2D Array

Act
as if what
you do makes
a difference.
It does.
—WILLIAM JAMES

Good
Morning ☺

starting at 7:05

Content

01. Matrix basics
02. Print row wise sum
03. Print col wise sum
04. Print diagonal of sq. matrix
05. Print all diagonal of rectangular matrix
06. Transpose of matrix
07. Rotate matrix by 90°

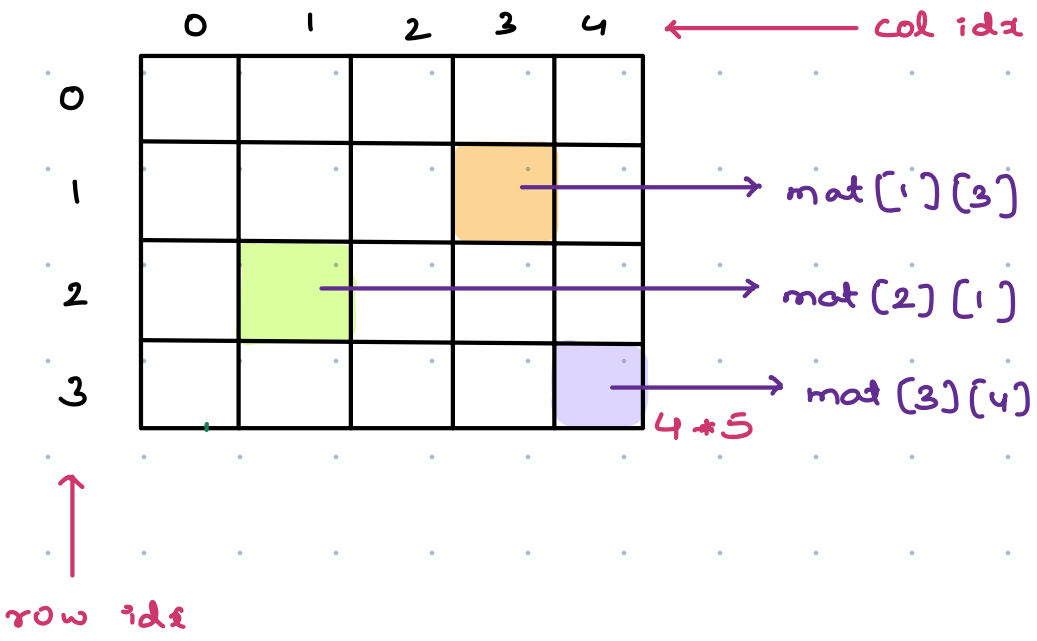
Matrix / 2D Array → Data in the grid format

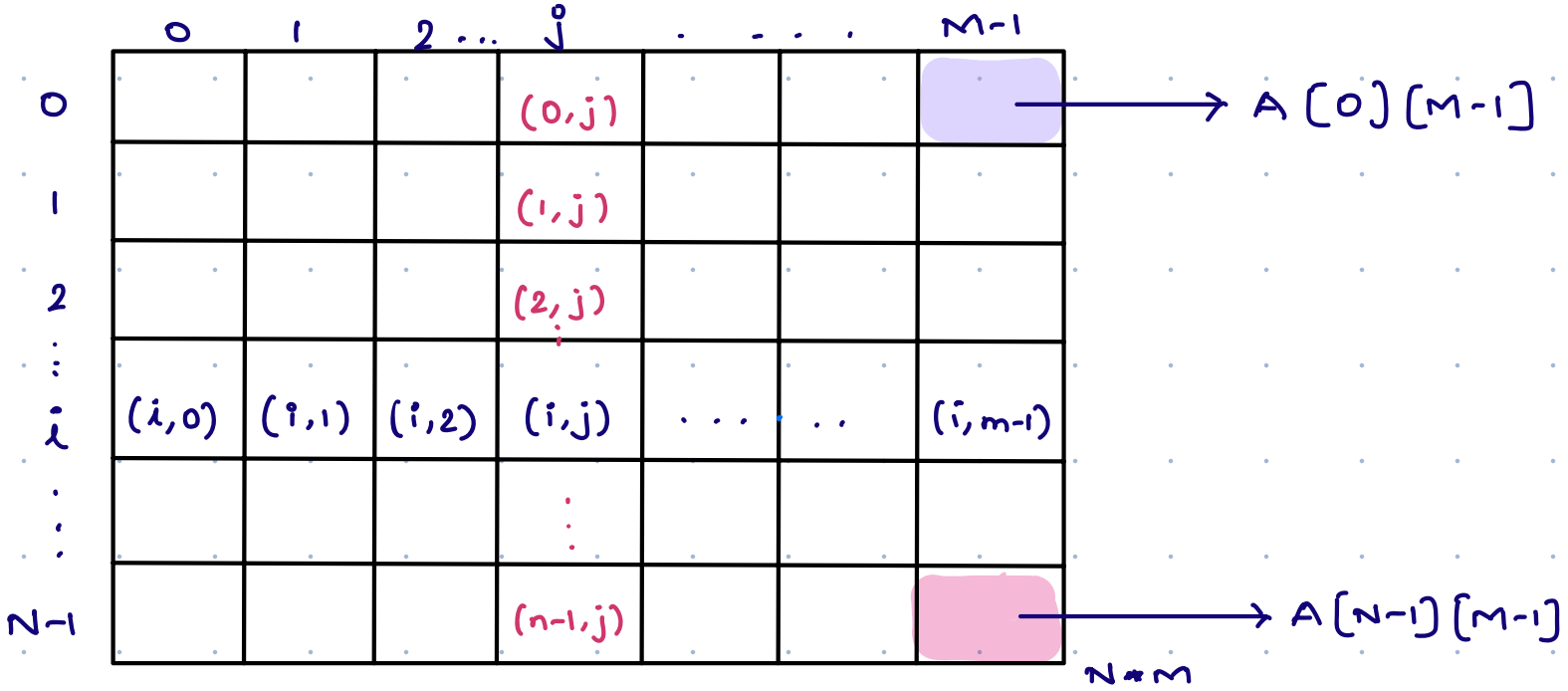
→ Data represented in form of tables

→ Rows & Columns

Delare → int mat[N][M]

Datatype — Name of matrix — rows — columns





If we want to move on i' row

↳ col will change from 0 to m-1

If we want to move on j* col

↳ row will change from 0 to n-1

Q Given mat[N][M], print row wise sum

	0	1	2	3	sum
0	4	3	1	7	15
1	6	2	3	4	15
2	5	3	2	7	17

Idea → Iterate on a particular row & calculate the sum.

Row = 0 to (n-1)

```
void printrowsum(int [][] A)
{
    int N = A.length;      // rows
    int M = A[0].length;   // cols

    for(r=0; r<N; r++){
        int sum=0
        for(c=0; c<M; c++){
            sum = sum + A[r][c];
        }
        println(sum);
    }
}
```

TC: O(N*M)

SC: O(1)

* Not optimal but another way

```
void printrowsum(int [][] A)
    int N = A.length;      // rows
    int M = A[0].length;   // cols
    int [] ans = new int[n]
    for(r=0; r<N; r++){
        int sum=0
        for(c=0; c<M; c++){
            sum = sum + A[r][c];
        }
        ans[r] = sum;
    }

    for(i=0; i<ans.length; i++){
        println(Ans[i]);
    }
}
```

TC: O(N*M)

SC: O(N)

Q Given mat[N][M], print col wise sum

	0	1	2	3
0	4	3	1	7
1	6	2	3	4
2	5	3	2	7
sum	15	8	6	18

```
void printColSum(int[][] A)
    int N = A.length;    // rows
    int M = A[0].length; // cols
    for (c=0; c<M; c++){
        int sum=0
        for(r=0; r<N; r++){
            sum = sum + A[r][c];
        }
        println(sum);
    }
```

TC : O(N*M)

SC : O(1)

03 Given a square matrix, print its diagonal.

→ Principal Diagonal → Top left to bottom right

→ Anti Diagonal → Top right to Bottom left corner

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

Principal diagonal

0	0
1	1
2	2
3	3
4	4 → stop

```
for ( i = 0 ⟶ n )
    for ( j = 0 ⟶ n ) {
        if ( i == j ) print A[i][j] ;
    }
}
```

TC : O(N*M)

Principal diagonal

```
i = 0 , j = 0

while ( i < N && j < N)
    println ( A[i][j]
    i++;
    j++;
}
```

TC : O(n)

SC : O(1)

Anti - diagonal

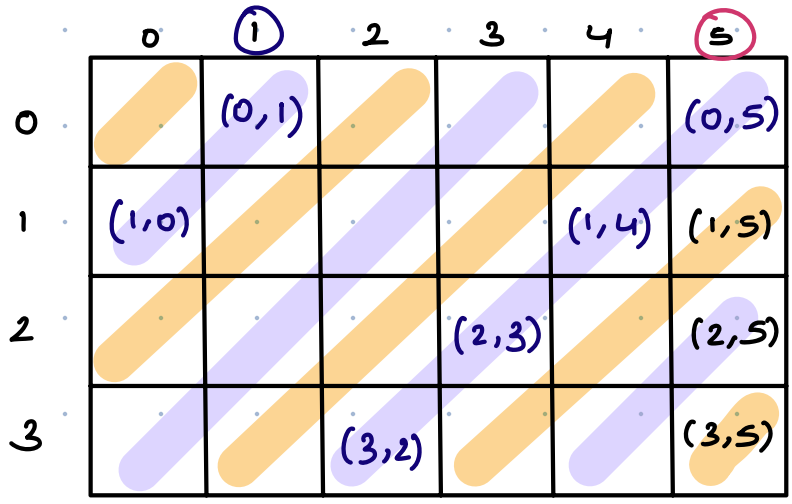
```
i = 0 , j = n-1

while ( i < N && j ≥ 0)
    println ( A[i][j])
    i++;
    j--;
}
```

Anti diagonal

row	col
0	3
1	2
2	1
3	0
4	-1 stop

03. Given a matrix mat[N][M], print all the diagonals from right to left. (Anti-diagonals)

Note: Diagonals should start from 0' row & can also start from last col.



i	j
0	5
1	4
2	3
3	2
4	1

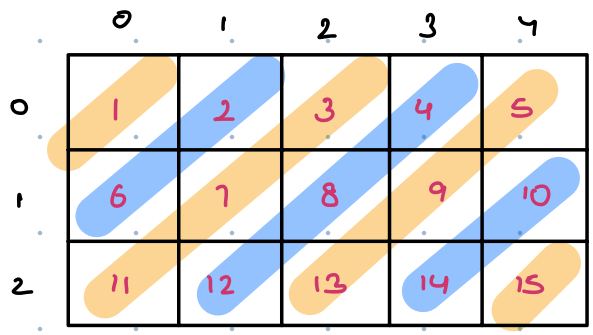
i exceeds n-1

i	j
0	1
1	0
2	-1

j < 0

Obs: i < n && j ≥ 0

A[3][5]



Output

1

2 6

3 7 11

4 8 12

5 9 13

10 14

15

Idea → Print all diagonals start from 0th row
Print all diagonals start from last col

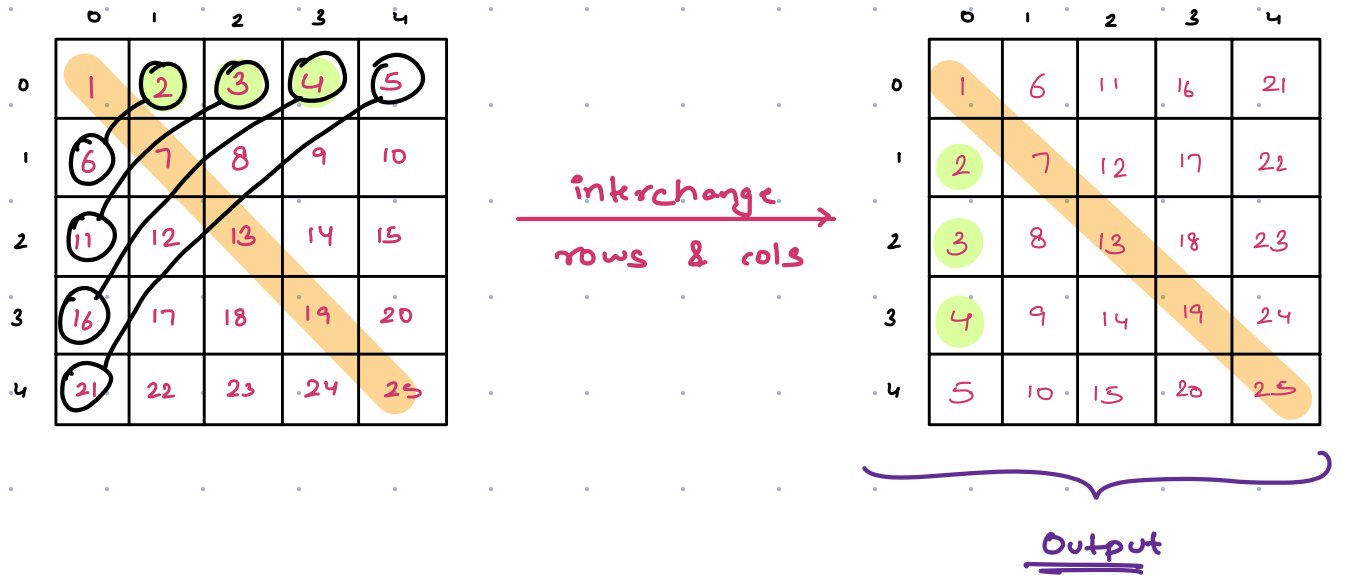
```
void printantidiagonals(int[][] A)                    N*M
{
    // print all diagonals from 0th row
    for(c=0; c<m; c++){
        i=0, j=c        // start point of a diagonal
        while(i<n && j≥0)
        {
            print(A[i][j]);
            i++;
            j--;
        }
        println();
    }
    // print diagonals from last col
    for(r=1; r<n; r++)
    {
        i=r, j=m-1
        while(i<n && j≥0)
        {
            print(A[i][j]);
            i++;
            j--;
        }
        println();
    }
}
```

TC: O(n*m)
SC: O(1)

04. Given a mat[N][N], calculate transpose of matrix

Expected SC: O(1)

interchanging rows & columns



mat[0][1] ⟷ mat[1][0]

mat[0][2] ⟷ mat[2][0]

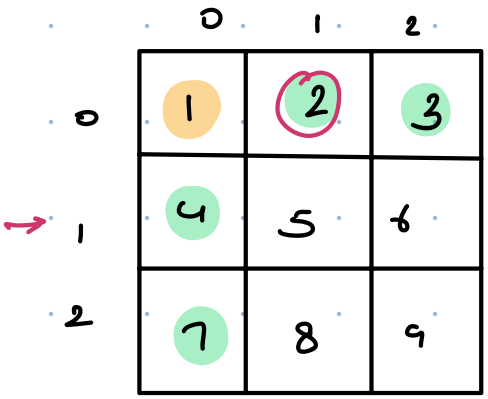
mat[0][3] ⟷ mat[3][0]

mat[i][j] ⟷ swap ⟷ mat[j][i]

i == j ⟶ no need to swap

```
void transpose ( int [][] A)
{
    for ( i=0; i<n; i++){
        for (j=i+1 ; j<n;j++)
        {
            temp= A[i][j]
            A[i][j] = A[j][i]
            A[j][i] = temp;
        }
    }
}
```

TC : $O(n^2)$
SC : $O(1)$



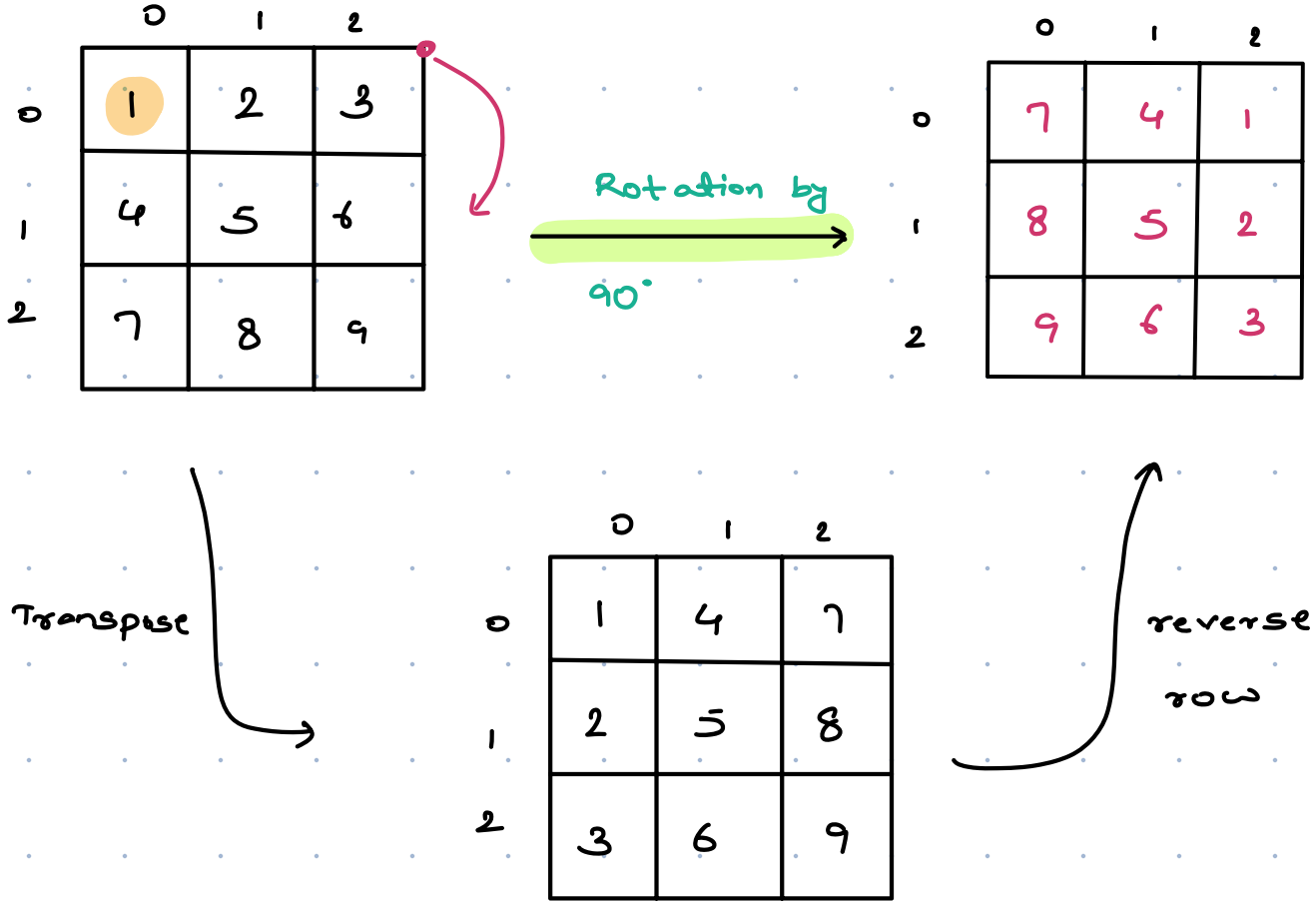
i	j
0	0
0	1
0	2
0	3 → x
1	0

A[0][1] ⟷ A[1][0]

A[0][2] ⟷ A[2][0]

A[1][0] ⟷ A[0][1]

Q Given a mat[N][N], rotate the matrix by 90° in clockwise direction.




```
void rotatematrix(int [][]A)
    transpose(A);
    for(r=0; r<n; r++){
        i=0, j=n-1
        while(i<j){
            temp = A[r][i]
            A[r][i] = A[r][j]
            A[r][j] = temp;
            i++, j--;
        }
    }
```

TC: $O(n^2)$

SC: $O(1)$

Doubts

* Rectangular matrix

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15

n*m

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

Ans

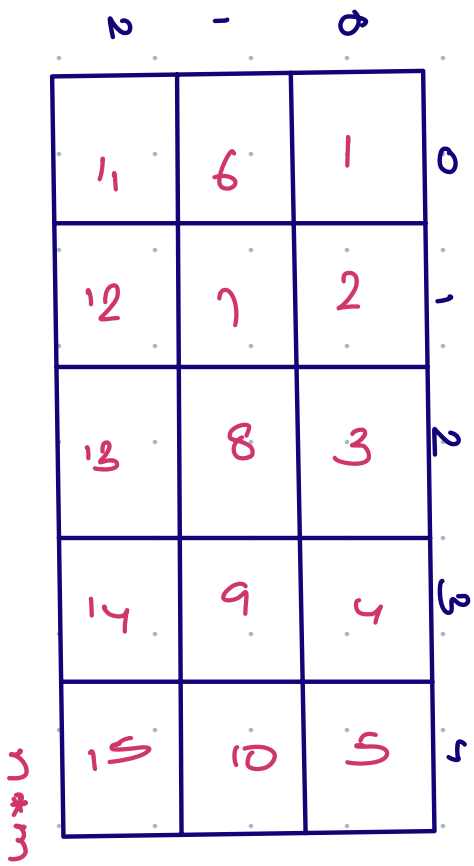
m*n

A[i][j]

A[0][1] ⟶ A[1][0]

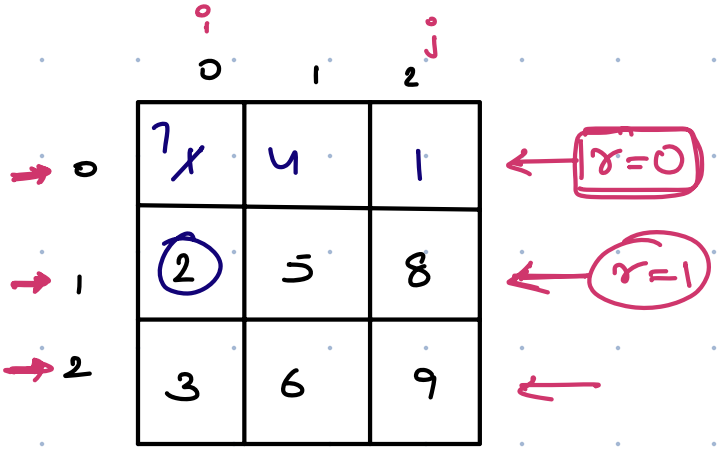
```
for (i=0; i<n; i++)
    for (j=0; j<m; j++)
        Ans[j][i] = A[i][j]
```

Transpose



reverse

Transpose




```
for (r=0; r<n; r++)
    i=0, j=n-1
    while (i<j)
        int temp = A[r][i]
        A[r][i] = A[r][j]
        A[r][j] = temp;
    i++, j--
```

r

0 < 3

i = 0 < j = 2

1 j = 1

tmp = A[0][0] = 1

A[0][0] = A[0][2]

A[0][2] = 1

r

1 < 3

i j

0 2

tmp = A[1][0] = 2

Q : elements ≤ B together

B = 5

4 6 5 8 3 2 7 7

count no. of elements that needs to be group together = 4 minimum swaps

4 6 5 8 3 2 7 7

int bad elements ⟶ look at no. of elements i need to swap

bad ele count = 2

= 2

bad = 2 − 1 = 1

bad = 1 + 1 = 2

ans = ∞ vs 2 = 2

ans = 2

ans = 1

ans = 1

bad = 2 - 1 + 1 = 2 ans = 1

```
int minswaps (int [] ar, int B)
    count = 0
    for ( int i=0 ; i<n; i++) {
        if (ar[i] ≤ B) count++;
    }
// count → No. of ele we have to group together.
    if (count == 0 || count == 1 || count == n) return 0;

// Calculate ans for first window
    bad = 0 , ans = ∞
    for ( i = 0 ; i< count ; i++ ){
        if (ar[i] > B ) bad ++;
    }
    if (bad < ans) ans = bad ;
// Apply slidind window technique
    s = 1 , e = count
    while (e<n)
        if ( ar[s-1] > B) bad--; //dropping bad
        if (ar[e] > B) bad++;    // adding bad
        if (bad < ans) ans = bad ;
        s++ , e++;
    }
return ans;
```

TC: O(n)

SC: O(1)

3