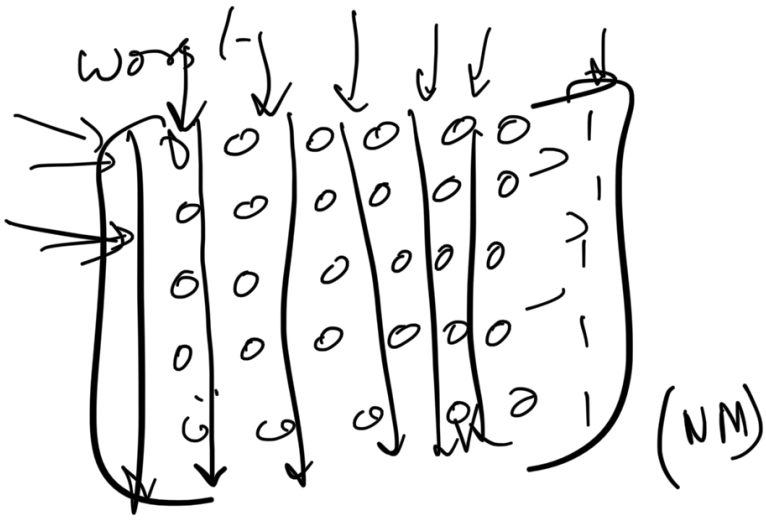
a[2] =



① **Brute** → Traverse each row & find total 1's → max (...)

T.C ⇒ O(N*M)

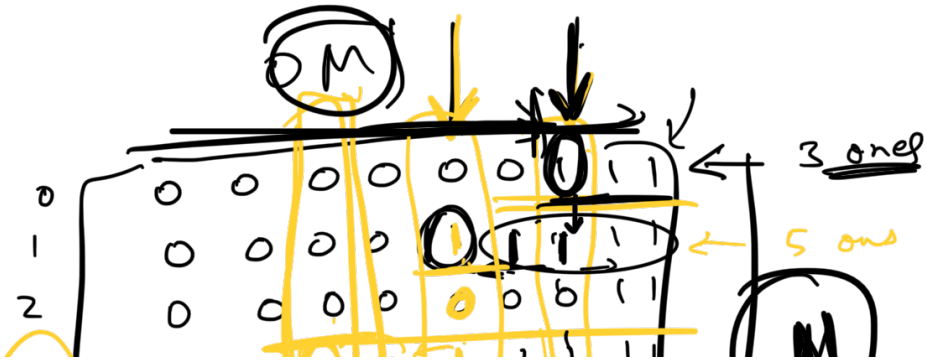


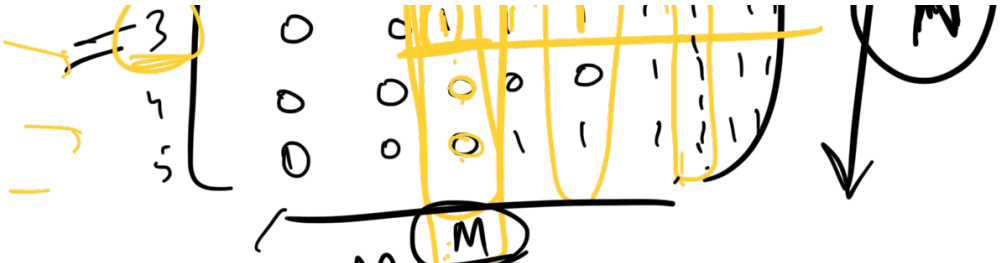


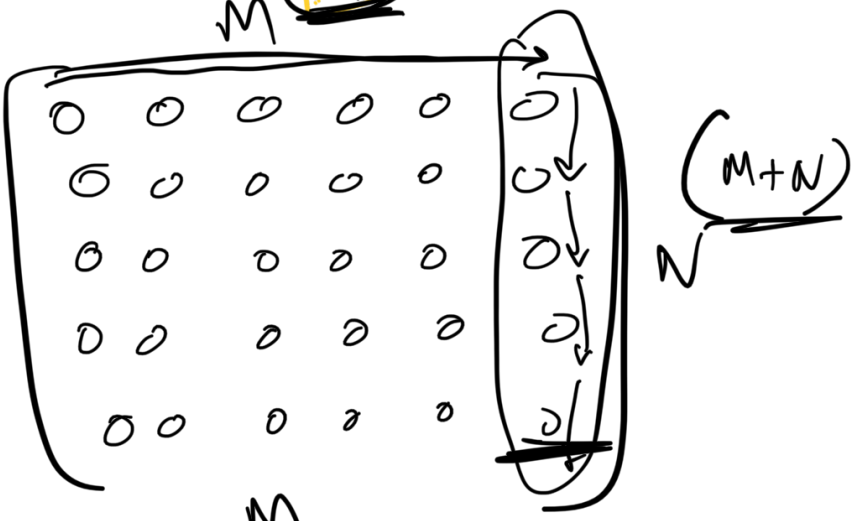
② **Binary Search**

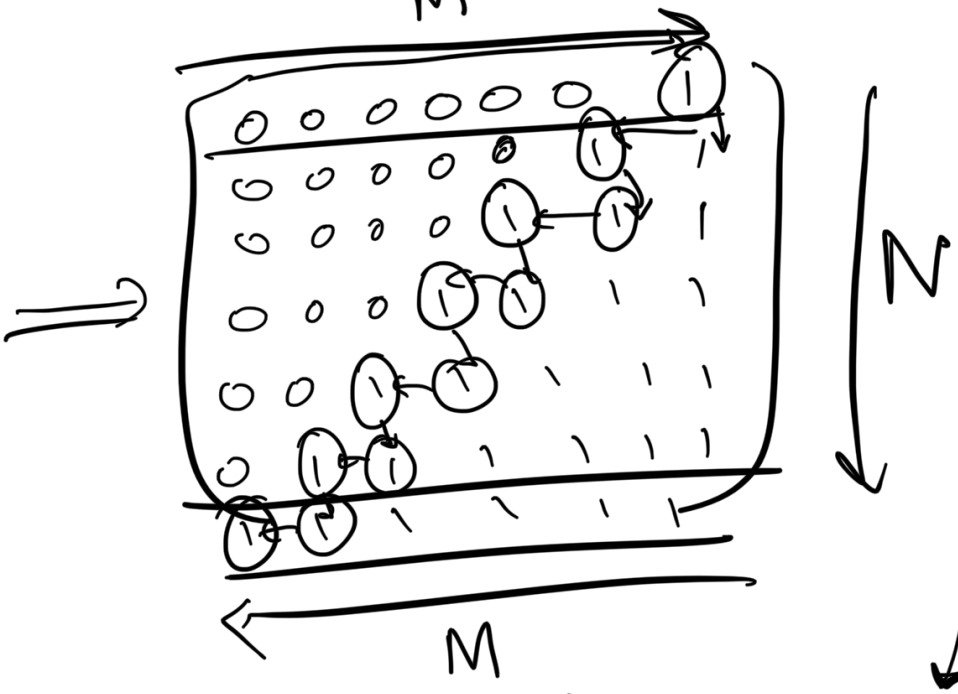
T.C ⇒ O(N log M)

③ **Obs:**

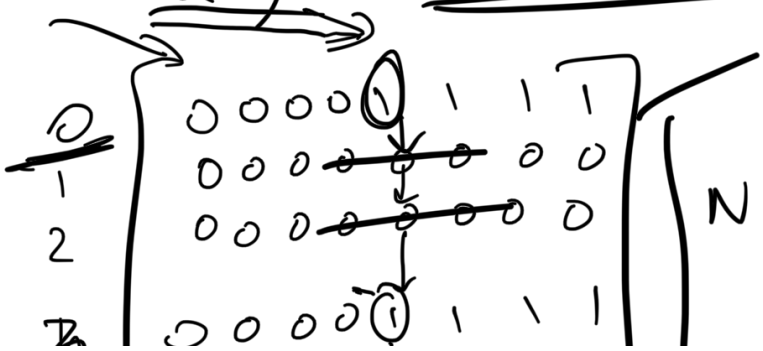


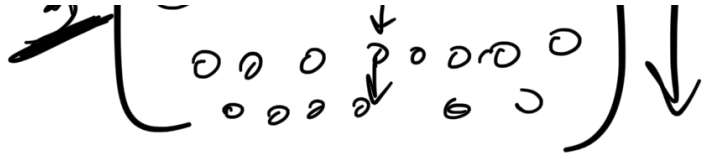






$$O(2M+N) \rightarrow O(M+N)$$





A[100][2]

A[2][100]

HW Given matrix, unique elements

Google Amazon

↳ all rows are sorted

↳ all columns are sorted.

Search for a number.

	0	1	2	3	4
0	1	3	5	7	10
1	2	7	10	15	20
2	5	12	18	31	42
3	17	19	29	34	56

34 ?

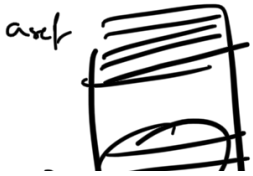
(3, 3)

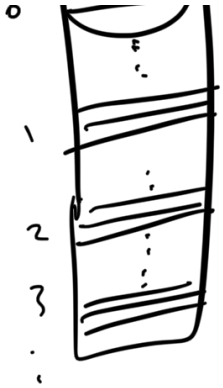
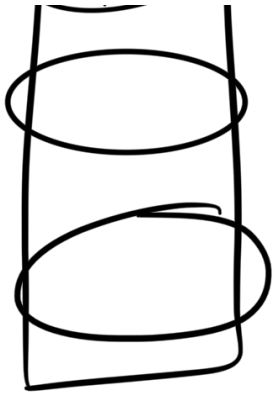
70? → No (-1

A[100] [2]

Jagged

A[2] [100]




0
1
2
3