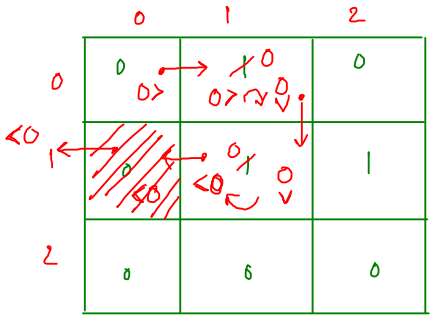
★ Find the way :




★ 0 → forward

1 → make it 0, turn right, move forward

0> (right) = 0

0 (down) = 1

<0 (left) = 2

^ 0 (top) = 3

op: (1, 0)

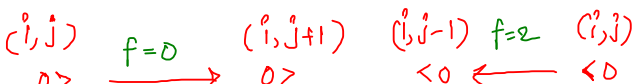
The last cell before coming out of matrix


# How to know the facing of mouse ?

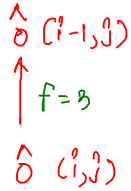
→ maintain a facing variable (0, 1, 2, 3)

Initial, facing = 0

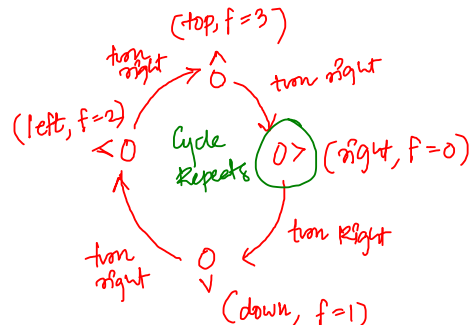
# How to move forward ?








# How to turn right !




f=0 → f=1
f=1 → f=2
f=2 → f=3
f=3 → f=0

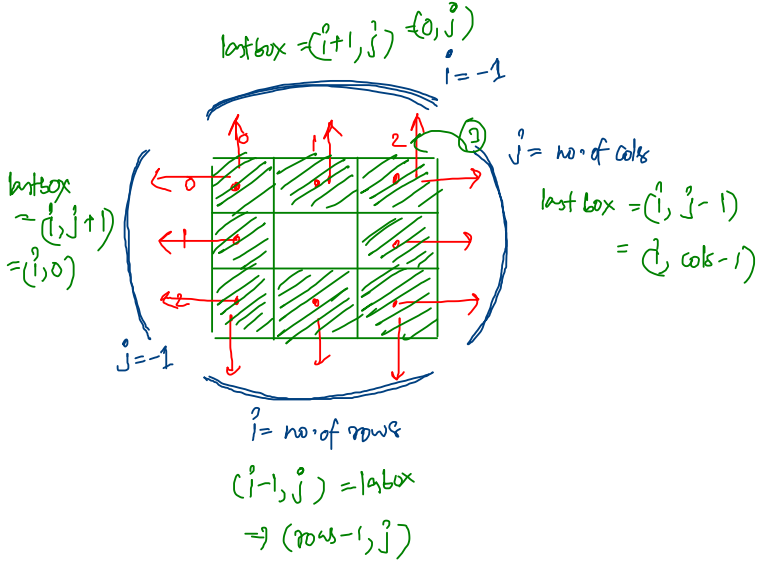
3++ → 4 ✗

~~for(int p=0; p<3; p++)~~

* f = (f+1) % 4;

(0+1) % 4 ⇒ 1
(1+1) % 4 ⇒ 2
(2+1) % 4 ⇒ 3
(3+1) % 4 ⇒ 0

# How to check that you are out :



## ★ Maxima / Minima ;

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

★ Ele which is minimum in its row and maximum in its column.

op: 7

minR0 = 1 ① ✗
maxC0 = 7

minR2 = 7 ① ✓
maxC0 = 7

minR0 = 1 ② ✗
maxC1 = 8

★ SC: O(1)

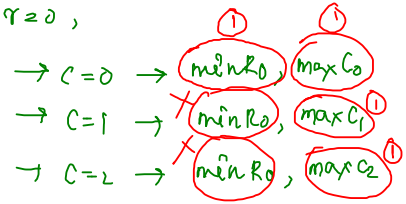
★ TC: O(rows * cols * (rows + cols))
= O(N * N * 2N) = O(N³)

```
for(let r=0; r<rows; r++) {
    for(let c=0; c<cols; c++) {
        const minR = findMinRow
                     (matrix, r)
        const maxC = findMaxCol
                     (matrix, c)
        if(matrix[r][c] == minR &&
           matrix[r][c] == maxC) {
            return matrix[r][c];
        }
    }
}
return -1;
```

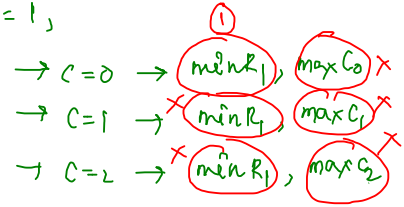
# Optimize :

3 x 3 matrix,

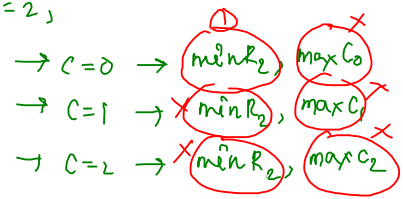
① r = 0,




② r = 1,




② r = 2,




⇒ We need to avoid redundant calculations, By storing the results.

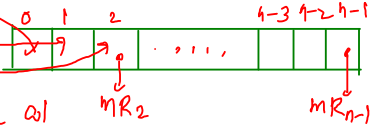
① Compute min in each row and store

$minR_0 = 1$

$minR_1 = 4$

$minR_2 = 7$

What if N Rows how to store?




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

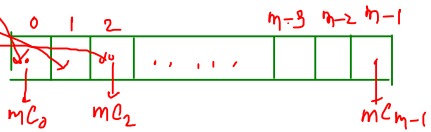
② Compute max in each col and store

$maxC_0 = 7$

$maxC_1 = 8$

$maxC_2 = 9$

What if M cols how to store?




r = 0

→ c = 0, $minR_0$ = 1, $maxC_0$ = 7

★ Now at r, c

earlier,

```
const minRow = findMinRow();
const maxCol = findMaxCol();
```

Now, need not calculate already we done it once and stored.

PRECOMPUTATION TECHNIQUE

Calculate once use many times

```
1578  function maximaMinima(matrix) {
1579    //Write code here
1580    const rows = matrix.length;
1581    const cols = matrix[0].length;
1582
1583    // 1. precomputation
1584    const minRow = [];
1585    for (let r = 0; r < rows; r++) {
1586      const val = findMinRow(matrix, r);
1587      minRow.push(val);
1588    }
1589
1590    const maxCol = [];
1591    for (let c = 0; c < cols; c++) {
1592      const val = findMaxCol(matrix, c);
1593      maxCol.push(val);
1594    }
1595
1596    // 2. problem solution, utilize the precomputed results
1597    for (let r = 0; r < rows; r++) {
1598      for (let c = 0; c < cols; c++) {
1599        const minR = minRow[r];
1600        const maxC = maxCol[c];
1601        if (matrix[r][c] == minR && matrix[r][c] == maxC) {
1602          return matrix[r][c];
1603        }
1604      }
1605    }
1606
1607    return -1;
1608  }
```

O(Rows * Cols)

+

O(Cols + Rows)

+

O(Rows * Cols)

O(1)

findMinRow() → O(cols)

findMaxCol() → O(rows)

TC : O(3 * Rows + Cols)

O(Row * cols)

$O(N^2)$

SC : we are using two additional arrays

minRow[] → O(rows)

maxCol[] → O(cols)

⇒ O(rows + cols)

⇒ O(N + N) ⇒ O(N)

1. running stream → sum, max, subarrays (used for many problems)

2. Generating subarrays, Generating pairs, triplets, ....

3. Matrix Multiplication, transpose, rotation (useful in ML)

4. Functional way of writing codes (do not write all code in a single function, split into smaller tasks)

5. Simpler way of finding Time Complexity (No. of Iterations, tip $1 \to n$, $n \to 1$, $O(n)$; $n \to \frac{n}{2}$, $\frac{n}{2} \to 1$, $O(\log_2 n)$)

6. Precomputation Technique → sum, max, min (used for many problems)

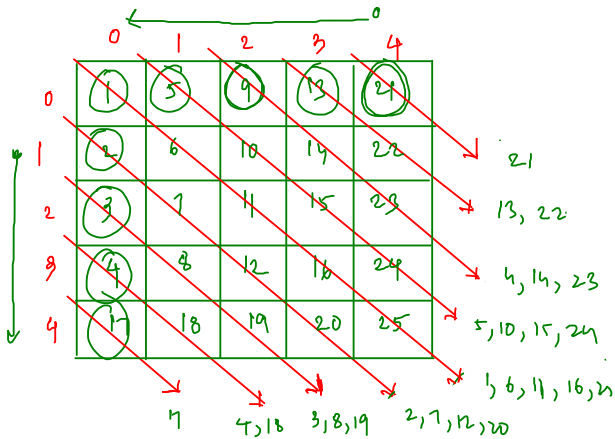
7. Two pointer → reverse, palindrome

8. Digit Extraction / Iteration → num%10 → extract; num = num/10 → remove

9. Some behind the Scenes of JS (Execution, hoisting, process)

10. behind the scenes of Arrays (stack, heap, pass by val, pass by ref)

## ★ Diagonal traversal :




Op: 21 13 22 4 14 23 . . --

★ go to every starting point of diagonal

★ follow the diagonal traversal

r = 0 1 X
c = 4 5

r = 0 1
c = 3 4 5

r = 0 1 2 3 X
c = 2 3 4 5

r = 0 1 2 3 4 X
c = X X X X 5

One sentence / word about the module → positive / negative
any Feedback

* pace ↓

⇒ 1 month into coding
(Fast) → 3-6 months

⇒ Qs, (think how a normal solver it on a paper)

practice, practice, practice ⇓ Code (some of the problems require techniques)
⇓
..... TC optimization (Techniques)