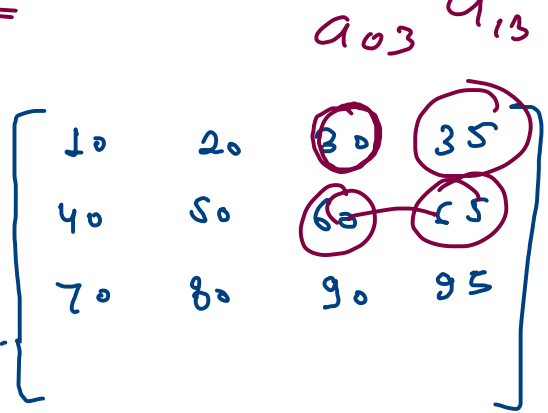
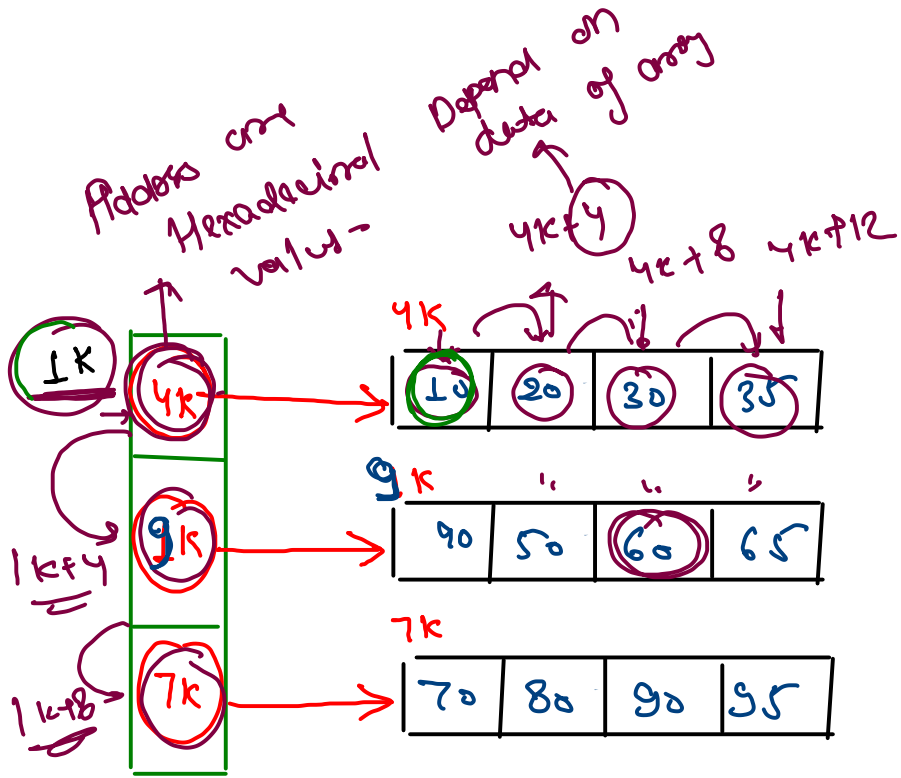
Two D Array →

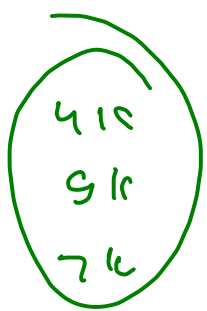


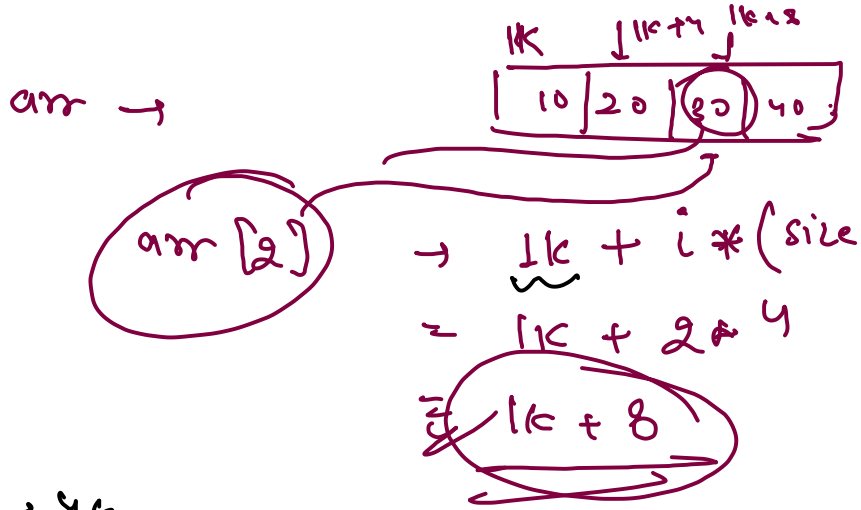
int[][] arr = new int[n][m]

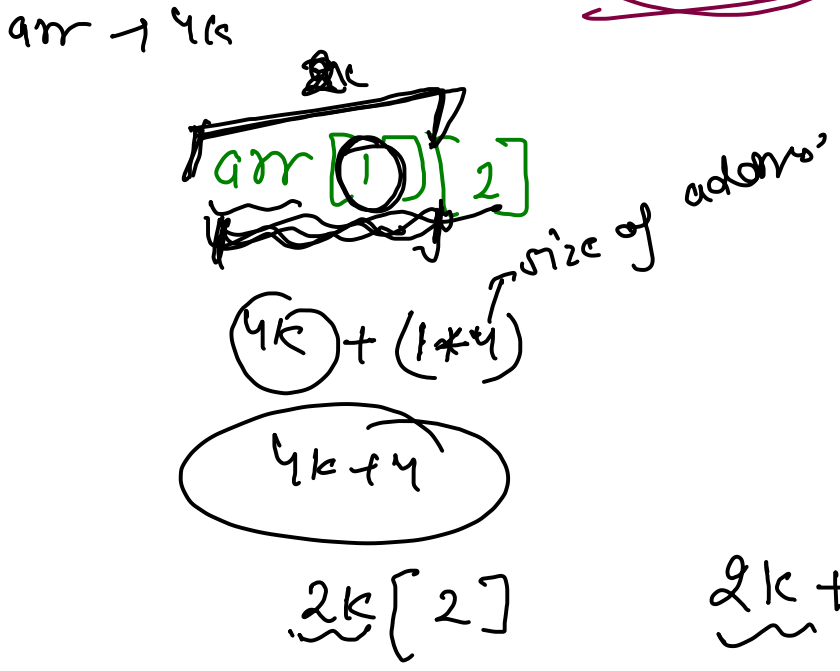
Reference Variable

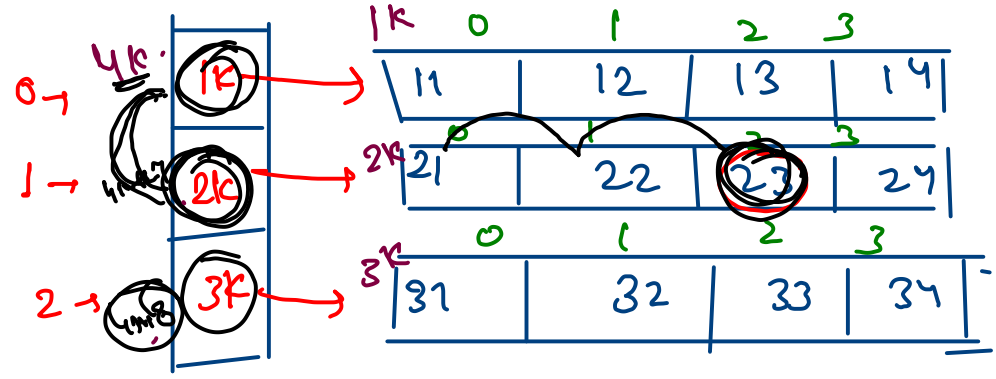
arr = 1K






arr →
1K
1K+4
1K+8
10
20
30
40
arr[2]
→ 1K + i * (size of data)
= 1K + 2 * 4
= 1K + 8


arr → 4K
arr [1] [2]
size of
address
4K + (1*4)
4K+4
2K[2]
2K + (2 * 4)
= 2K + 8
= 23


1K
4K
0 →
1 →
2 →
1K
2K
3K
0
1
2
3
11
12
13
14
21
22
23
24
31
32
33
34

(i) Create & Initialise

(ii) Input

(iii) Output

(iv) Rows and colums.

int [][] arr = new int[r][c]

int rows = arr.length; (3)

int col = arr[0].length; → (4)

arr[0].length

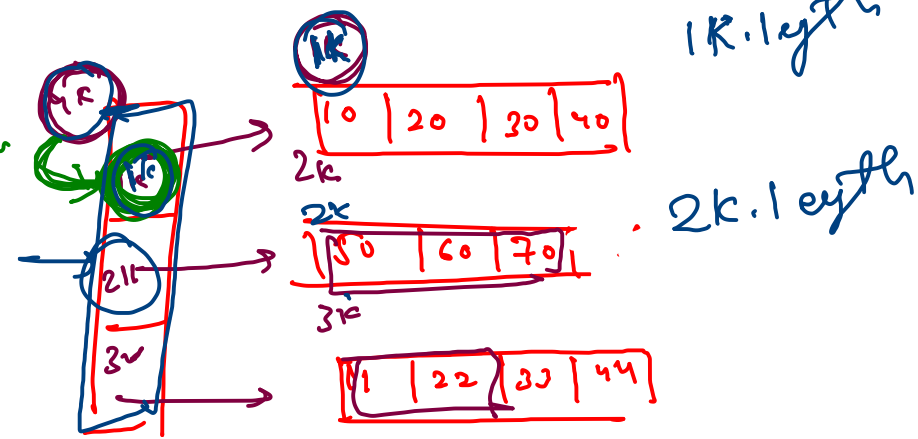
arr.length

4K.length

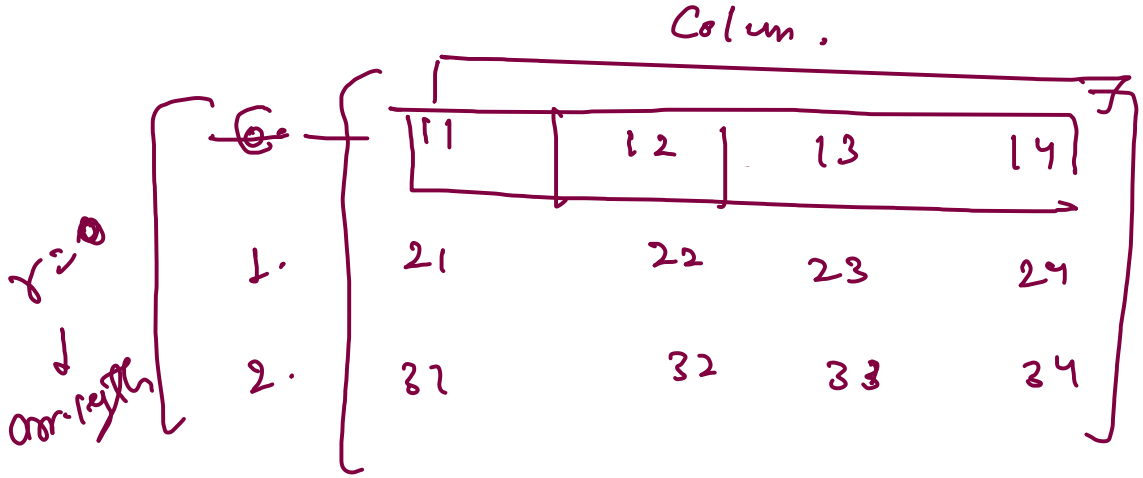
1K.length

arr[0] = 4K + 0K4

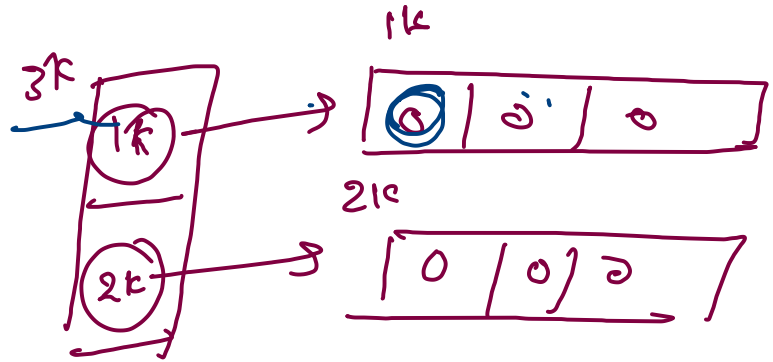
= 4K



= arr[r].length



int [ ][ ] arr = new int [2][3]



arr

$m_1 =$ $\begin{bmatrix} 1 & 9 & 4 & 5 \\ 3 & 2 & 7 & 9 \\ 8 & 5 & 6 & 4 \end{bmatrix}_{3\times4}$ $r_1 \times c_1$

$m_2 =$ $\begin{bmatrix} 1 & 9 & 3 \\ 3 & 2 & 4 \\ 8 & 5 & 9 \\ 6 & 7 & 7 \end{bmatrix}_{4\times3}$

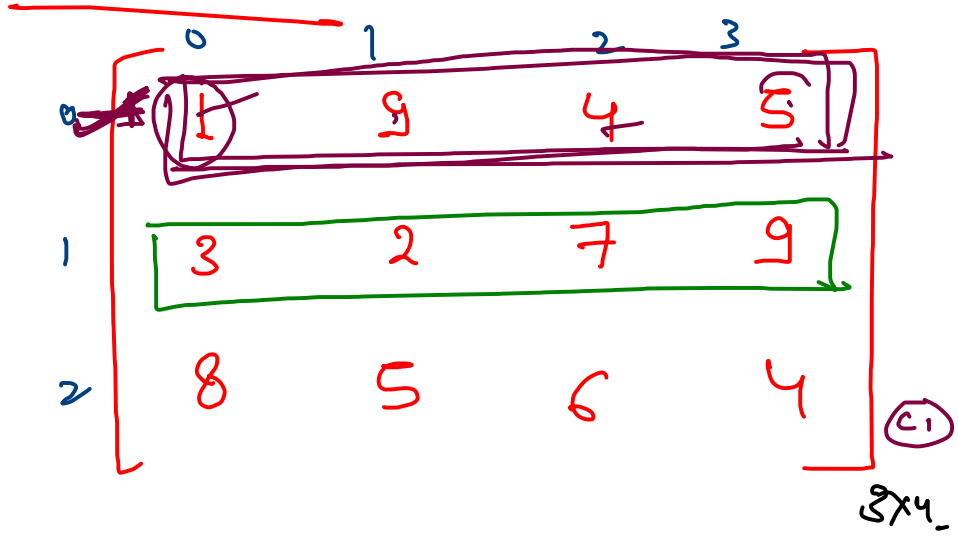
$m_1 \times m_2$

Matrix multiplication is only possible if $(c_1 == r_2)$ $r_2 \times c_2$

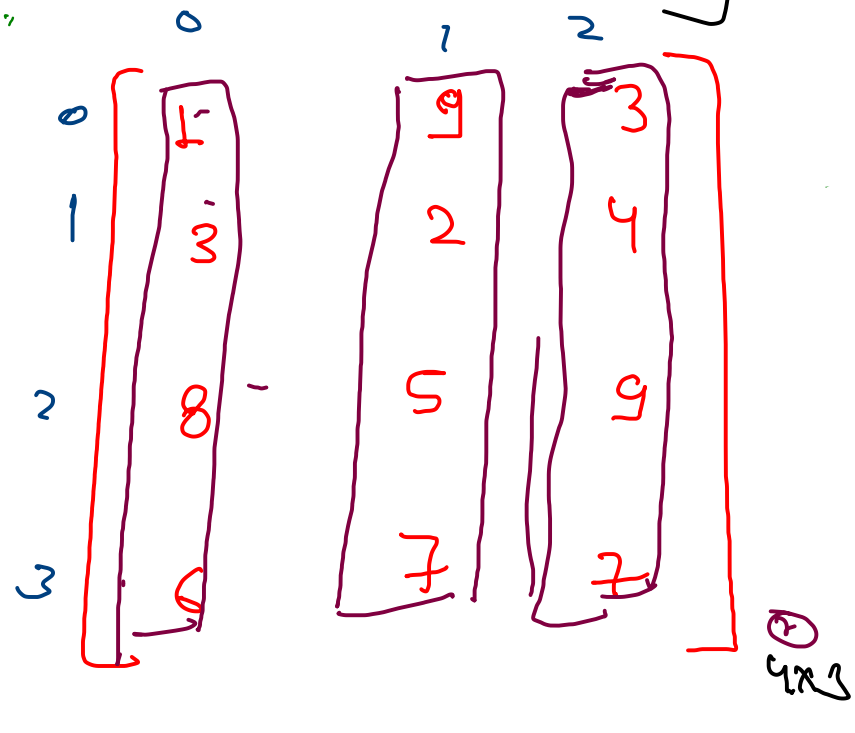
otherwise invalid input.

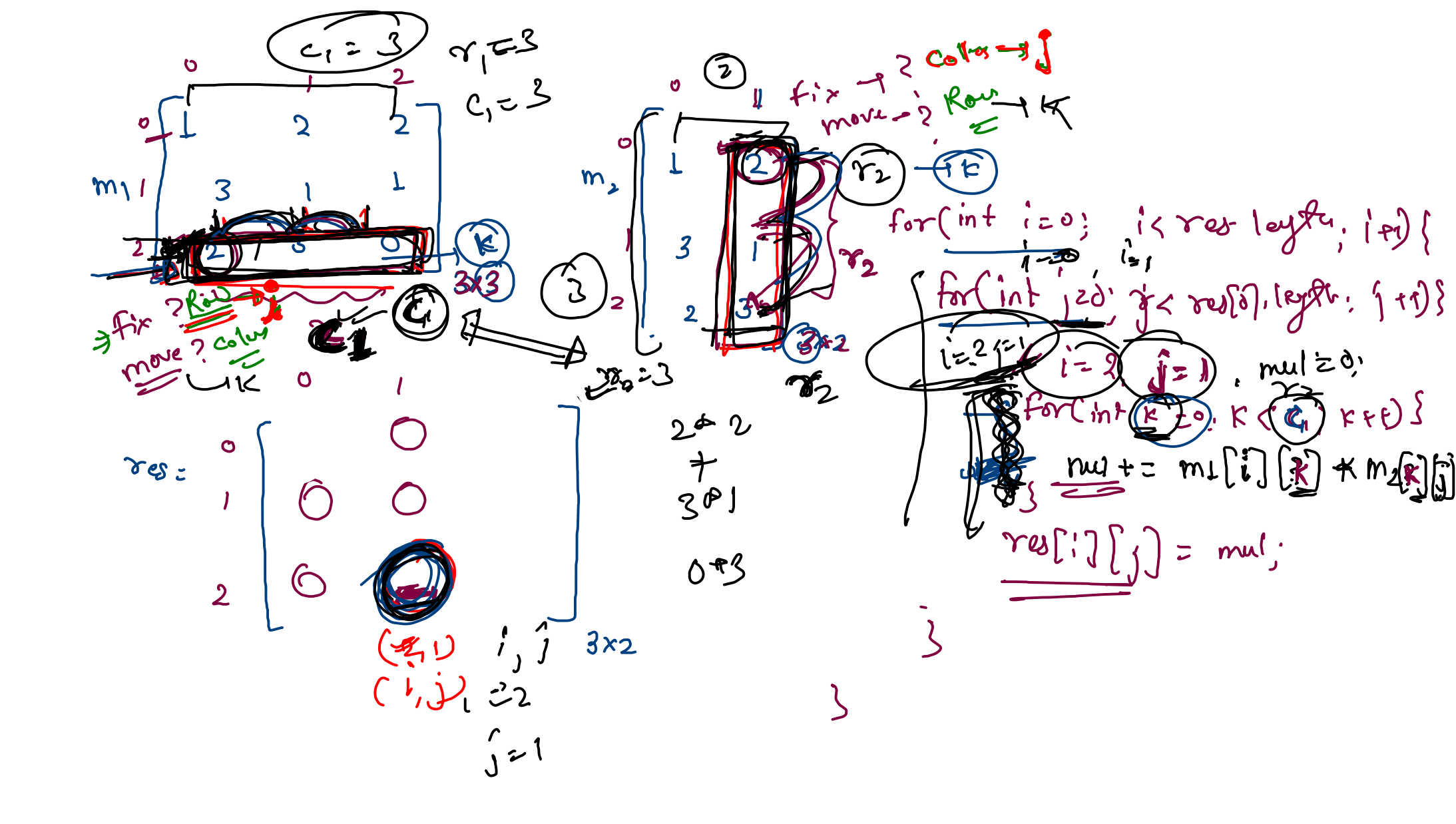
resultant matrix size : $[r_1 \times c_2]$

3×4 $\quad$ 4×3



$res =$ $\begin{bmatrix} 90 & 82 & 110 \\ 119 & - & - \\ - & - & - \end{bmatrix}$



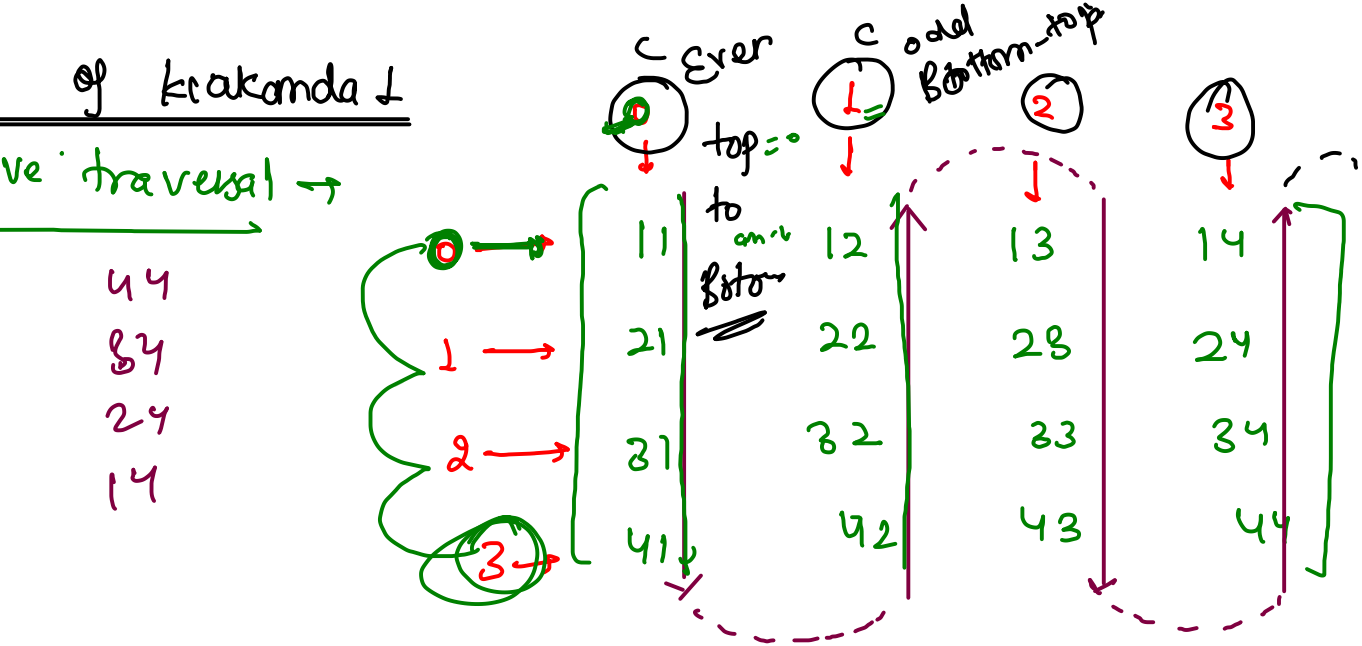

c1 = 3
r1 = 3
c1 = 3
m1
fix ? Row
move ? colm
k
3x3
r2 = 3
m2
fix → 2 Colm → j
move → 2 Row → k
r2
for(int i=0; i< res.length; i++){
for(int j=0; j< res[0].length; j++){
i=2 i=1
i = 2 j = 1
mul = 0;
for(int k=0; k< c1 k++){
mul += m1[i][k] * m2[k][j]
res[i][j] = mul;
res =
3x2
(i, j)
i, j
(i, j) = 2
j = 1
2 * 2
+
3 * 1
0 * 3

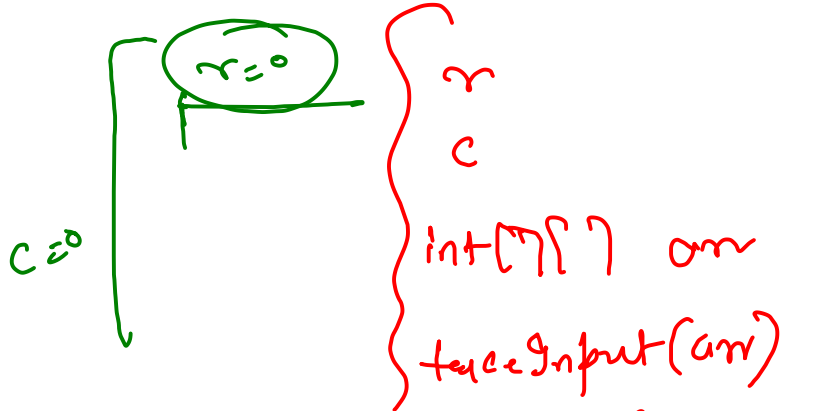
State of Kraakanda 1

Klave traversal →

11 44
21 34
31 24
41 14

42
32
22
12
13
23
33
43





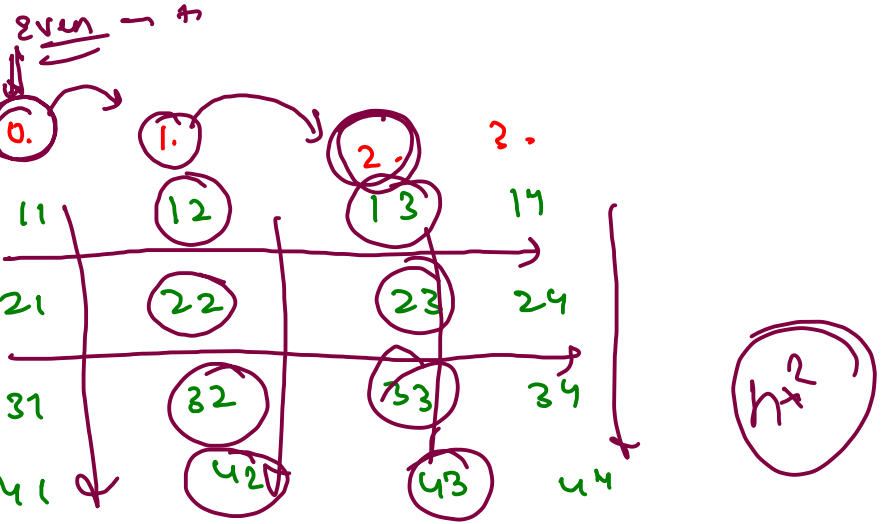
0 → row = 0 to total Row
col

1 → r0 = total col to row0
col

2 → row = 0 to tol
c

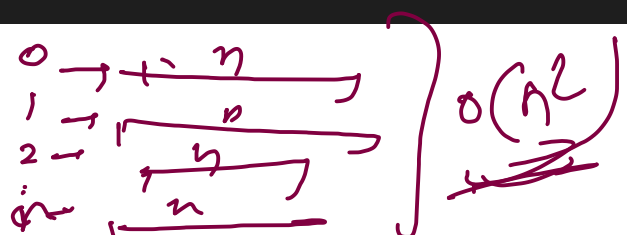
3 → rot = tol → r0
col

Time Complexity → O(n²)

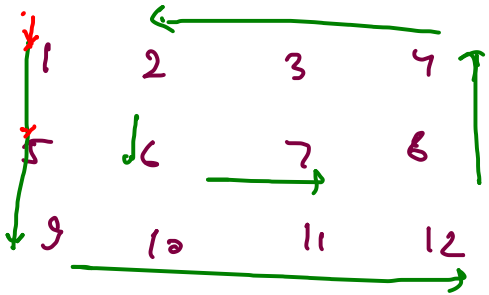



```java
public static void waveDislpay(int[][] arr) {
    for(int c = 0; c < arr[0].length; c++) {
        if(c % 2 == 0) {
            // even column -> display row top -> bottom
            for(int r = 0; r < arr.length; r++) {
                System.out.println(arr[r][c]);
            }
        } else {
            // odd column -> display row bottom -> top
            for(int r = arr.length - 1; r >= 0; r--) {
                System.out.println(arr[r][c]);
            }
        }
    }
}
```

11
13
21
23
31
33
41
43
42
32
22
12

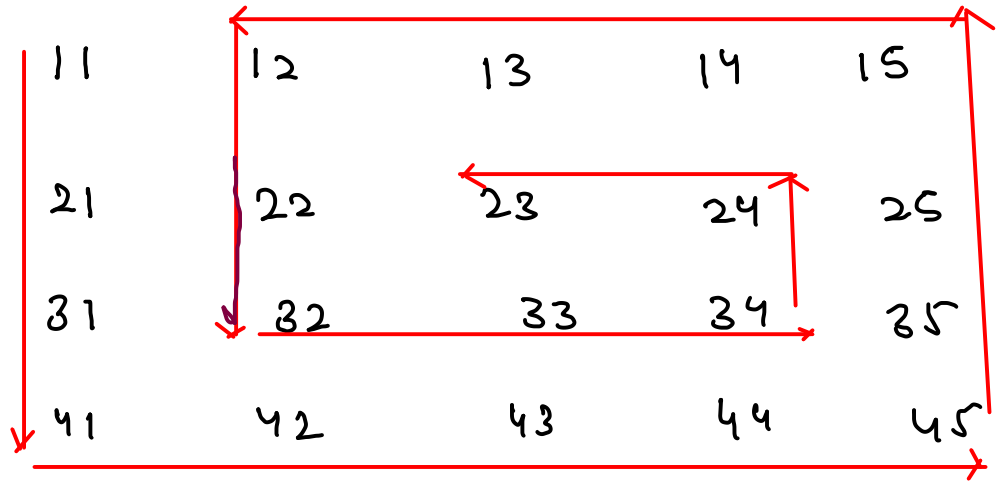


Spiral Display -



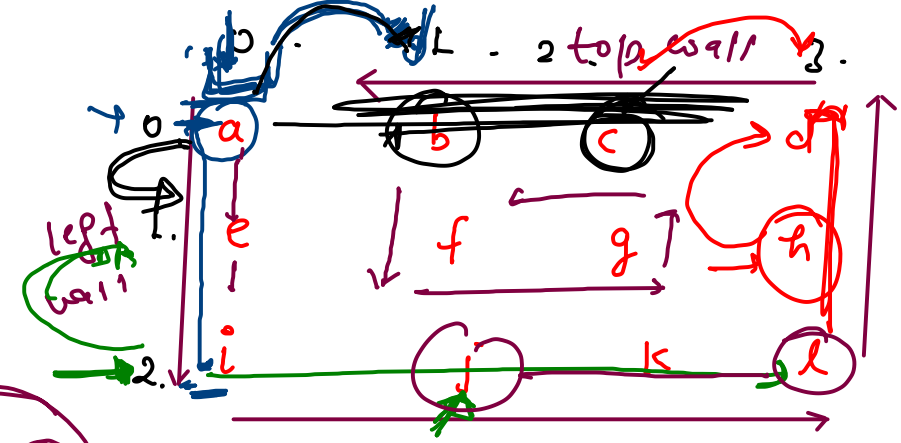
1, 2, 3, 4, 9, 10, 11, 12, 8, 7, 6, 5

3, 2, 6, 7



O/p → 11, 21, 31, 41, 42, 43, 44, 45, 35, 25, 15, 14, 13, 12,

22, 32, 33, 34, 24, 23



ele = 0

rmin = 0

rmax = 2

cmin = 0 1

cmax = 3

a e i

total Element → r x c

tele = n*m

while(tele > 0)

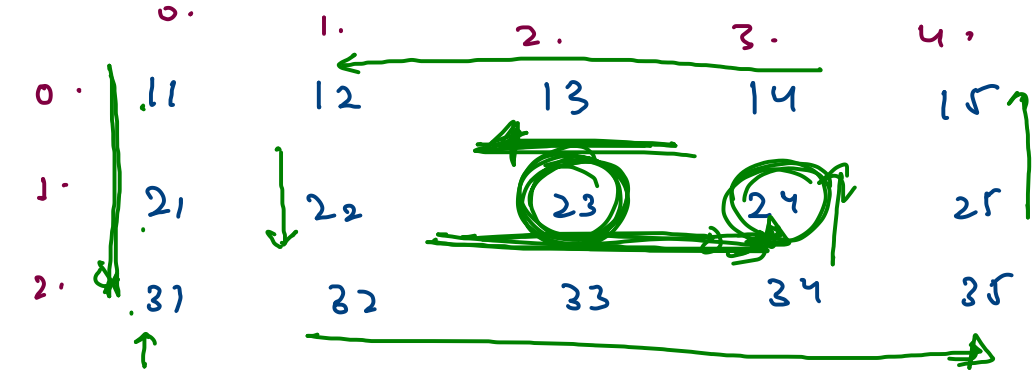
// left wall →
fix → min column.
vary → Row (minRow → maxRow)
print → tele--
cmin++;

// Bottom wall →
fix → max Row
vary → column (minCol to maxCol)
print → tele--
rmax--;

// Right wall →
fix → max. Colum.
vary → Row (rmax to rmin)
print → tele--
cmax--;

// top wall →
fix → Row (minRow)
vary → colum (cmax to cmin)
print → tele--
rmin++

```
11
21
31
32
33
34
35
25
15
14
13
12
22
23
24
23
```



tele: ~~15~~ ~~12~~ ~~8~~ ~~6~~ ~~3~~ ~~2~~ ~~0~~ -1

rmin: ~~0~~ 1 2

rmax: ~~2~~ ~~1~~ 0

cmin: ~~0~~ ~~1~~ 2

cmax: ~~4~~ ~~3~~ 2