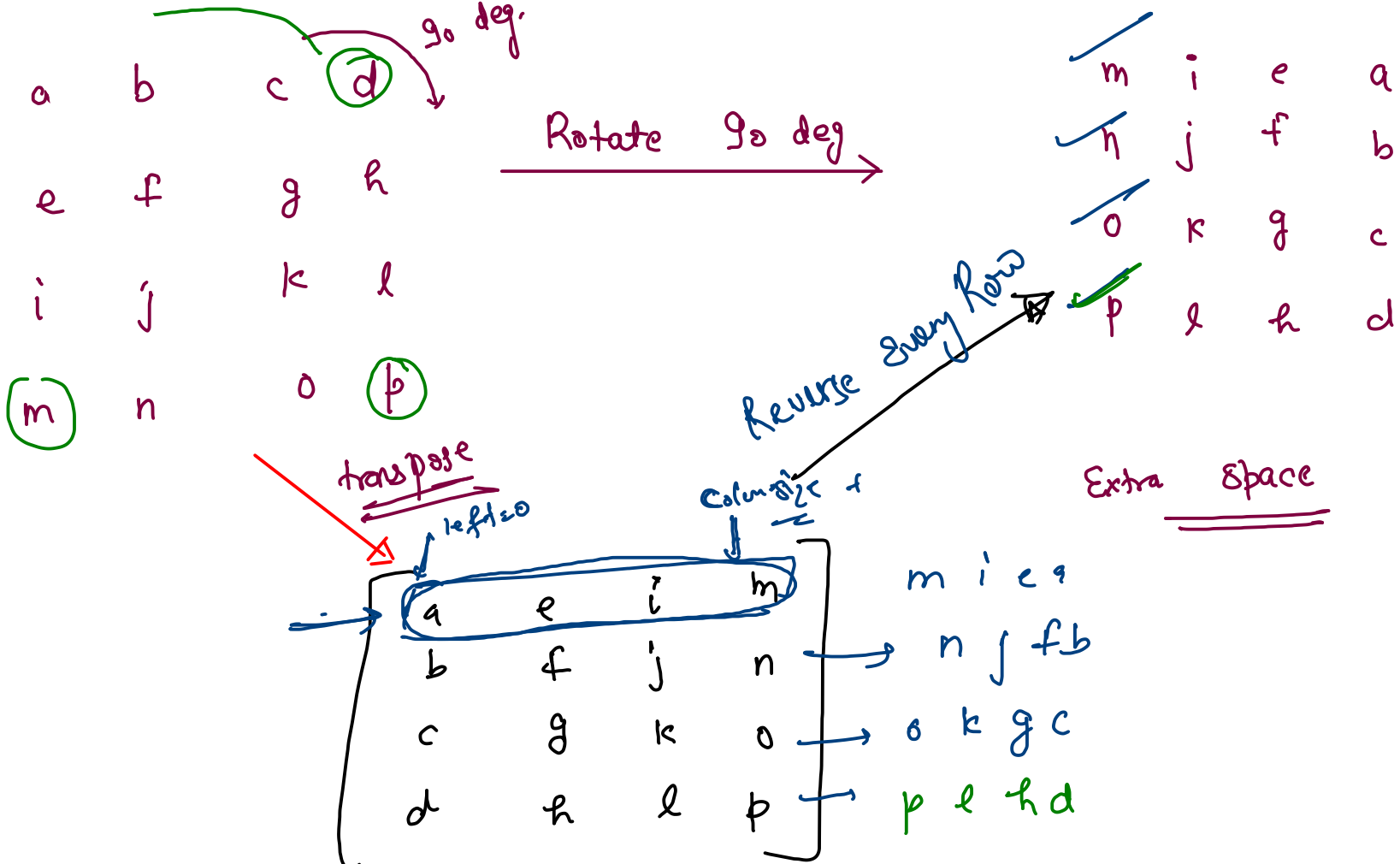
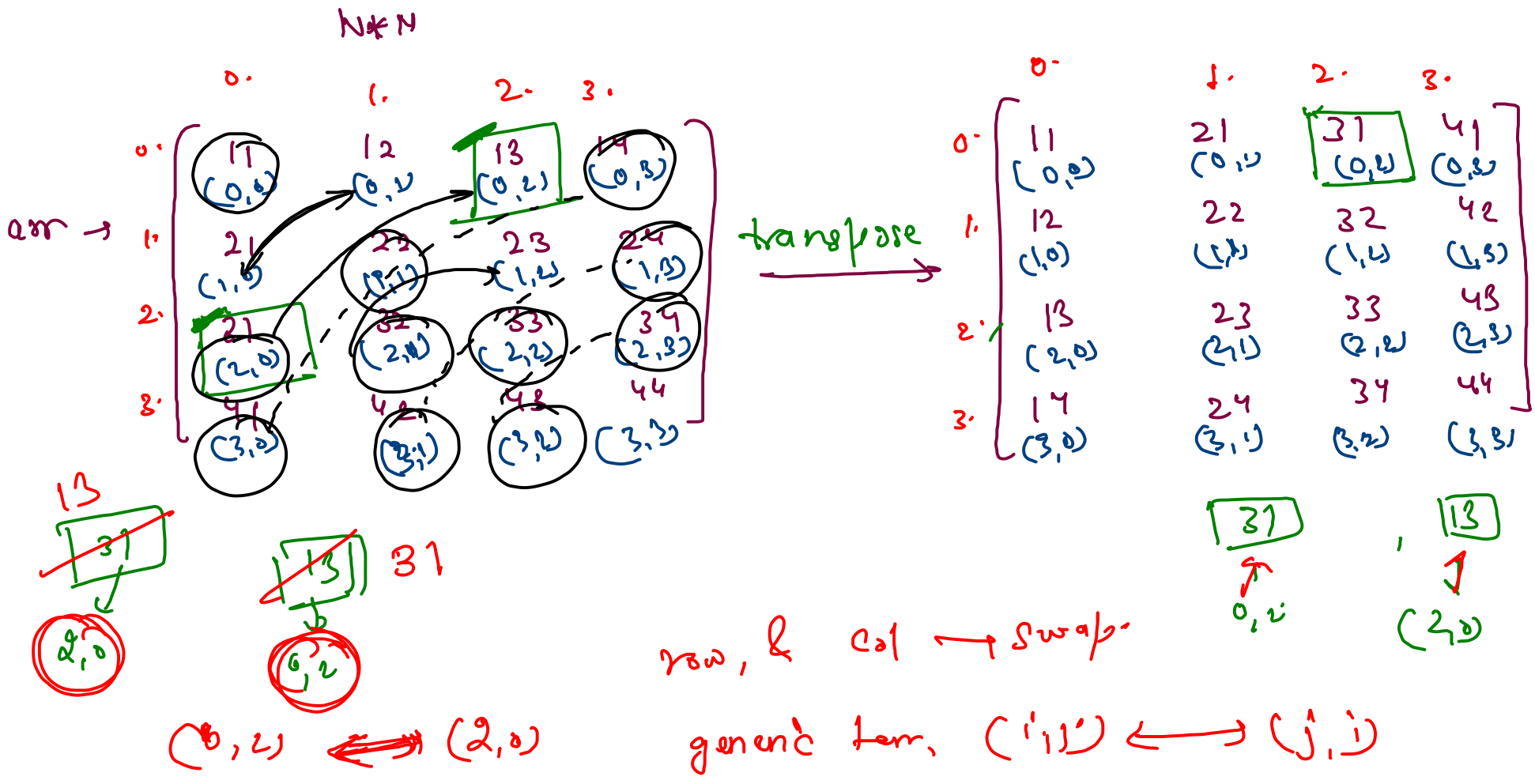
Rotate by 90 deg - nxn



transpose of a matrix →



11 12 13

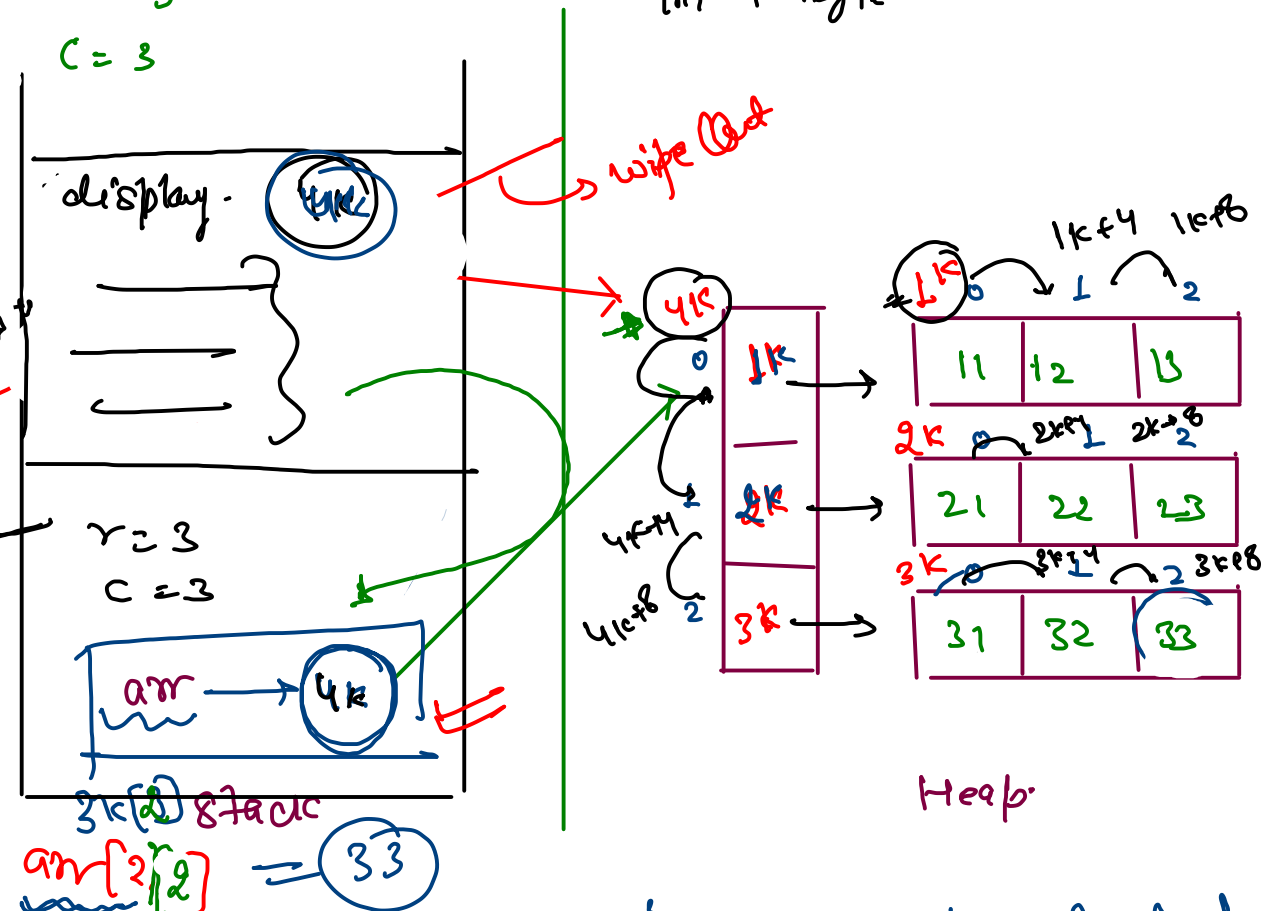
21 22 23

31 32 33

```java
public class demo {

    public static Scanner scn = new Scanner(System.in);

    public static void takeInput(int[][] arr) {
        for(int i = 0; i < arr.length; i++) {
            for(int j = 0; j < arr[0].length; j++) {
                arr[i][j] = scn.nextInt();
            }
        }
    }

    public static void display(int[][] arr) {
        for(int i = 0; i < arr.length; i++) {
            for(int j = 0; j < arr[0].length; j++) {
                System.out.print(arr[i][j] + " ");
            }
            System.out.println();
        }
    }
    Run | Debug
    public static void main(String[] args) {
        int r = scn.nextInt();
        int c = scn.nextInt();

        int[][] arr = new int[r][c];
        takeInput(arr);
        display(arr);
    }
```

r = 3

c = 3



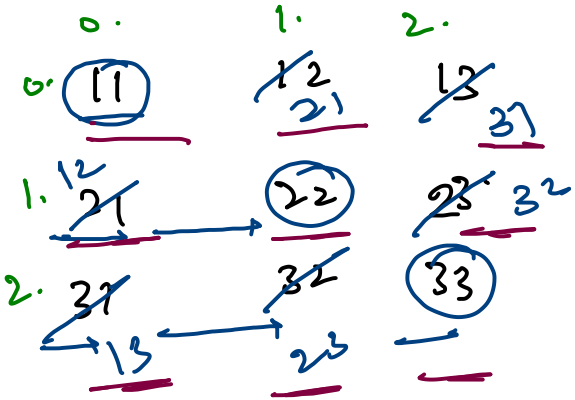
size of int → 4byte

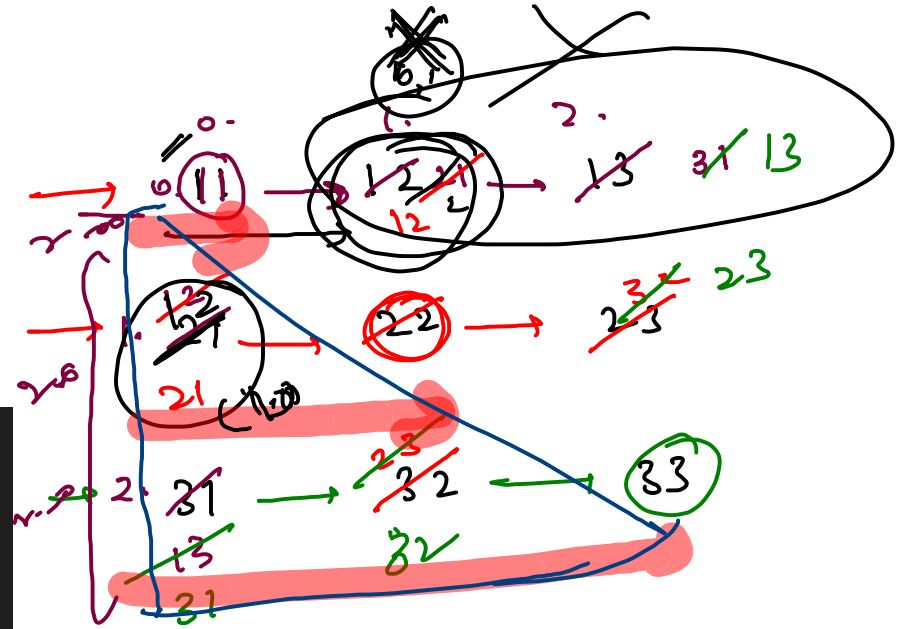
arr[2][2] = 33

arr[0] = 1k

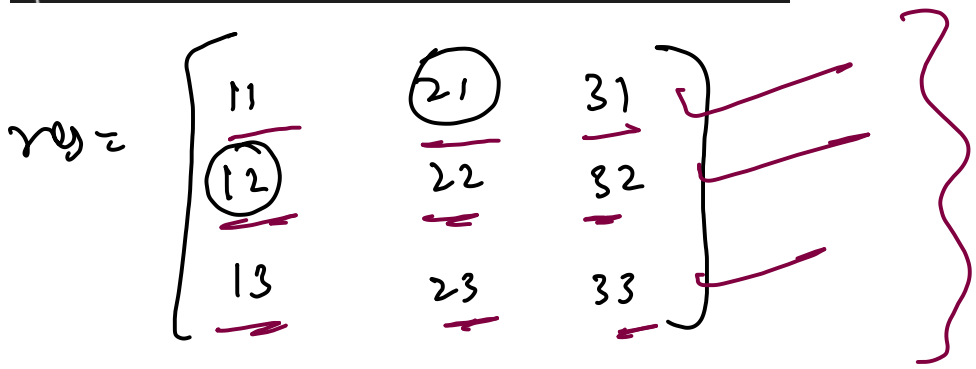
arr[1] = 2k arr[2] = 3k

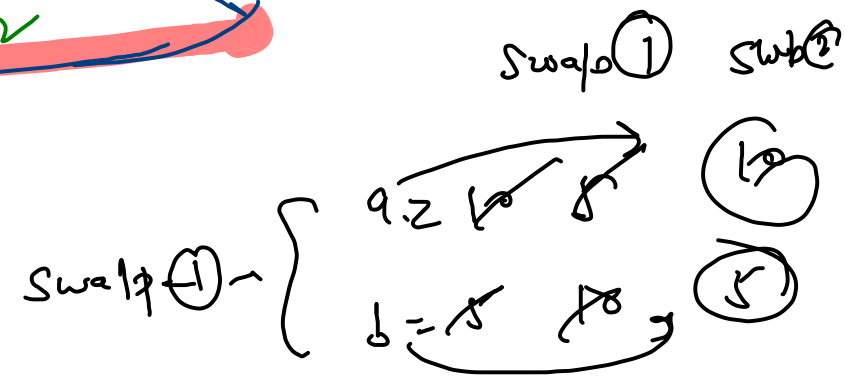
Address → Hexadecimal → 4byte.

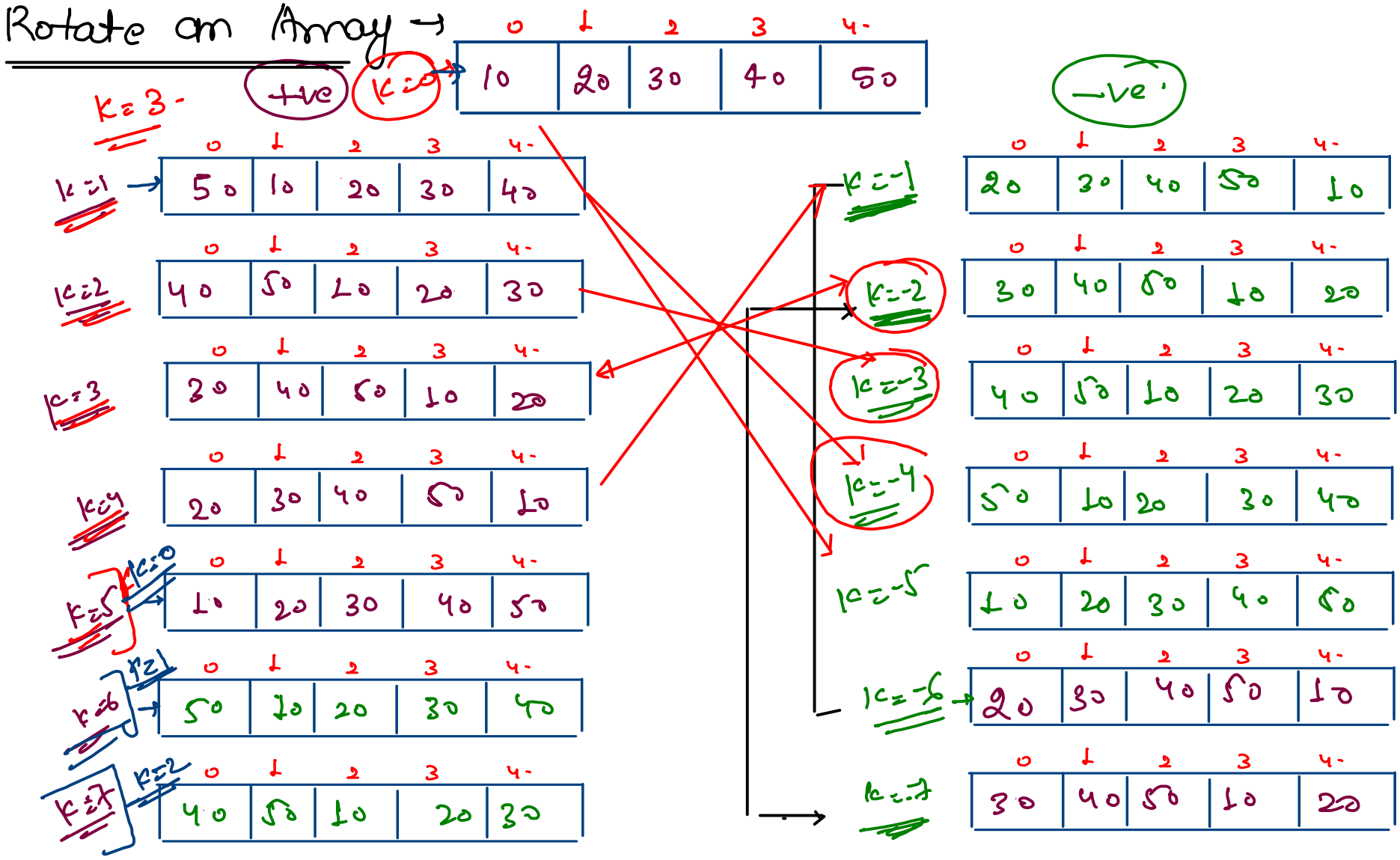




```java
public static void transposeMatrix(int[][] arr) {
    for(int r = 0; r < arr.length; r++) {
        for(int c = 0; c < arr.length; c++) {
            int temp  = arr[r][c];
            arr[r][c] = arr[c][r];
            arr[c][r] = temp;
        }
    }
}
```






Rotate an Array →
+ve
k=0
10 20 30 40 50
k=3
k=1
50 10 20 30 40
k=2
40 50 10 20 30
k=3
30 40 50 10 20
k=4
20 30 40 50 10
k=5
k=0
10 20 30 40 50
k=6
k=1
50 10 20 30 40
k=7
k=2
40 50 10 20 30
-ve
k=-1
20 30 40 50 10
k=-2
30 40 50 10 20
k=-3
40 50 10 20 30
k=-4
50 10 20 30 40
k=-5
10 20 30 40 50
k=-6
20 30 40 50 10
k=-7
30 40 50 10 20

k - Management

```
k = k % arr.length;
if(k < 0) {
    k += arr.length;
}
```

k - valid → (0 to length-1)

k = -7

k = -7 % 5 → +3

k = -2

k = k + arr.length = +3

if k pos

k = k % arr.length → across the length → 0 to arr.length-1

if k neg

k = -7

-4%3 = -1

4%3 = 1
5%3 = 2

-7%5 = 3

k = k % arr.length

-7 % 5 = -2 → -length+1

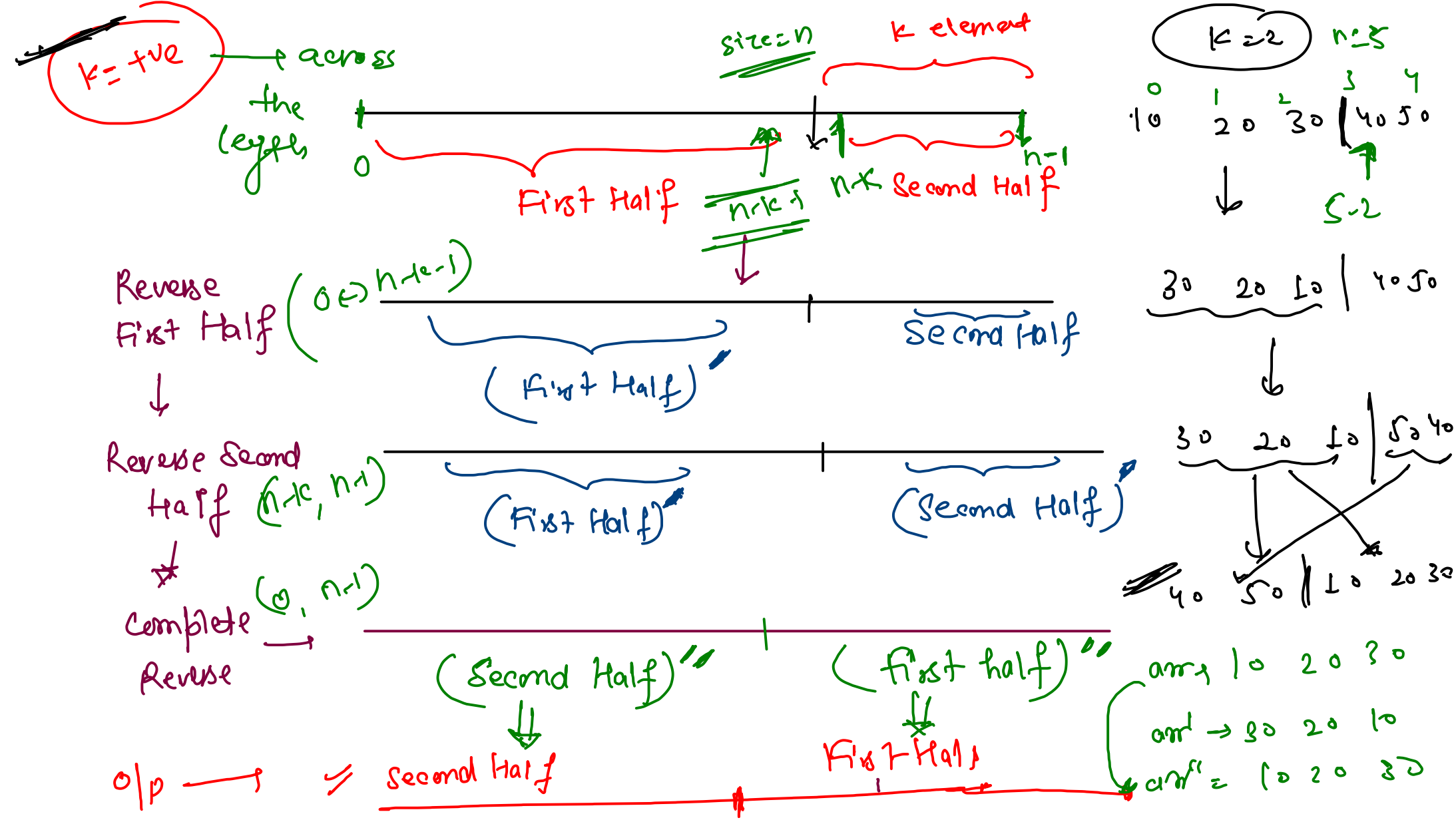
if(k < 0) {

k = k + arr.length;

k = -2 + 5

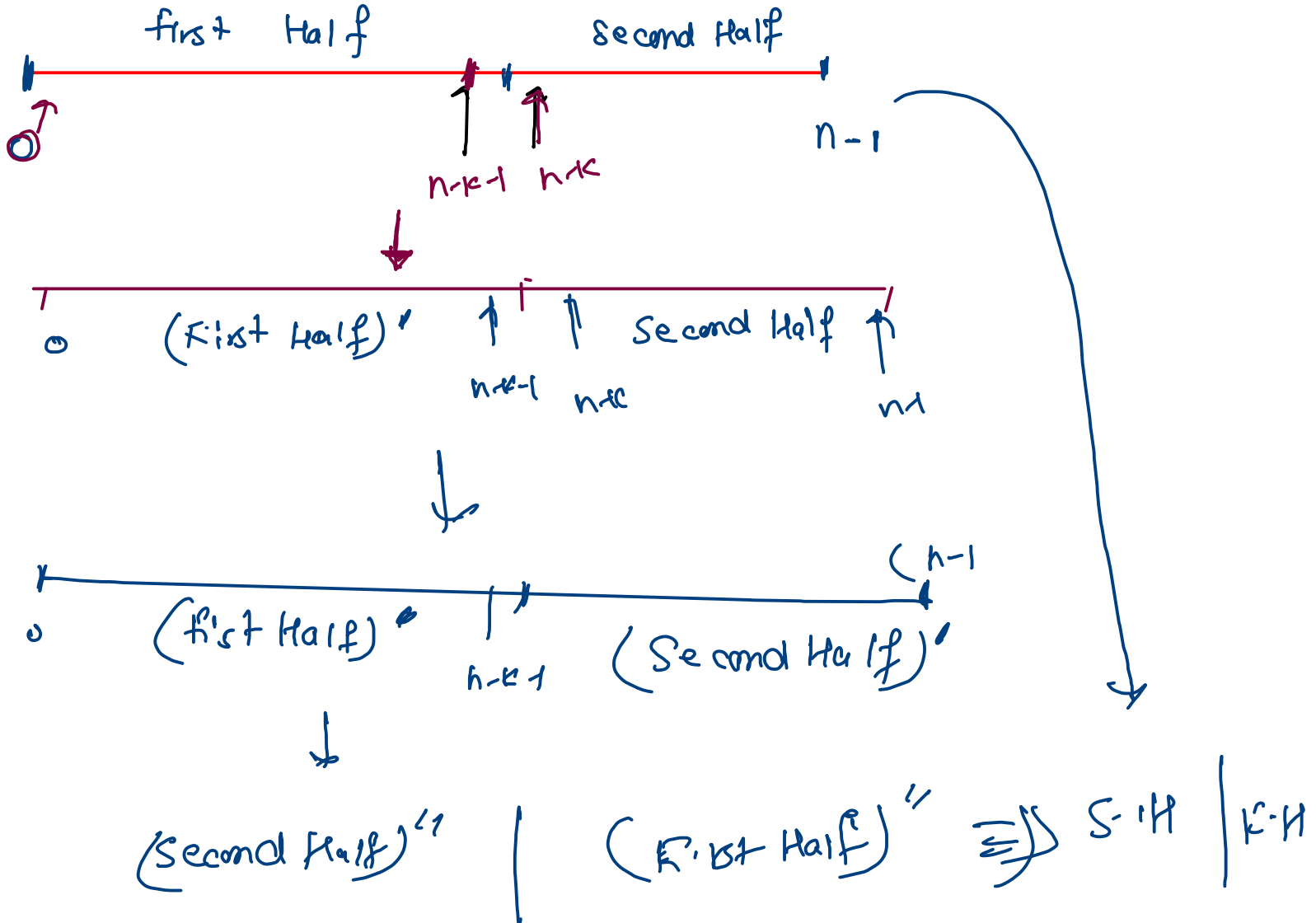
k = 3

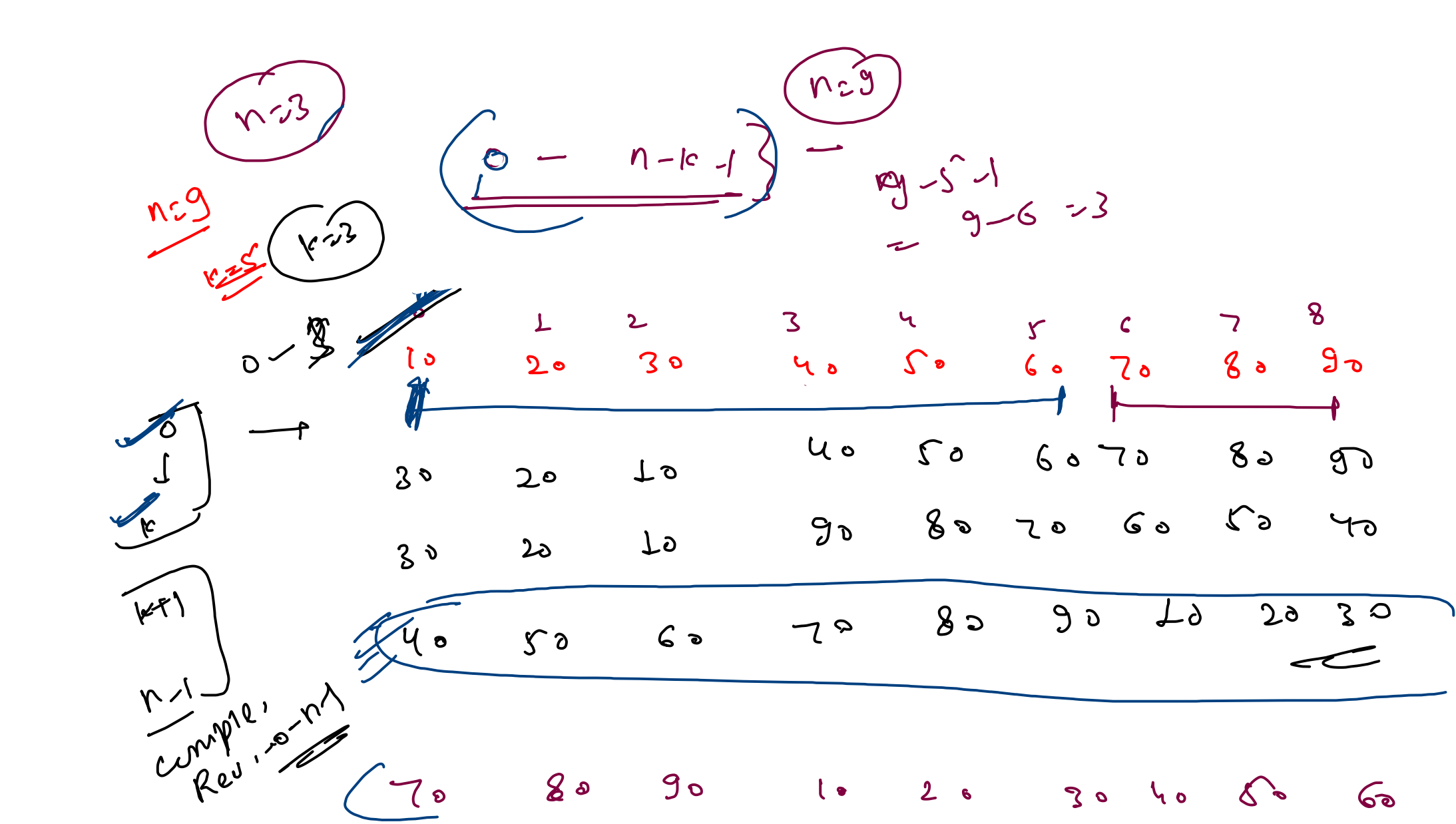
}


k = +ve
across the left
size = n
k element
0
n-1
First Half
n-k-1
n-k Second Half
Reverse First Half (0 ↔ n-k-1)
First Half
Second Half
Reverse Second Half (n-k, n-1)
First Half
Second Half
Complete Reverse (0, n-1)
(Second Half)''
(first half)''
Second Half
First Half
o/p
k = 2
n = 5
0 1 2 3 4
10 20 30 40 50
5-2
30 20 10 40 50
30 20 10 50 40
40 50 10 20 30
arr → 10 20 30
arr → 30 20 10
arr = 10 20 30

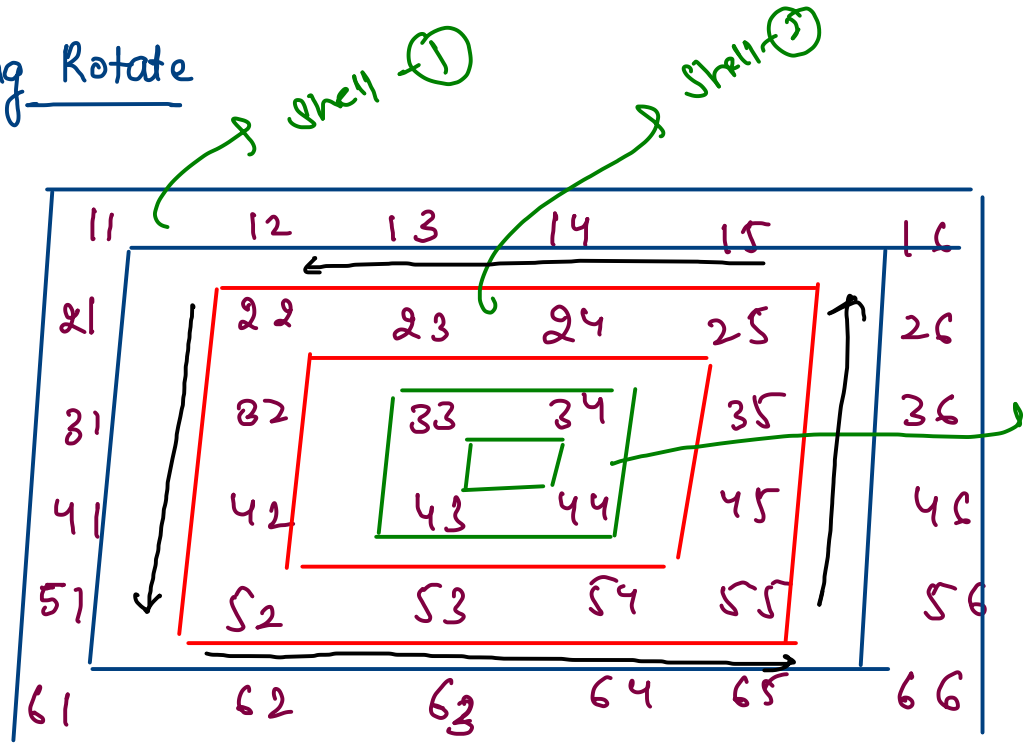
n = arr.length.

k = +ve.




n=3
n=9
k=3
0 — n-k-1
n=9
9-5-1
9-6=3
0 — 3
1 2 3 4 5 6 7 8
10 20 30 40 50 60 70 80 90
0
1
k
30 20 10 40 50 60 70 80 90
30 20 10 90 80 70 60 50 40
k+1
n-1
compile, Rev. 0-n-1
40 50 60 70 80 90 10 20 30
70 80 90 10 20 30 40 50 60

Ring Rotate



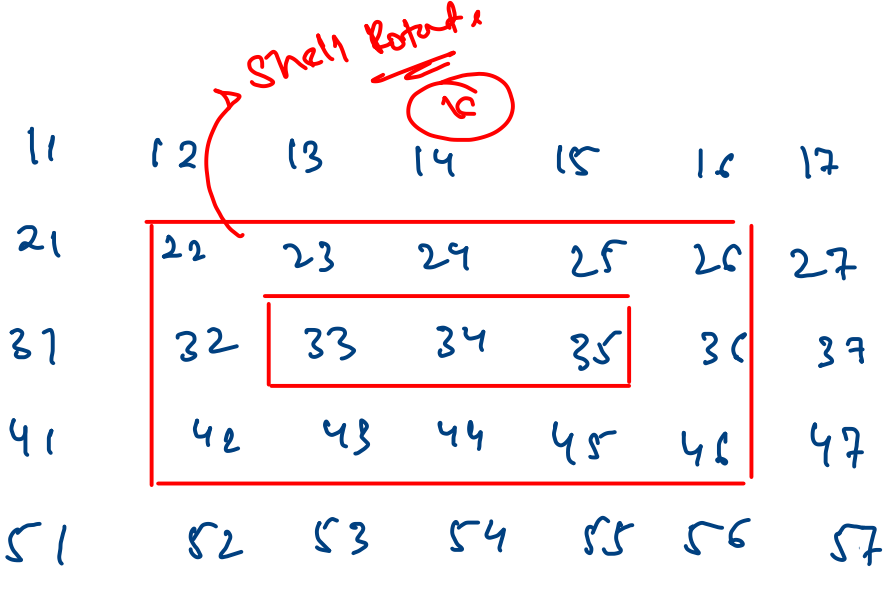
25 35 45 55
24 54
23 53
22 32 42 52

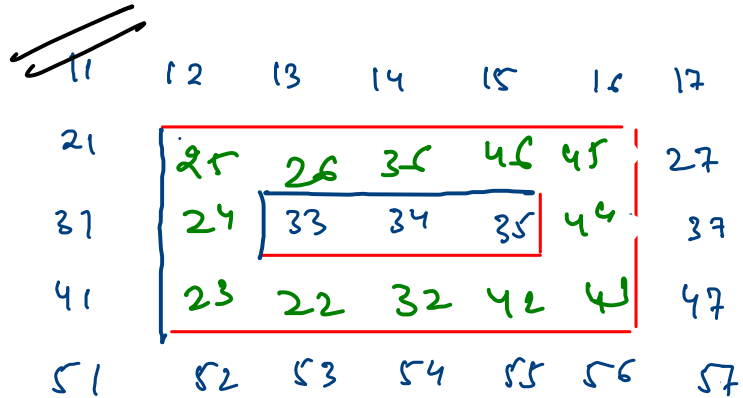
S-2 → 22 32 42 52 53 54 55 45 35 25 24 23

Input — matrix
→ Shell
→ k

k=3 25 24 23 22 32 42 52 53 54 55 45 35

Rotate.
display.





Shell No → 22 32 42 43 44 45 46 36 26 25 24 23

K - Rotate → 25 24 23 22 32 42 43 44 45 46 36 26

k 23

Fill

// left wall

& Bottom wall

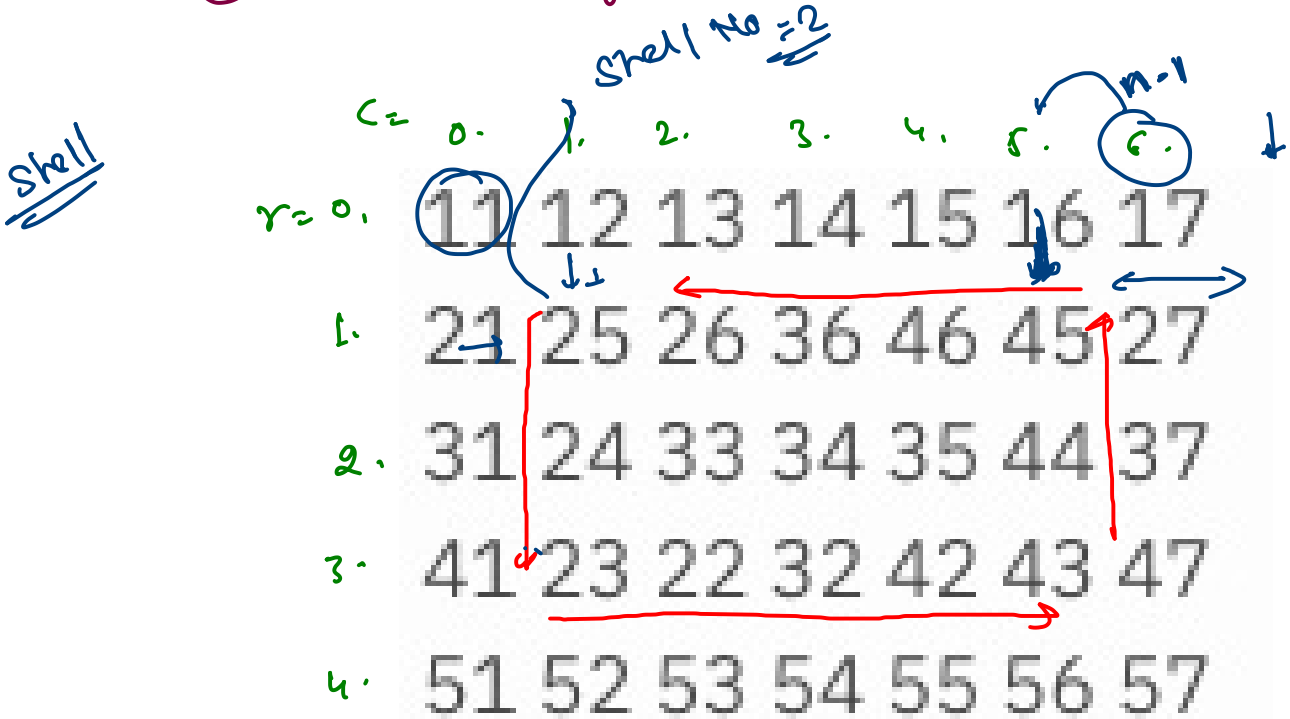
& Right wall

& top wall

Steps: (i) get oneD array from matrix (sth shell)

(ii) Rotate oneD array

(iii) Set oneD array in matrix



cmin = shell - 1;

cmax = colum.length - shell

rmin = shell - 1;

rmax = ro.colength - shell

travel boundary

// left → fix → col (cmin)
vary → Row (min to max)

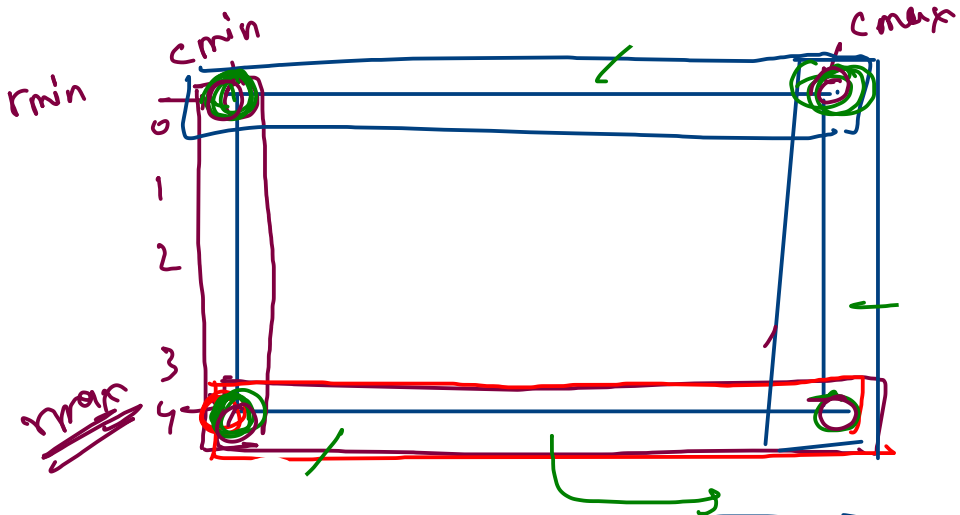
// Bottom → fix → maxRow
vary → col
min to max

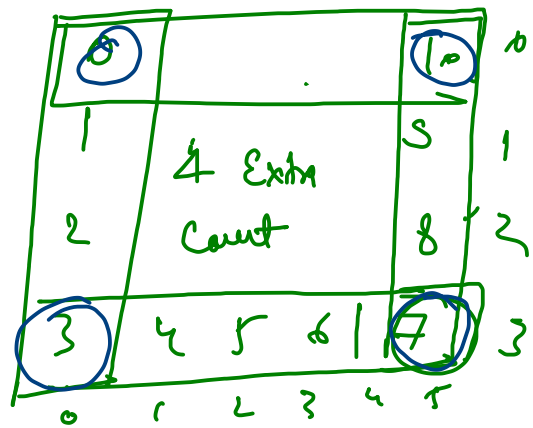
// Right → fix → col (cmax)
vary → row
max - minRow

// top → fix - Row (minRow)
vary → max to min

No. of Elements in s^{th} shell

4 + 6 + 4





$r_{max} - r_{min} + 1$ + $c_{max} - c_{min} + 1$ + $r_{max} - r_{min} + 1$ + $c_{max} - c_{min}$

Left wall — Bottom wall — Right

$= r_{max} - r_{min} + r_{max} - r_{min} + c_{max} - c_{min} + c_{max} - c_{min}$

Left w. — R. w — To p. w. — B. r

$$\text{size} = 2[r_{max} - r_{min} + c_{max} - c_{min}]$$