Subarrays Problem

```
for(int i=0;i<n;i++) {
    for(int j=i;j<n;j++) {
        for(int k=i;k<=j;k++) {
            System.out.print(arr[k] + "\t");
        }
        System.out.println();
    }
}
```

Example

Sample Input

```
3
10
20
30
```

Sample Output

```
10
10  20
10  20  30
20
20  30
30
```

10 20 30

→ 10

→ 10 20

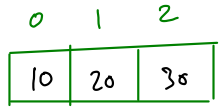
→ 10 20 30

→ 20

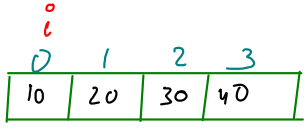
→ 20 30

→ 30

Total No of Subarrays = $\frac{n(n+1)}{2}$



⇒ Starting index
⇒ ending index } nested loops
⇒ print



Starting index = 0 (i)

ei 0 10
ei 1 10 20
ei 2 10 20 30
ei 4 10 20 30 40

Starting index = 1 (i)

ei 1 20
ei 2 20 30
ei 3 20 30 40

Starting index = 2

ei 2 30
ei 3 30 40

Starting index = 3

ei 3 40

Subsets of an Array

Example

Sample Input

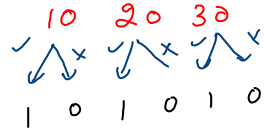
```
3
10
20
30
```

Sample Output

```
-   -   -
-   -   30
-   20  -
-   20  30
10  -   -
10  -   30
10  20  -
10  20  30
```

0	1	2
10	20	30

→ 2^n subsets



** Used Reverse Engineering
Analysed subsets → Binary → Decimal
but solved decimal → Binary → Subsets

$2^3 = 8$ binary nos & analogy with the problem

0	000	- - -
1	001	- - 30
2	010	- 20 -
3	011	- 20 30
4	100	10 - -
5	101	10 - 30
6	110	10 20 -
7	111	10 20 30

↳ loop from 0 to $2^N - 1$ to generate decimal nos

↳ convert these decimal to binary

↳ convert binary to subsets

```
for(int dec=0;dec<(int)(Math.pow(2,n));dec++) {

    int bin = convertToBinary(dec);

    int power = (int)(Math.pow(10,n-1));

    for(int i=0;i<n;i++) {

        int bit = ((bin/power)%10);
        if(bit == 1) {
            System.out.print(arr[i] + "\t");
        } else {
            System.out.print("-\t");
        }

        power = power/10;
    }

    System.out.println();
}
```

Extracting bits from binary no

$(011010 / 10^5) = 0 \% 10 = 0$

$(011010 / 10^4) = 01 \% 10 = 1$

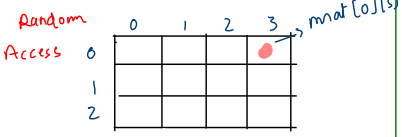
$(011010 / 10^3) = 011 \% 10 = 1$

$((binary / power) \% 10)$ → gives the MSB

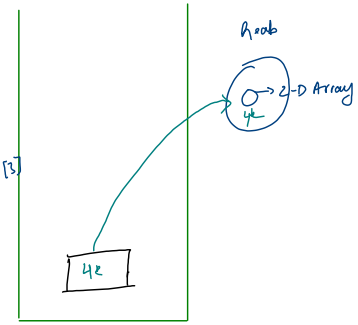
power starts from 10^{n-1}

2-D Arrays

```
//declaration & initialisation
int [][] mat = new int[3][4]
```

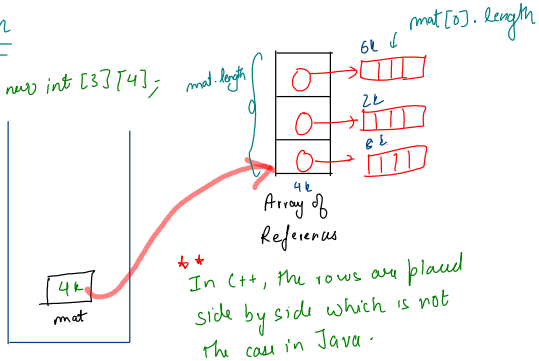


logical view i.e. only assumption



Memory Allocation

```
int [][] mat = new int[3][4];
```



** In C++, the rows are placed side by side which is not the case in Java.

2-D Arrays Demo (I/O)

```java
int n = scn.nextInt();
int m = scn.nextInt();

int[][] mat = new int[n][m];

for(int i=0;i<n;i++) {
    for(int j=0;j<m;j++) {
        mat[i][j] = scn.nextInt();
    }
}

for(int i=0;i<n;i++) {
    for(int j=0;j<m;j++) {
        System.out.print(mat[i][j] + " ");
    }
    System.out.println();
}
```

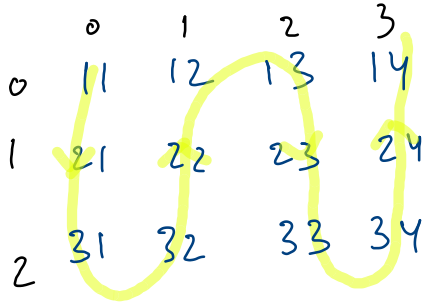
Ques 1) # State of Wakanda-1 (Wave Traversal)

Sample Input

3
4
11
12
13
14
21
22
23
24
31
32
33
34

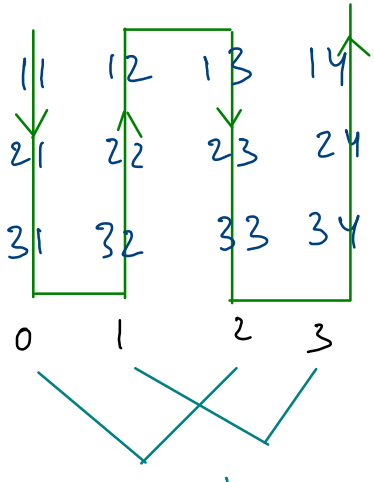
Sample Output

11
21
31
32
22
12
13
23
33
34
24
14



Column wise alternate up & down printing.

* Since we have to fix cols, outer loop will be of cols.



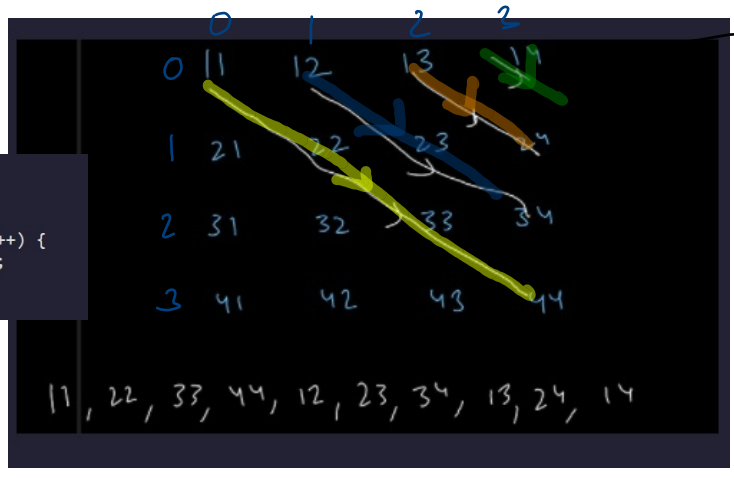
Even cols = ↓

Odd cols = ↑

```java
public static void waveTraversal(int[][] arr) {

    int rows = arr.length;
    int cols = arr[0].length;

    for(int j=0;j<cols;j++) {
        if(j%2 == 0) {
            for(int i=0;i<rows;i++) {
                System.out.println(arr[i][j]);
            }
        } else {
            for(int i=rows-1;i>=0;i--) {
                System.out.println(arr[i][j]);
            }
        }
    }
}
```

Quiz2 State of Wakanda-2 (Diagonal Order Traversal)

```
//gap strategy
for(int gap=0;gap<cols;gap++) {
    // gap signifies j - i
    for(int i=0,j = gap;j<cols;i++,j++) {
        System.out.println(mat[i][j]);
    }
}
```



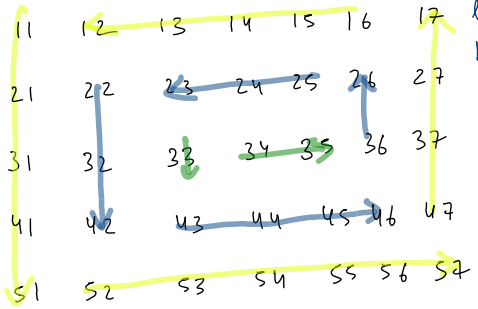
*** VV Imp Observation

Note that all the diagonals start from any element of first row. So gap will depend on no of elements in the first row i.e. columns.

11, 22, 33, 44, 12, 23, 34, 13, 24, 14

j - i = 0 j - i = 1 j - i = 2 j - i = 3

Q113 Spiral Display

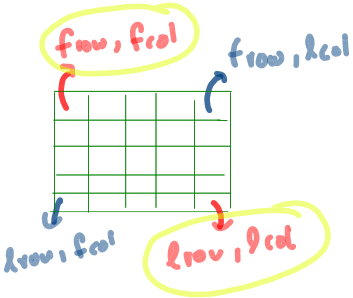


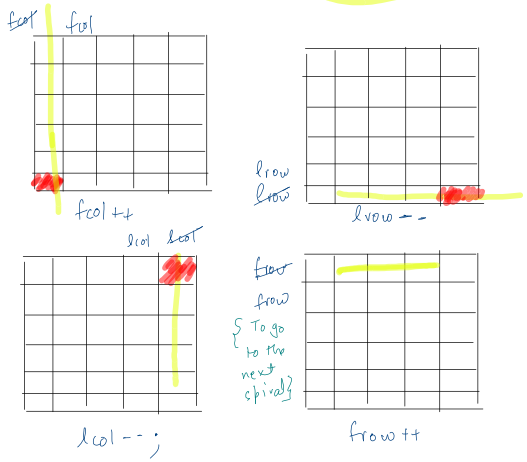
left wall → T to B
bottom wall → L to R
Right wall → B to T
Top wall → R to L

Caution about Corner elements

```
while(true) {

    //left wall
    for(int i=frow;i<=lrow;i++) {
        System.out.println(arr[i][fcol]);
        count++;
        if(count == rows*cols) {
            return;
        }
    }
    fcol++;

    //bottom wall
    for(int j=fcol;j<=lcol;j++) {
        System.out.println(arr[lrow][j]);
        count++;
        if(count == rows*cols) {
            return;
        }
    }
    lrow--;

    //right wall
    for(int i=lrow;i>=frow;i--) {
        System.out.println(arr[i][lcol]);
        count++;
        if(count == rows*cols) {
            return;
        }
    }
    lcol--;

    //top wall
    for(int j=lcol;j>=fcol;j--) {
        System.out.println(arr[frow][j]);
        count++;
        if(count == rows*cols) {
            return;
        }
    }
    frow++;
}
```

frow = 0
fcol = 0
lrow = mat.length - 1
lcol = mat[0].length - 1



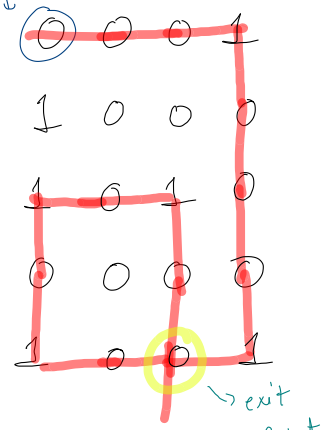


Avoid Repetition of corner elements

↳ We are counting each element after it gets printed & we know that total no of ele = row * col. So, if any where in blw printing any spiral count = row * col then stop the procedure.

Ques 4 Exit Point of a Matrix

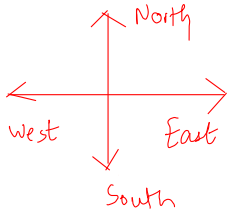
↓ starting pt & init dir is east

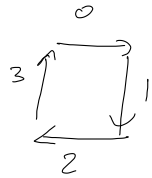


* → There will always be an Exit Point as we are starting from (0,0)

→ If we start from somewhere in the middle, it is possible that there will not be any exit point.

§ Constructive algo: when we are just given a sequence of steps & we follow them as it is in our solution & reach the final ans, it is constructive algo §





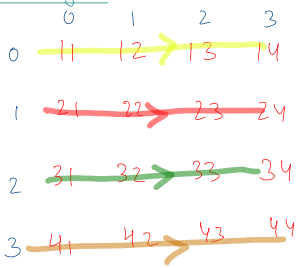
dir = (dir + 1) % 4

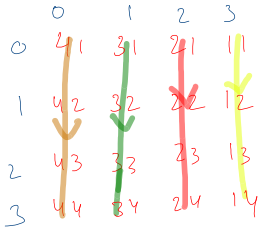
```
int ansCol = 0;
while((currCol >= 0) && (currRow >= 0) && (currCol < cols) && (currRow < rows)) {

   if(arr[currRow][currCol] == 1) {
      direction = (direction + 1) % 4;
   }

   ansRow = currRow;
   ansCol = currCol;

   if(direction == 0) {
     currCol++;
   } else if(direction == 1){
     currRow++;
   } else if(direction == 2) {
      currCol--;
   } else {
      currRow--;
   }
}

System.out.println(ansRow);

System.out.println(ansCol);
```

r-1, c-1	r-1, c	r-1, c+1
r, c-1	r, c	r, c+1
r+1, c-1	r+1, c	r+1, c+1

Directions

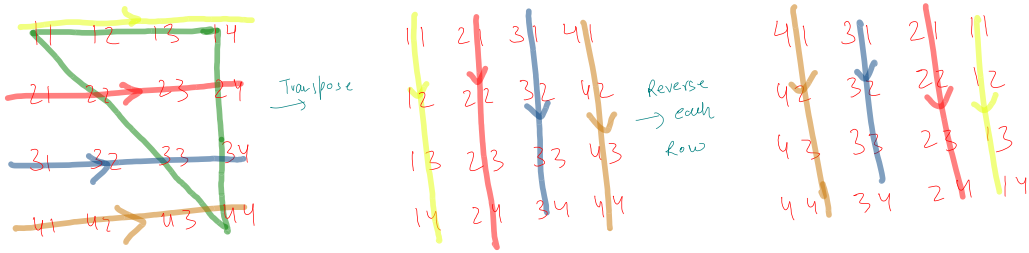
Ques) Rotate by 90°





↳ Transpose i.e a[i][j] ⇔ a[j][i]

(only upper triangle or lower triangle)




```java
public static void swap(int[][] arr, int i, int j) {

    int temp = arr[i][j];
    arr[i][j] = arr[j][i];
    arr[j][i] = temp;
}

public static void transpose(int[][] arr) {

    for(int i=0;i<arr.length;i++) {
        for(int j=i+1;j<arr[0].length;j++) {
            swap(arr,i,j);
        }
    }
}

public static void rotate(int[][] arr) {
    transpose(arr);

    for(int i=0;i<arr.length;i++) {

        int l = 0;
        int h = arr[0].length-1;
        while(l<h) {
            int temp = arr[i][l];
            arr[i][l] = arr[i][h];
            arr[i][h] = temp;
            l++;
            h--;
        }
    }
```

Ques) Matrix Multiplication

$a_{n_1 * m_1}$ $b_{n_2 * m_2}$

Multiplication is possible only if $(m_1 = n_2)$

resultant matrix will have dimensions $(n_1 * m_2)$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}_{3\times4} \quad \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}_{4\times2} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}_{3\times2}$$

Obs 1: Row of a is equal to Row of c

Obs 2: Col of b is equal to col of c

Obs 3: Col of a & Row of b equal to k.

$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31} + a_{14} * b_{41}$

$c_{12} = a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32} + a_{14} * b_{42}$

$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31} + a_{24} * b_{41}$

$c_{22} = a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32} + a_{24} * b_{42}$

c_{ij} → 1st matrix ki i^{th} row ko 2nd matrix ki j^{th} col se multiply karna do

↳ 3 nested Loops

Outer 2 Loops for traversing row-wise in resultant matrix to fill it & inner most loop to calculate the value to be kept at res[i][j].

$$c_{ij} = \sum_{k=1}^{m_1 \text{ or } n_2} a_{ik} * b_{kj}$$

```
if(m1 != n2) {
    System.out.println("Invalid input");
    return;
}

int[][] res = new int[n1][m2];

for(int i=0;i<res.length;i++) {
    for(int j=0;j<res[0].length;j++) {

        for(int k=0;k<m1;k++) { //m1 = n2 so can take n2 also

            res[i][j] += (a1[i][k] * a2[k][j]);
        }
    }
}

display(res);
```

Ques) Saddle Point / Price

ALL elements are unique

(Rows = Cols is not necessary)

	0	1	2	3
0	34	42	30	59
1	66	85	44	97
2	42	16	25	36
3	15	21	05	50

Saddle Point : min in its row & max in its col

① is it possible to have 0 saddle points? Yes

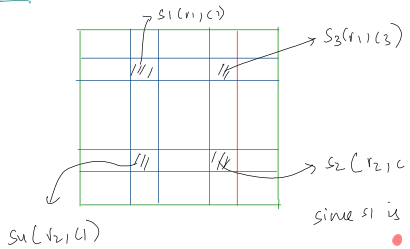
② is it possible to have multiple saddle points? No

Example for 0 saddle points

	0	1	2	3
0	34	42	30	59
1	20	85	44	97
2	42	16	25	36
3	15	21	05	50

Proof that only 1 Saddle Point Exists

Assumption : There are 2 saddle points S_1 and S_2



Since S_1 is a S.P

- $S_1 < S_3$
- $S_1 > S_4$

Since S_2 is a SP

- $S_2 < S_4$
- $S_2 > S_3$

$S_1 < S_3 < S_2$

$S_1 > S_4 > S_2$

$S_1 < S_2$

$S_1 > S_2$

Cannot happen simultaneously.

Hence our assumption is wrong.

There will be atmost 1 Saddle point

	0	1	2	3
0	34	42	30	59
1	66	85	44	97
2	42	16	25	36
3	15	21	05	50

↳ find min val in row

↳ Then find the max value in the col in which min val of row was found

↳ if the max val of col is at the same position as that of min row val, we have found the saddle point.

```
for(int i=0;i<arr.length;i++) {
    int minCol = 0;

    for(int j=0;j<arr[0].length;j++) {
        if(arr[i][j] < arr[i][minCol]) {
            minCol = j;
        }
    }

    int maxRow = 0;
    for(int k=0;k<arr.length;k++) {
        if(arr[k][minCol] > arr[maxRow][minCol]) {
            maxRow = k;
        }
    }

    if(maxRow == i) {
        System.out.println(arr[maxRow][minCol]);
        return;
    }
}
System.out.println("Invalid input");
```

Maths for DSA

Most imp ***
1. Permutations & Combinations

Imp **
2. Sequence & Series → AP, GP

3. Number Theory → Prime Nos, GCD and LCM, factorial

4. Matrix Concepts

Imp **
5. Logarithm
6. Sets