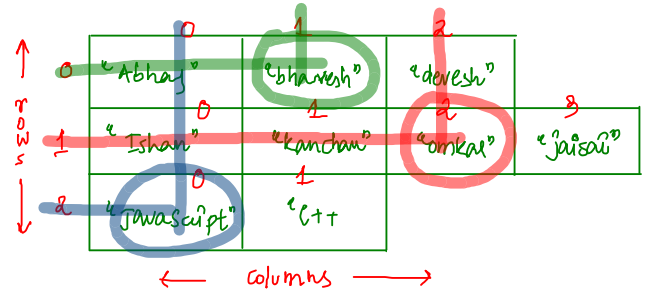
* Array of Arrays (2D Arrays):

```
const friends1 = ["abhay", "bhavesh", "devesh"];
const friends2 = ["Ishan", "kanchan", "omkar", "jaisai"];
const friends3 = ["Javascript", "C++"];

const allFriends = [friends1, friends2, friends3];
console.log(allFriends);
console.log(allFriends[0]);
console.log(allFriends[1]);
console.log(allFriends[2]);
```

Matrix Representation



allFriends =

[["abhay", "bhavesh", "devesh"], ["Ishan", "kanchan", "omkar", "jaisai"], ["Javascript", "C++"]]

0 1 2 3 0 1 0 1 2

allFriends[1][2] → "omkar"

allFriends[2][0] → "Javascript"

1st row, 2nd col ⇒ allFriends[1][2] ⇒ "omkar"

2nd row, 0th col ⇒ allFriends[2][0] ⇒ "Javascript"

0th row, 1st col ⇒ allFriends[0][1] ⇒ "bhavesh"

Matrix Representation

=) allFriends

	0	1	2	3
0	"Abhay"	"bhavesh"	"devesh"	
1	"Ishan"	"kanchan"	"omkar"	"jaisai"
2	"Javascript"	"C++"		

rows ↓ ↑

columns ← →

* In this example, every row has different no. of columns hence it is called a "Jagged Array".
* In general, no. of cols will be same in every row.

Q1: Find no. of rows

A: no. of rows = 3

= no. of smaller Arrays

= no. of elements in allFriends Array

= allFriends.length

Q2: Find no. of cols

A: no. of cols = no. of elements in a smaller Array

= no. of elements in a row

row-0 ⇒ allFriends[0].length = 3

row-1 ⇒ allFriends[1].length = 4

row-2 ⇒ allFriends[2].length = 2

allFriends[row].length

Q3: Given a n x m matrix, print it as follows

rows, cols

Ip: n = 3, m = 3

	0	1	2
0	10	11	12
1	15	13	14
2	16	17	18

3 x 3

op: 10 11 12

15 13 14

16 17 18

cols = matrix[0].length → 3

for (let c = 0; c < cols; c++) {

psw(matrix[0][c]);

}

2

① c = 0,
→ matrix[0][0]

② c = 1,
→ matrix[0][1]

③ c = 2,
→ matrix[0][2]

④ c = 3 x (3 < 3)

Q4 : Create a N x m matrix in the following pattern

eg: ip : n = 3, m = 4

op:

	0	1	2	3
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4

mat = [[1,2,3,4], [1,2,3,4], [1,2,3,4]]

→ To build an entire matrix,

= first you need to build smaller arrays

```
const arr1 = [1,2,3,4];
const arr2 = [1,2,3,4];
const arr3 = [1,2,3,4];
```

① build smaller arrays

```
const mat = [arr1, arr2, arr3];
```

② store every thing in bigger Array

eg: n = 4, m = 4

	0	1	2	3
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	1	2	3	4

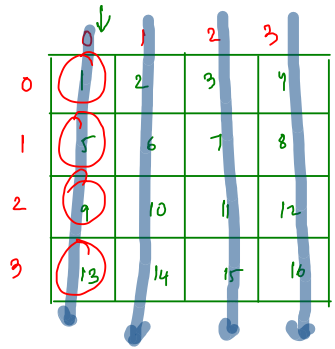
```
function buildMatrix(rows, cols) {
  const matrix = [];
  for (let r = 0; r < rows; r++) {
    // 1. make smaller array
    const smallArr = [];
    for (let c = 0; c < cols; c++) {
      smallArr.push(c + 1);
    }

    // 2. add it to bigger array (matrix)
    matrix.push(smallArr);
  }

  return matrix;
}
```

* print column wise :

Eg :




```
for(let c = 0; c < cols; c++) {
    for(let r = 0; r < rows; r++) {
        psw( mat[r][c] + " " );
    }
}
```

① c = 0
- → r = 0, m[0][0]
- → r = 1, m[1][0]
- → r = 2, m[2][0]
- → r = 3, m[3][0]

② c = 1
- → r = 0, m[0][1]
- → r = 1, m[1][1]
- → r = 2, m[2][1]
- → r = 3, m[3][1]

op :

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

→ go to every row, → go to every col

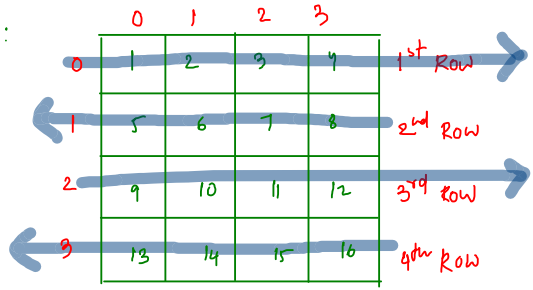
→ go to every col → go to every row

* Alternate Matrix Traversal :

odd Rows
(L → R)
1st, 3rd etc...

even Rows
(R → L)
2nd Row, 4th Row etc..

Eg :




```
function printElementsAlternately(mat, m, n) {
  // M rows and N columns
  for (let r = 0; r < m; r++) {
    if (r % 2 == 0) {
      for (let c = 0; c < n; c++) {
        process.stdout.write(mat[r][c] + " ");
      }
    } else {
      for (let c = n - 1; c >= 0; c--) {
        process.stdout.write(mat[r][c] + " ");
      }
    }
  }
}
```

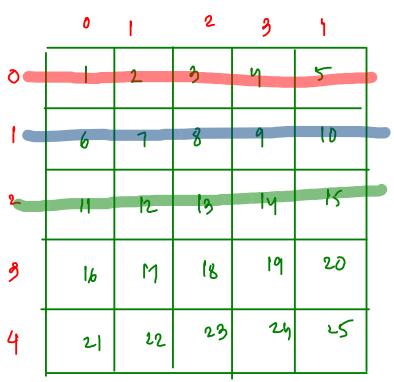
Op; 1 2 3 4 8 7 6 5 9 10 11 12 13 14 15 16

row_idx even ⇒ L → R
row_idx odd ⇒ R → L

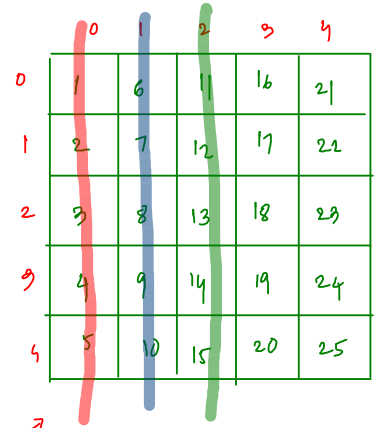
① r = 0
→ r%2 == 0
0%2 == 0 ✓
→ c = 0 1 2 3 ✗

② r = 1
→ r%2 == 0
1%2 == 0 ✗
→ c = 3 2 1 0 ✗

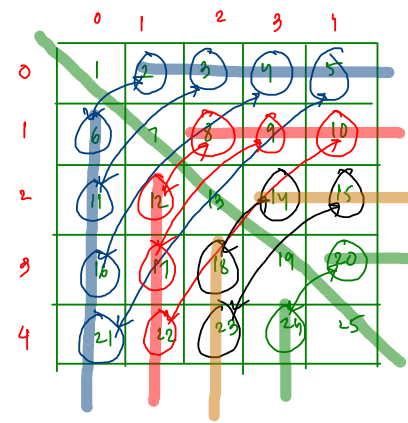
* Transpose of a Matrix : [In-place]



⇒ Interchanging row ⟷ col's



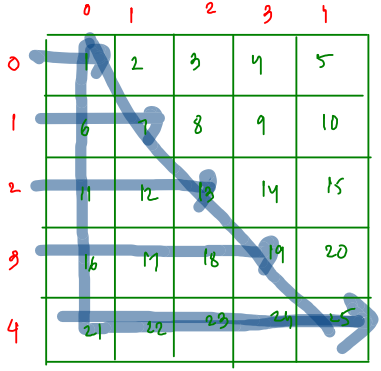
transpose



Code should perform all these swaps.

r c c r

m(1)(0) ⟷ m(0)(1)
m(2)(0) ⟷ m(0)(2)
m(3)(1) ⟷ m(1)(3)
m(3)(0) ⟷ m(0)(3)
m(4)(3) ⟷ m(3)(4)
m(3)(1) ⟷ m(1)(3)
m(3)(2) ⟷ m(2)(3)
m(4)(0) ⟷ m(0)(4)
m(4)(1) ⟷ m(1)(4)
m(4)(2) ⟷ m(2)(4)



r = 0 , c = 0

r = 1, c = 0, 1

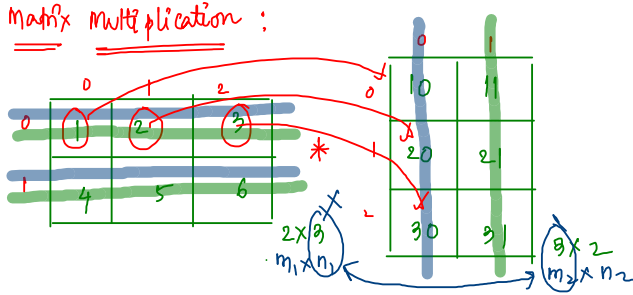
r = 2, c = 0, 1, 2

r = 3, c = 0, 1, 2, 3

r = 4, c = 0, 1, 2, 3, 4

```
for (let r = 0; r < rows; r++) {
    for (let c = 0; c < r+1; c++) {
        let temp = mat(r)(c);
        mat(r)(c) = mat(c)(r);
        mat(c)(r) = temp;
    }
}
```

★ Matrix multiplication :



	0	1
0	1×10 + 2×20 + 3×30 = 140	1×11 + 2×21 + 3×21 = 146
1	4×10 + 5×20 + 6×30 = 320	4×11 + 5×51 + 6×31 = 335

	0	1
0	140	146
1	320	335

2 x 2

1. you can multiply A ($m_1 \times n_1$) matrix and B ($m_2 \times n_2$) matrix if and only if $n_1 == m_2$ i.e; no. of cols in A (n_1) == no. of rows in B (m_2)

2. The resultant matrix will be of size, $m_1 \times n_2$

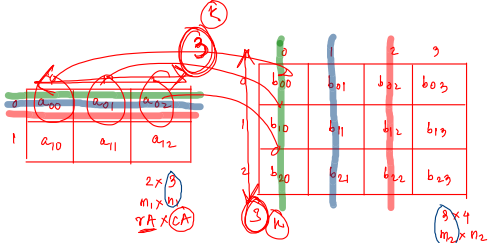
$(2 \times \boxed{3}) * (\boxed{3} \times 2) \Rightarrow 2 \times 2$

★ TC = O(rowsA * colsB * colsA) (rowsB)

= $O(m_1 * n_2 * n_1)$ ($n_1 = m_2$)

If N x N matrix, $m_1 = n_1 = m_2 = n_2 = N$

= $O(N^3)$



$c_{00} = a_{00} \cdot b_{00} + a_{01} \cdot b_{10} + a_{02} \cdot b_{20}$

$c_{01} = a_{00} \cdot b_{01} + a_{01} \cdot b_{11} + a_{02} \cdot b_{21}$

$c_{02} = a_{00} \cdot b_{02} + a_{01} \cdot b_{12} + a_{02} \cdot b_{22}$

$c[r][c] = sum(a[r][?] * b[?][c])$

$c[1][2] = a[1][0] * b[0][2] + a[1][1] * b[1][2] + a[1][2] * b[2][2]$

0, 1, 2

how to know this?

⇒ no. of cols in A or no. of rows in B

$$c[r][c] = \sum_{k=0}^{\text{numrowsB}-1 \,=\, \text{numcolsA}-1} A[r][k] * B[k][c]$$

$$c[1][2] = \sum_{k=0}^{3-1=2} A(1)(k) * B(k)(2)$$

$$= A(1)(0) * B(0)(2) + A(1)(1) * B(1)(2) + A(1)(2) * B(2)(2)$$

$k=0$ $\quad k=1$ $\quad k=2$