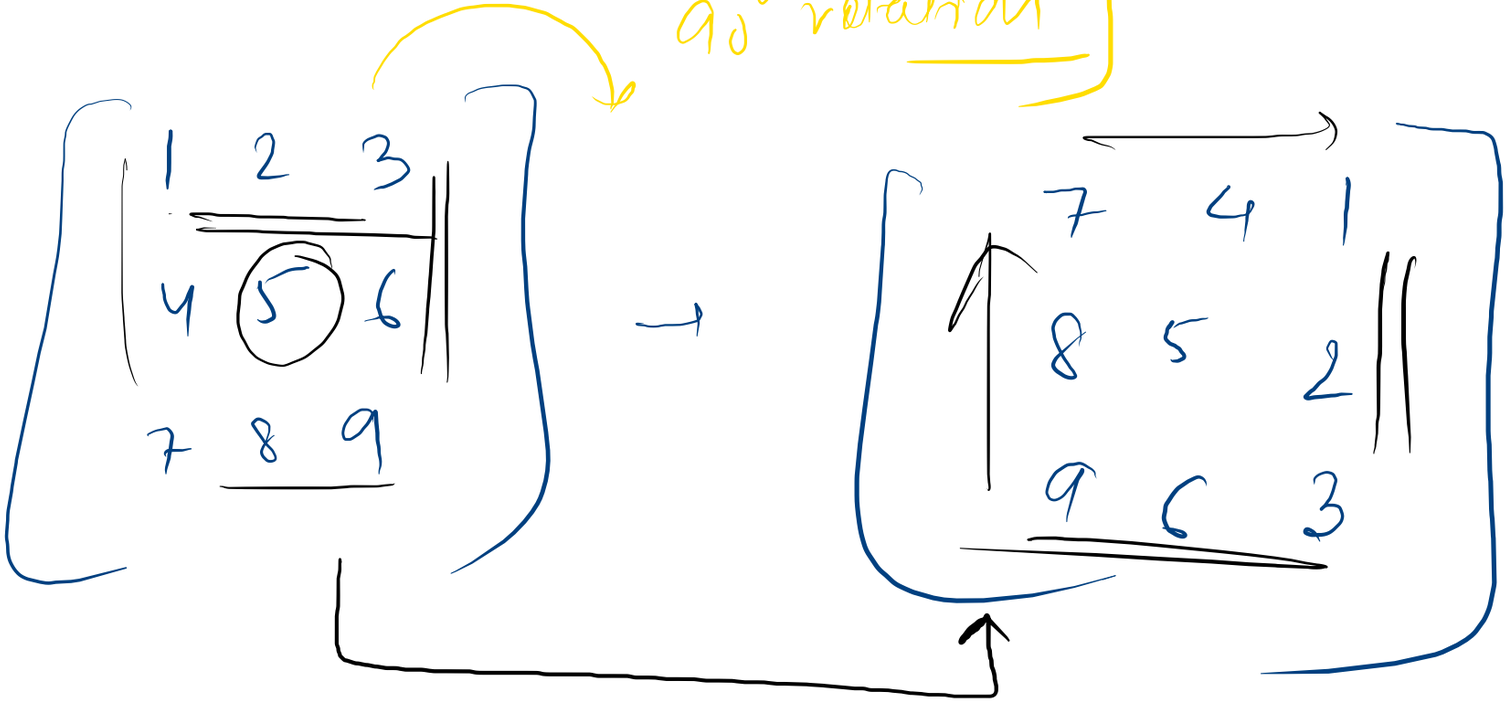
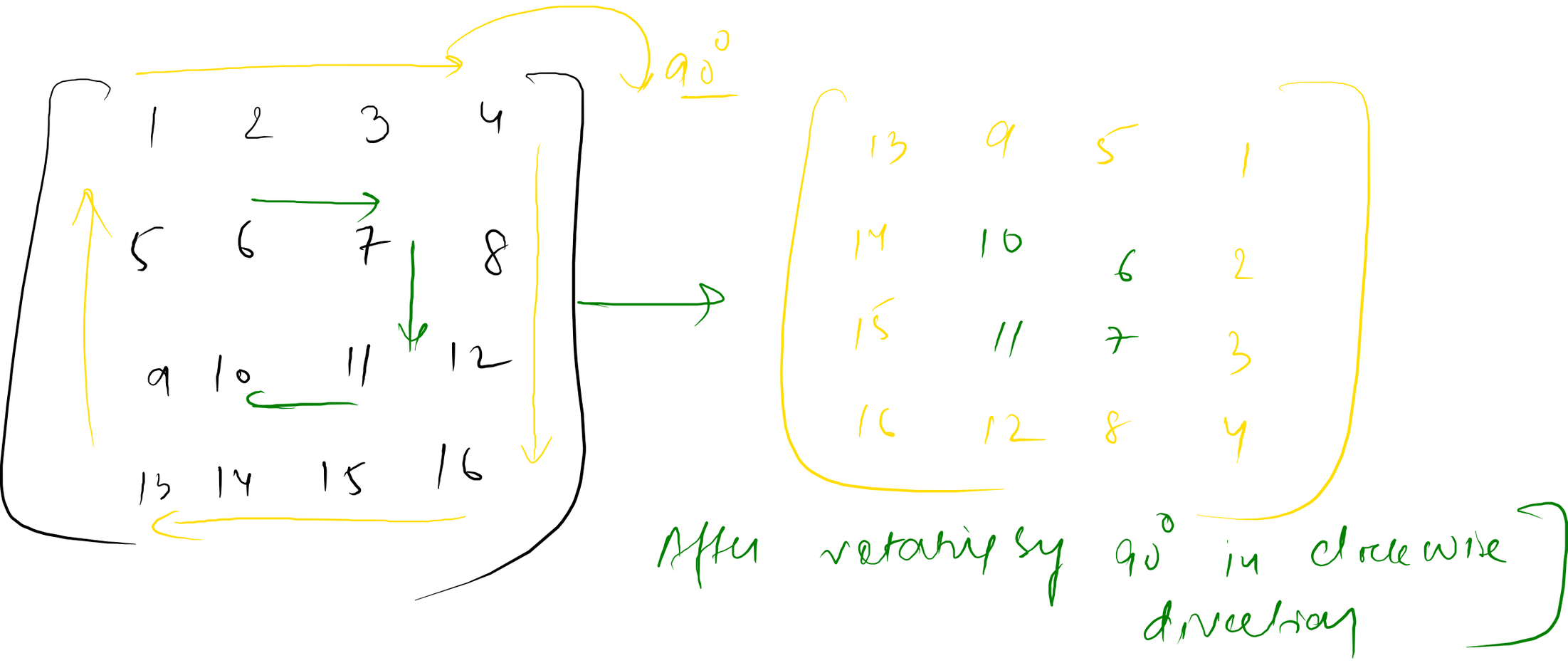
Square Matrix → n × n

90° rotation




90°
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
13 9 5 1
14 10 6 2
15 11 7 3
16 12 8 4
After rotating by 90° in clockwise direction

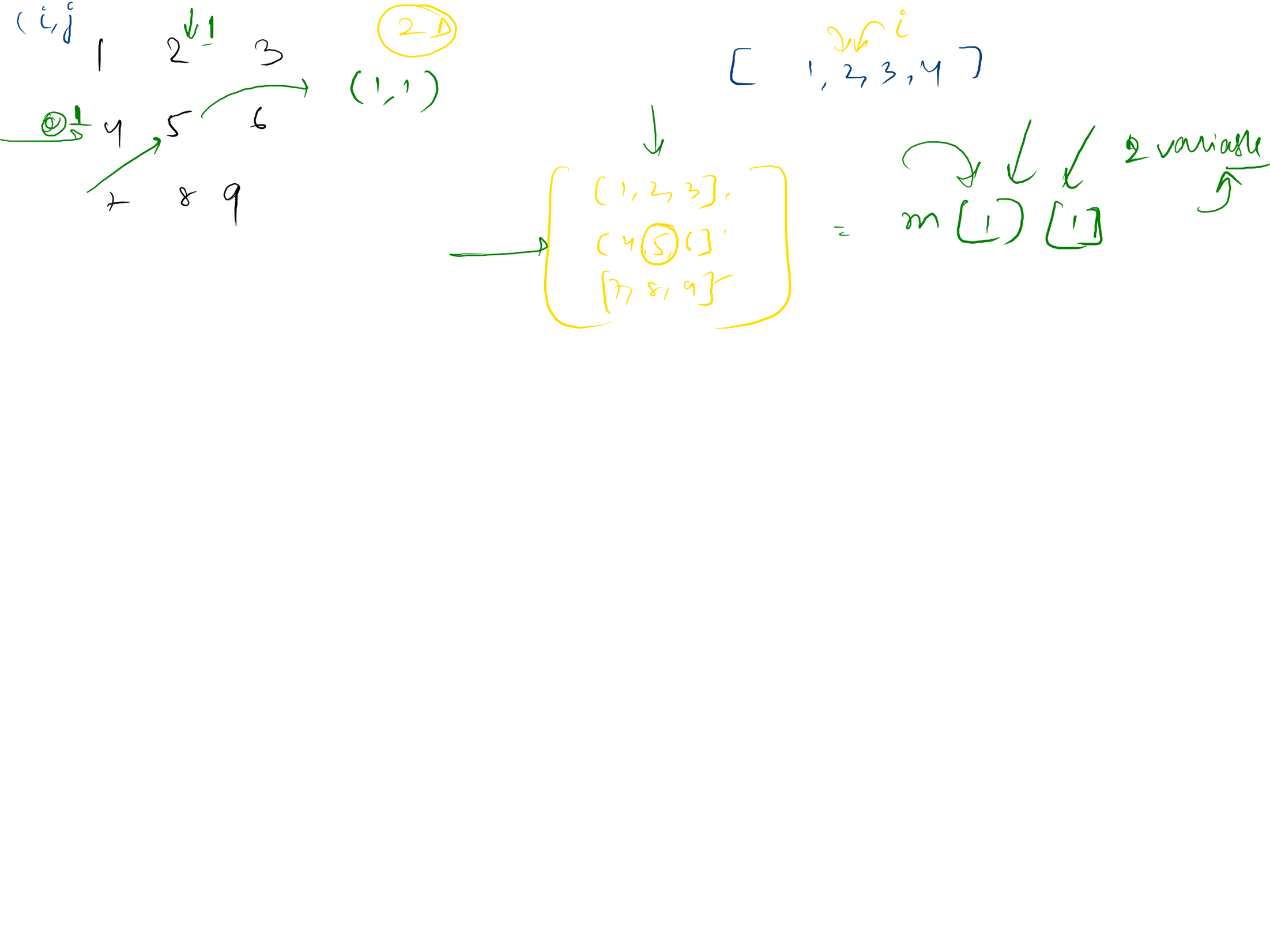
def rotate(mat):

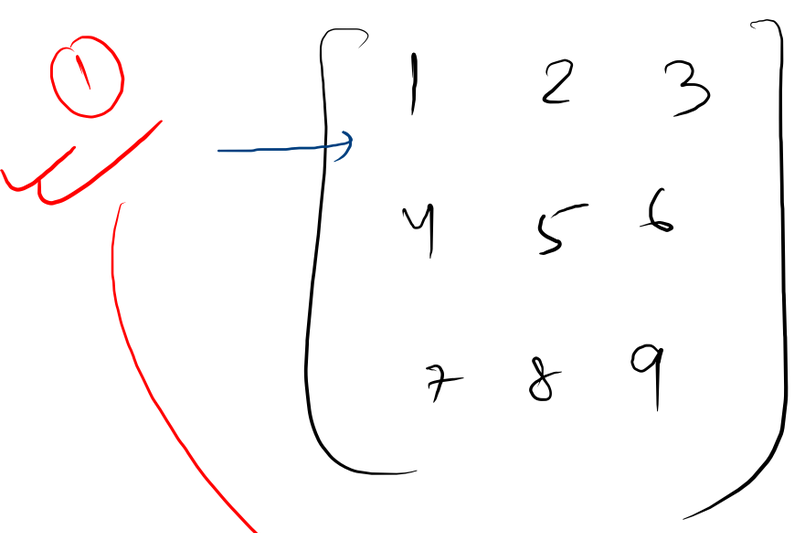
Rotate mat by 90° clockwise

No extra space allowed!

Q: So PM

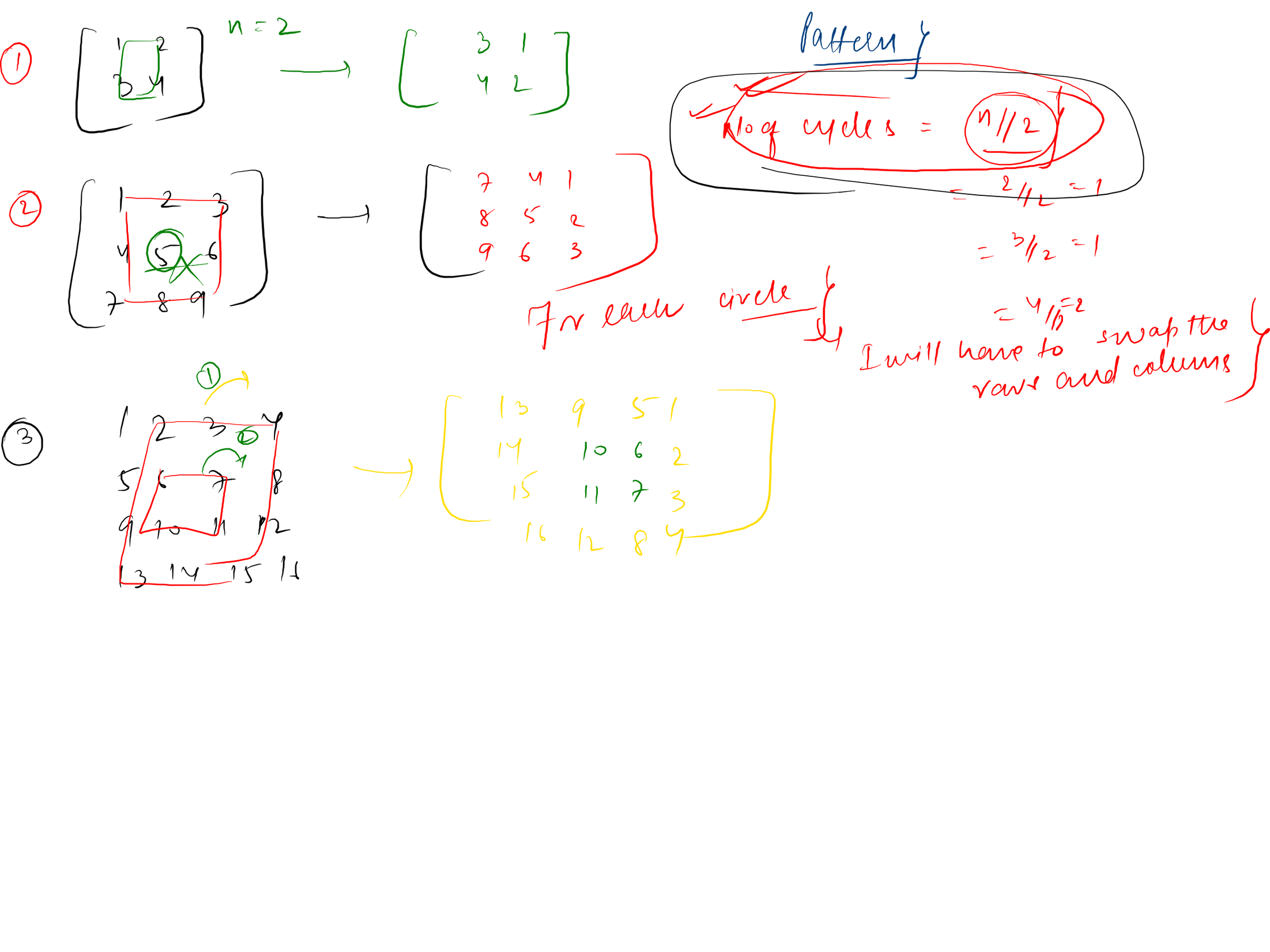
start now!!!


(i, j)
1 2 3
4 5 6
7 8 9
(1, 1)
2 1
[1, 2, 3, 4]
for i
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
= m [1] [1]
2 variable



Print all the elements of this matrix

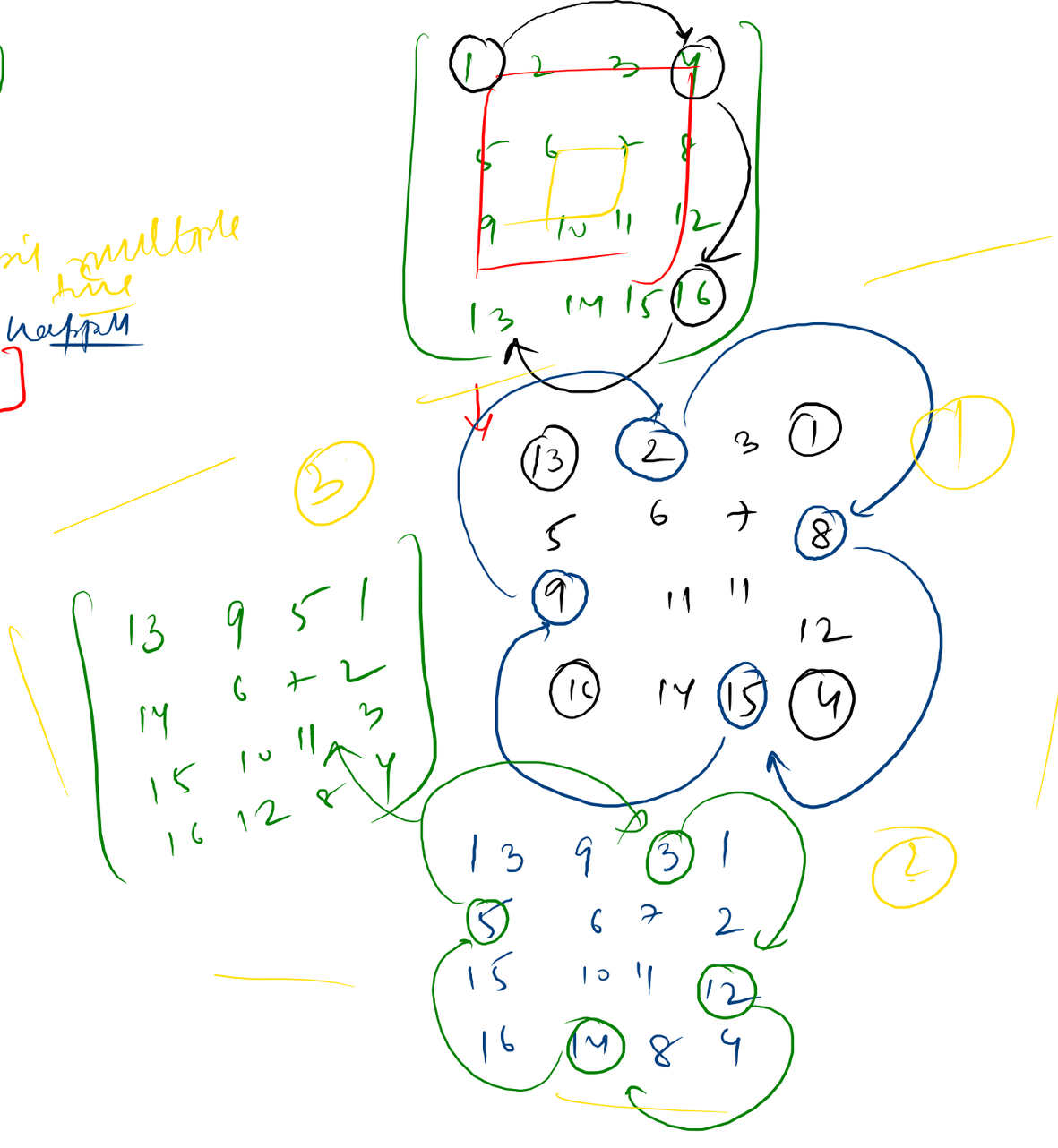
```
for i in range(n):
    for j in range(n):
        print(mat[i][j])
```

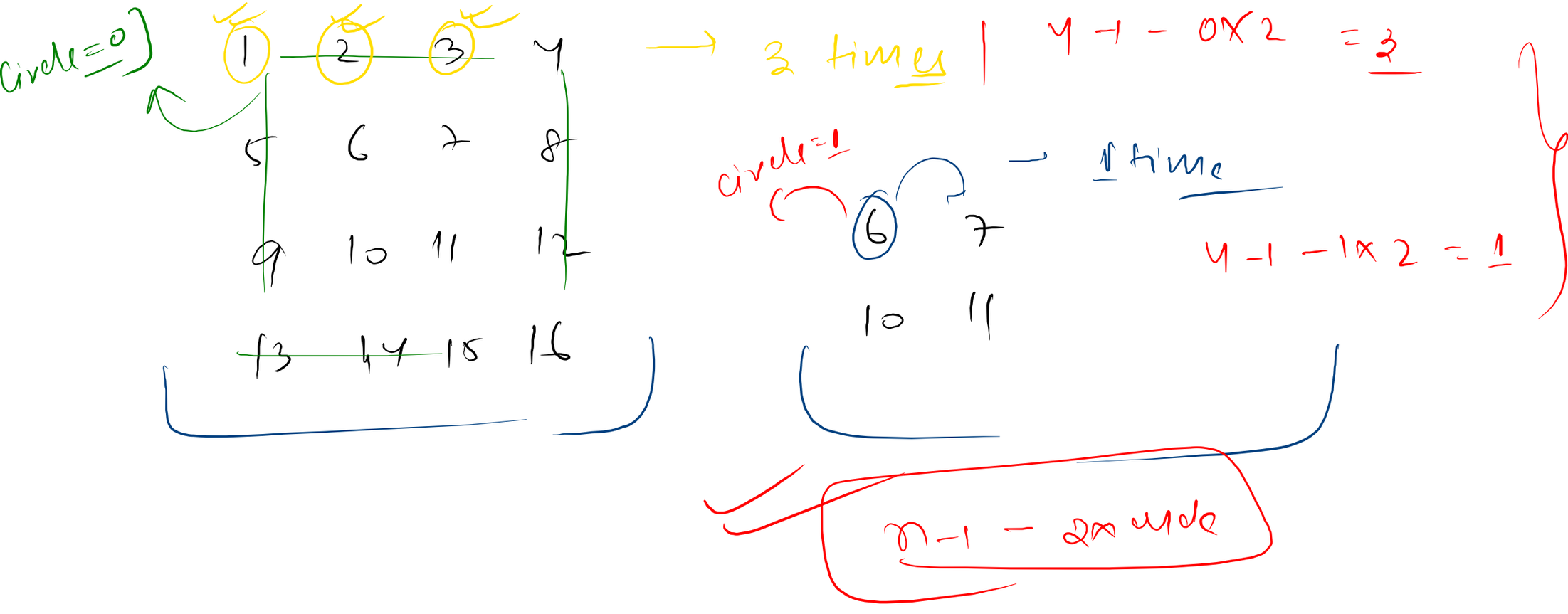

1
n = 2
3 1
4 2
Pattern
No of cycles = n//2
= 2//2 = 1
= 3//2 = 1
= 4//2 = 2
2
7 4 1
8 5 2
9 6 3
For each circle
I will have to swap the rows and columns
3
13 9 5 1
14 10 6 2
15 11 7 3
16 12 8 4

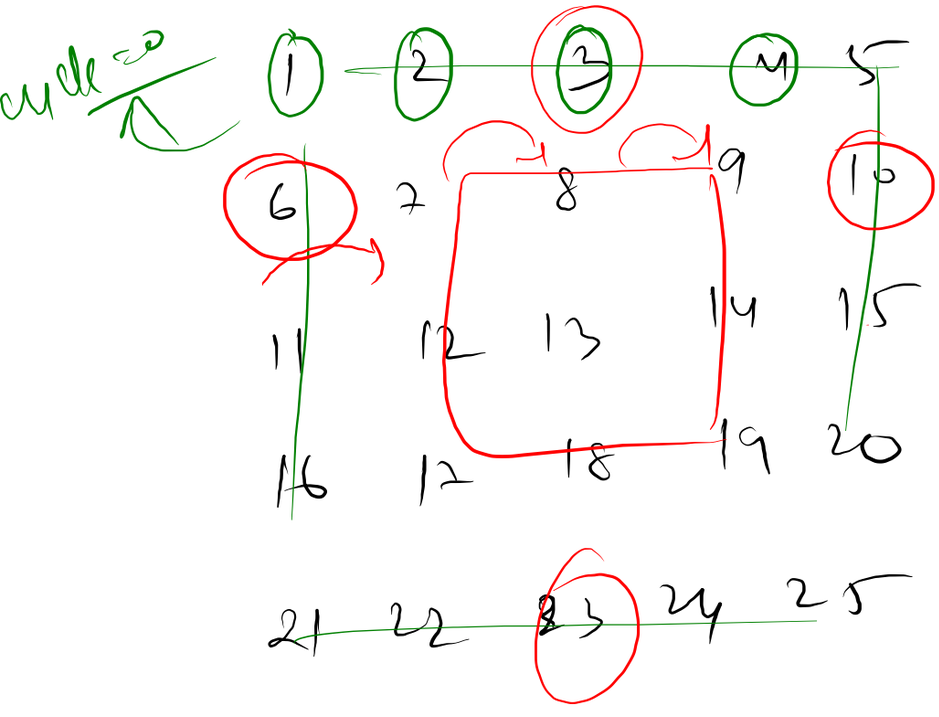
circles = 0 ↷ N/2 → (2 times)

while (circles < N/2) :

keep swapping the elements
rotate the circle → swapping multiple time
How many time swapping should happen
↳ circles and N




circle=0
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
→ 3 times | 4-1-0x2 = 3
circle=1
6 7
10 11
→ 1 time
4-1-1x2 = 1
n-1 - 2x rule



circle → 0 → 1

$n-1-2\times cycle$

$= 5-1-2\times 0 = 4$

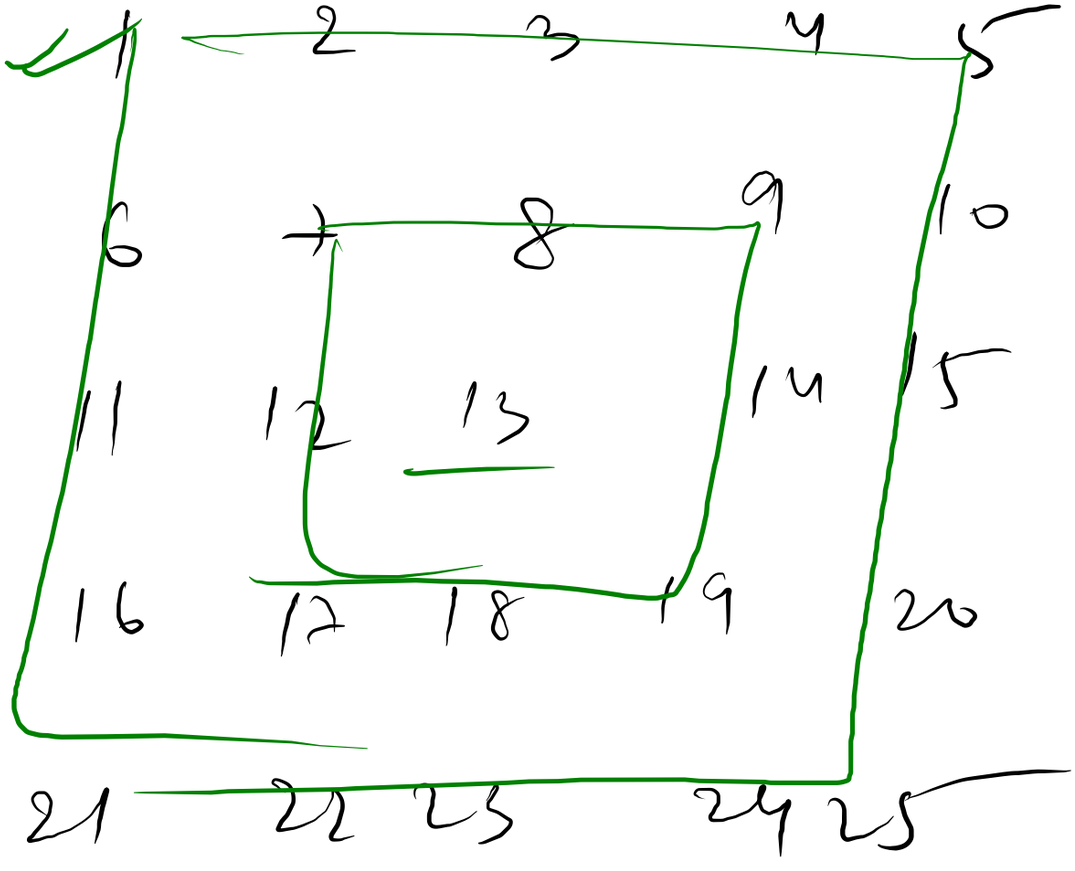
Circle = 1 | 2 times

$5-1-2\times 1$

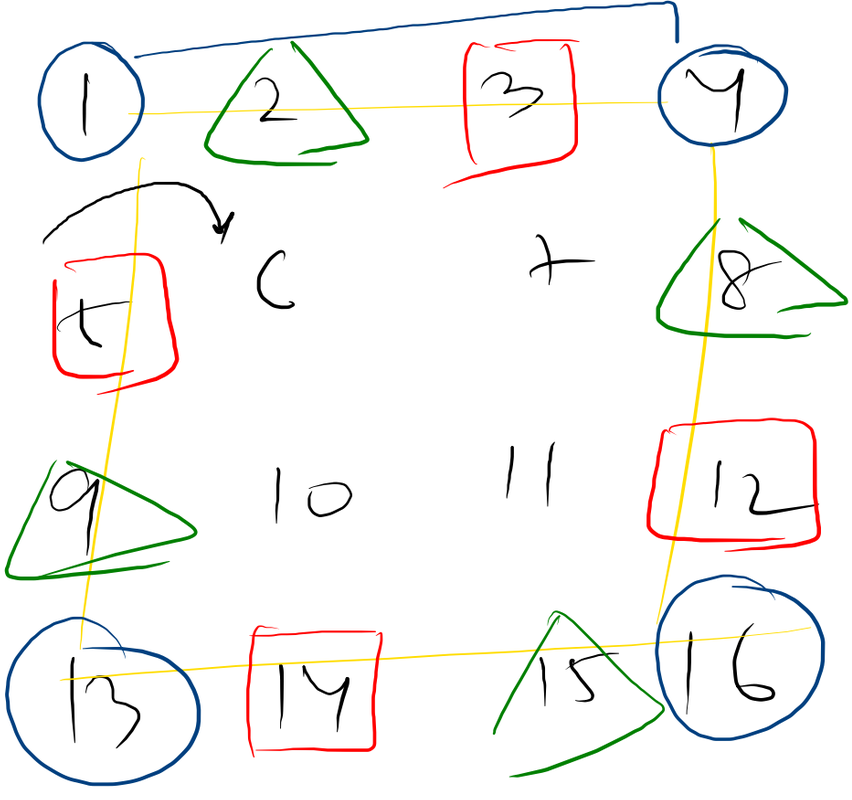
$= 5-1-2 = 2$

Generic formula

$N-1-2\times cycle$



$5 // 2 = 2$



How many time I need to exchange = 3 times

raw = 4

↳ inner = raw - 2

size of circle - 1

3 circles

outer circle raw count = N = n-0

2nd circle will have = N-2 = n-2×1

3rd circle will have = n-4 = n-2×2

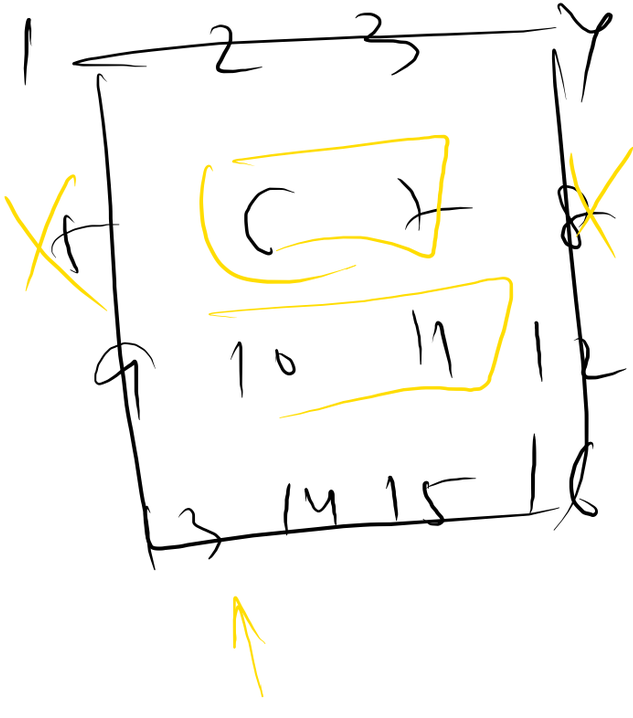
= n - circle × 2

6 7

10 11 [2][2]

[4][4]

Raw



No of swap = size - 1

size of any cycle = n - 2xcycle - 1

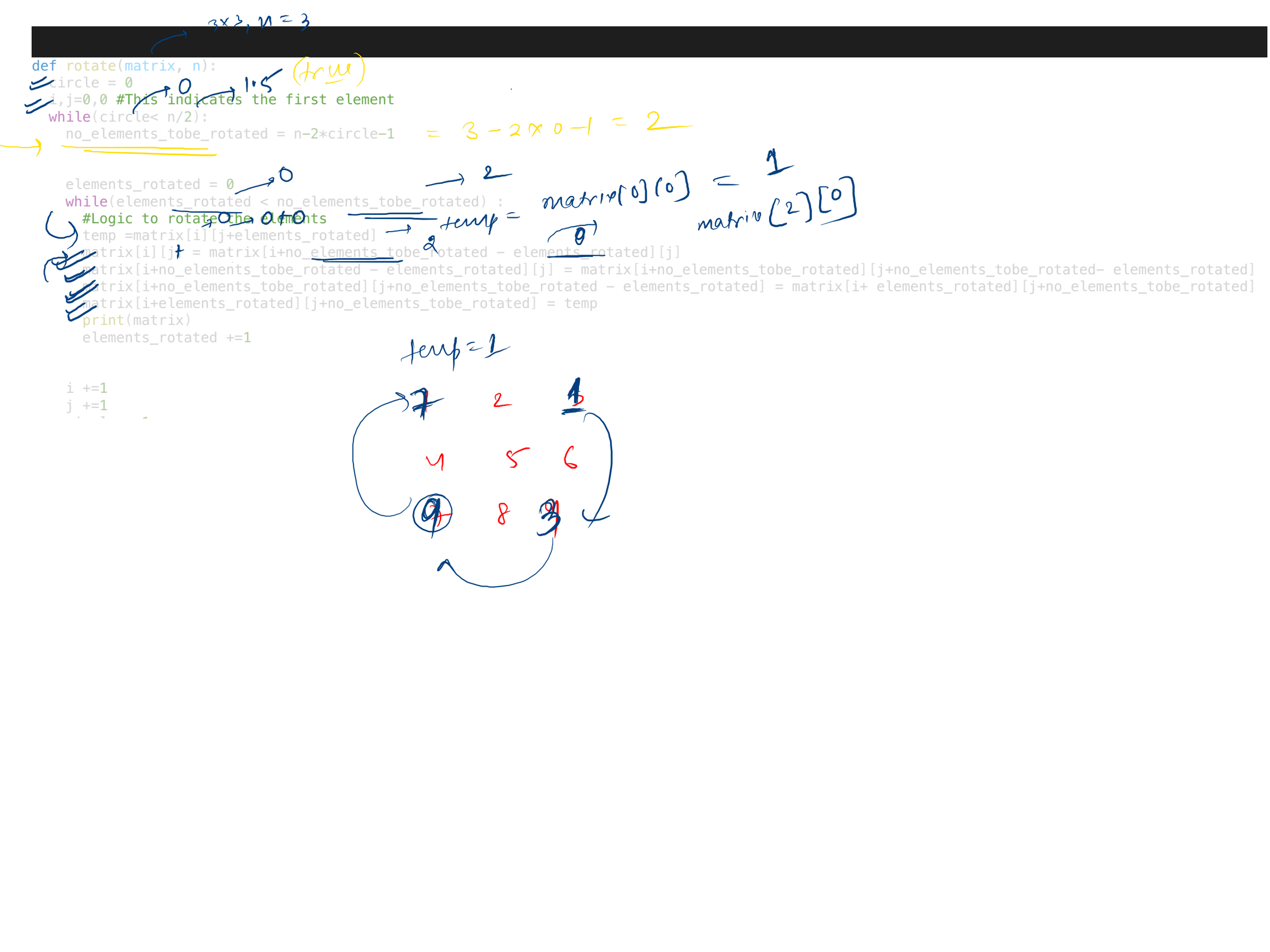
Diagonal problem

for loop has zero perfection

square matrix

```
def rotate(matrix, n):
    circle = 0
    i,j=0,0 #This indicates the first element
    while(circle< n/2):
        no_elements_tobe_rotated = n-2*circle-1
        elements_rotated = 0
        while(elements_rotated < no_elements_tobe_rotated) :
            #Logic to rotate the elements
            temp =matrix[i][j+elements_rotated]
            matrix[i][j+ = matrix[i+no_elements_tobe_rotated - elements_rotated][j]
            matrix[i+no_elements_tobe_rotated - elements_rotated][j] = matrix[i+no_elements_tobe_rotated][j+no_elements_tobe_rotated- elements_rotated]
            matrix[i+no_elements_tobe_rotated][j+no_elements_tobe_rotated - elements_rotated] = matrix[i+ elements_rotated][j+no_elements_tobe_rotated]
            matrix[i+elements_rotated][j+no_elements_tobe_rotated] = temp
            print(matrix)
            elements_rotated +=1


        i +=1
        j +=1
```



Diary → Records

Binary Number ⟷ Decimal no's } How do we do it

123 → Binary no

123

~~128~~ 64 32 16 8 4 2 1

1 1 1 1 0 1 1

1111011

64
32
16
112
8
120

55 → Binary

32	16	8	4	2	1
1	1	0	1	1	1

110111

32
16
48

4
52

1 1 0 1 1 1 ⟶ Number

2^5 2^4 2^3 2^2 2^1 2^0

$1\times2^0 + 1\times2^1 + 1\times2^2 + 0\times2^3 + 1\times2^4 + 1\times2^5$

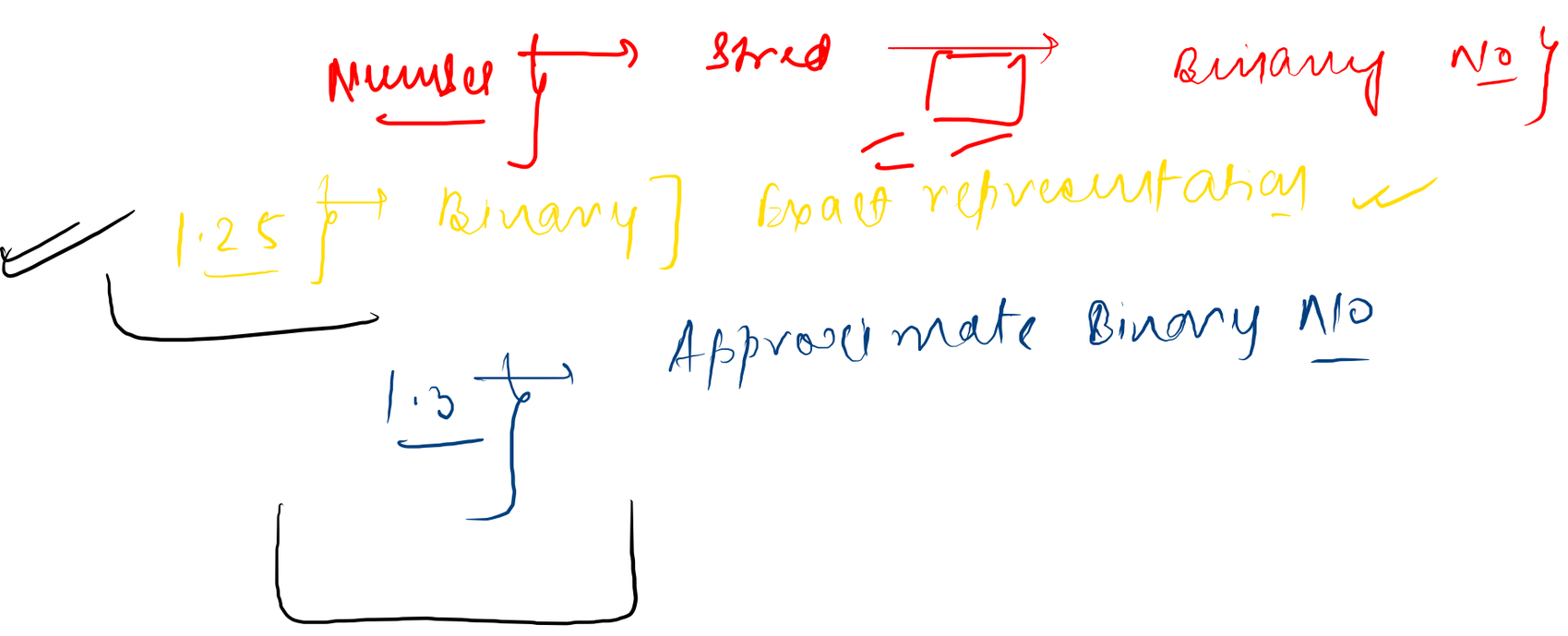
$= 1 + 2 + 4 + 0 + 16 + 32$

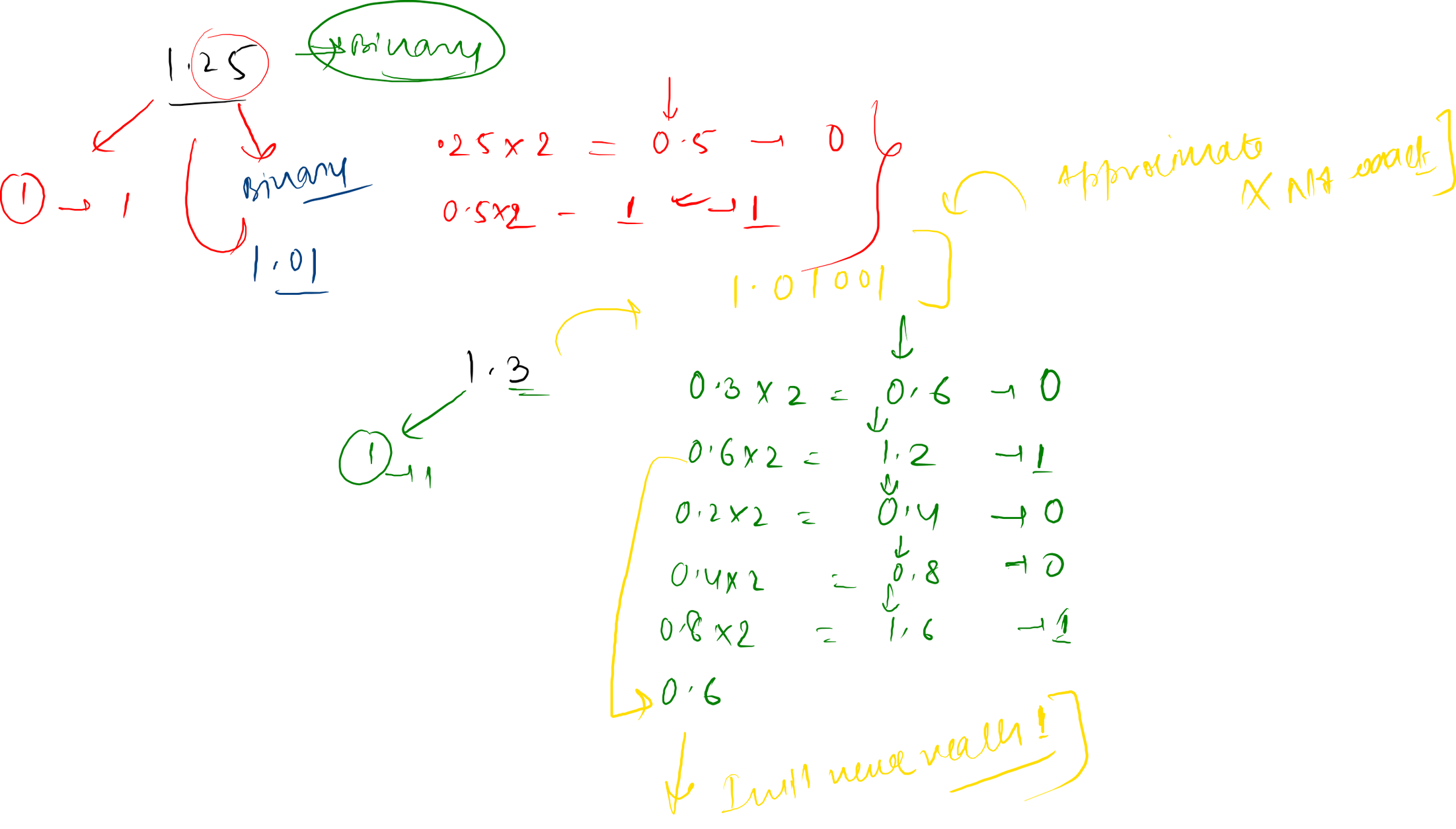
$= \boxed{55}$

Python

print (1 - 0.25 x 4) ⟶ 0.0

print (0.9 - 0.3 x 3) ⟶


Number
Stored
Binary No
1.25
Binary
Exact representation
Approximate Binary No
1.3


1.25
Binary
1
1
Binary
1.01
.25 x 2 = 0.5 → 0
0.5 x 2 = 1 → 1
1.01001
Approximate
NOT exact
1.3
1
1
0.3 x 2 = 0.6 → 0
0.6 x 2 = 1.2 → 1
0.2 x 2 = 0.4 → 0
0.4 x 2 = 0.8 → 0
0.8 x 2 = 1.6 → 1
0.6
Will never really!

A_{bpr} $0.9 - 3 \times 0.3$ A_{bpn}

$\Rightarrow$ very small value $\neq 0$

Count sort

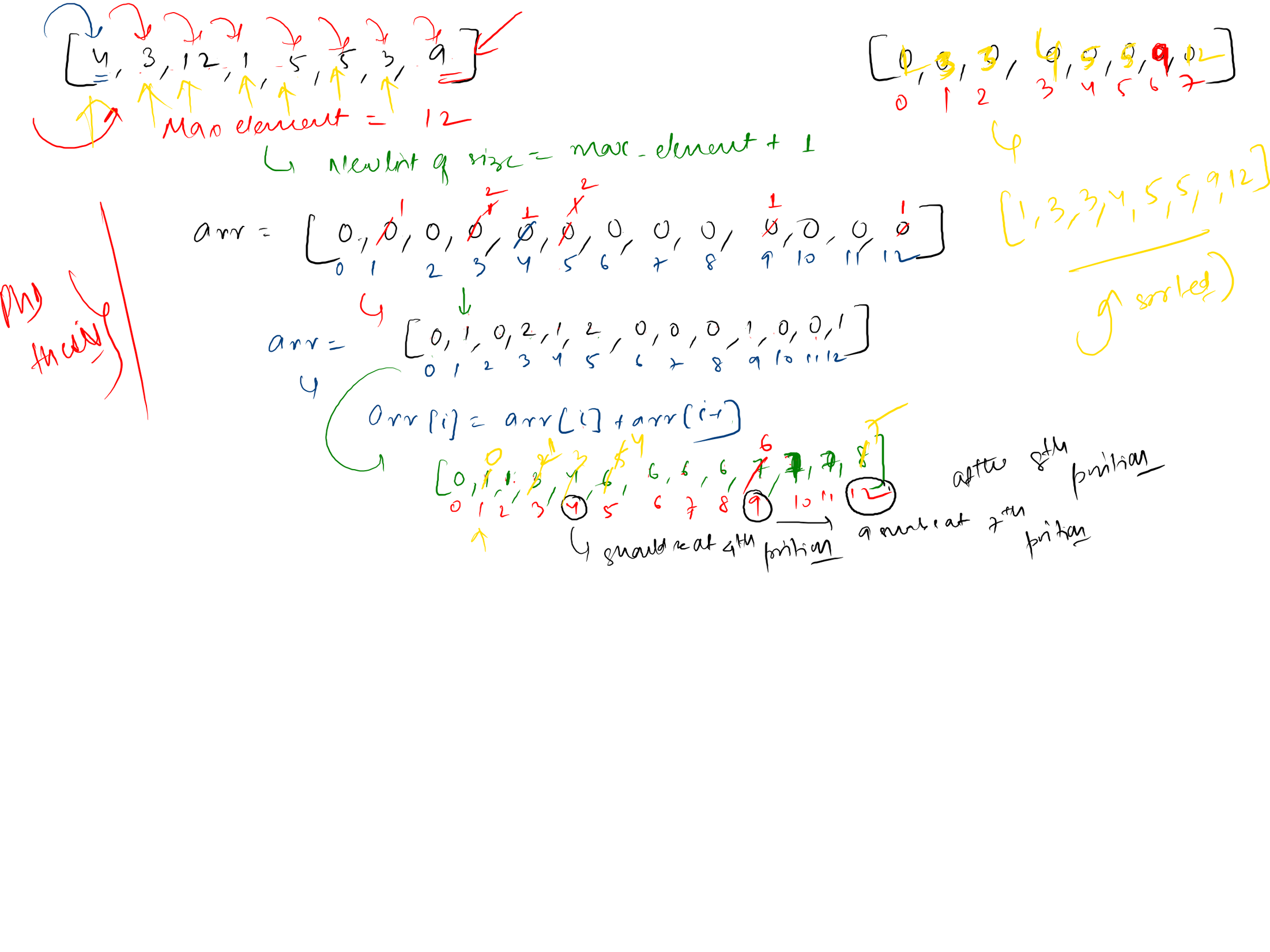
$O(n^2)$ → Bs
→ selection
→ Insertion sort

$O(n \log n)$ → Merge
→ Quick

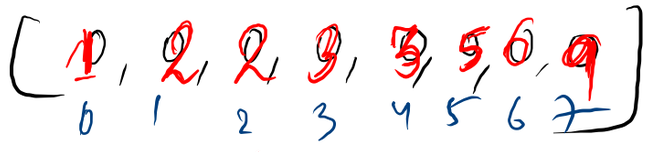
$O(n)$

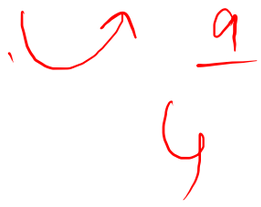
Count Sort → O(n)

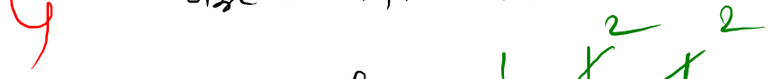
1. Array of Nos
2. Nos given should not be of very high value

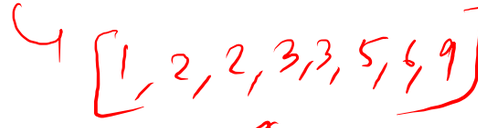

[4, 3, 12, 1, 5, 5, 3, 9]
Max element = 12
↳ New list of size = max_element + 1
arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
0 1 2 3 4 5 6 7 8 9 10 11 12
arr = [0, 1, 0, 2, 1, 2, 0, 0, 0, 1, 0, 0, 1]
arr[i] = arr[i] + arr[i-1]
[0, 1, 1, 3, 4, 6, 6, 6, 7, 7, 7, 8, 9]
↳ should be at 4th position
a number at 7th position
after 8th position
[0, 3, 3, 4, 5, 5, 9, 12]
[1, 3, 3, 4, 5, 5, 9, 12]
sorted
Phy theory

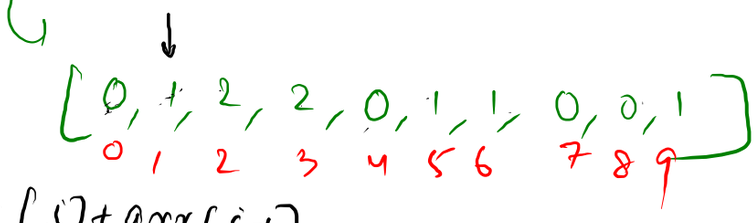
O(n)

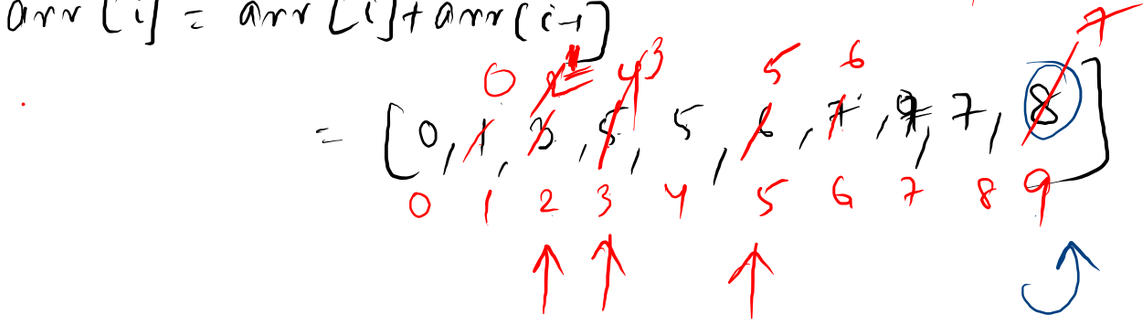












```python
def count_sort(input_arr):

    M = max(input_arr)

    #Create a new list of the size M+1
    count_arr = [0]*(M+1)

    for num in input_arr :
        count_arr[num] += 1


    # We need to find the prefix sum
    for i in range(1, M+1):
        count_arr[i] +=count_arr[i-1]


    #Create a final output array
    out_arr = [0]*len(input_arr)

    for i in range(len(input_arr)-1, -1, -1):
        out_arr[count_arr[input_arr[i]]-1] = input_arr[i]
        count_arr[input_arr[i]] -=1

    return out_arr


print(count_sort([4,3,1,1,7,8,6]))
```

count_sort([4, 3, 1, 1, 7, 8, 6])

count_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0]
0 1 2 3 4 5 6 7 8

count_arr = [0, 2, 0, 1, 1, 0, 1, 1, 1]

count_arr [0, 2, 2, 3, 4, 4, 5, 6, 7]
0 1 2 3 4 5 6 7 8

[4, 3, 1, 1, 7, 8, 6]

out_arr = [1, 1, 3, 4, 6, 7, 8]
0 1 2 3 4 5 6

[1, 1, 3, 4, 6, 7, 8]

```python
def count_sort(input_arr):

    M = max(input_arr)

    #Create a new list of the size M+1
    count_arr = [0]*(M+1)

    for num in input_arr :
        count_arr[num] += 1

    # We need to find the prefix sum
    for i in range(1, M+1):
        count_arr[i] +=count_arr[i-1]

    #Create a final output array
    out_arr = [0]*len(input_arr)

    for i in range(len(input_arr)-1, -1, -1):
        out_arr[count_arr[input_arr[i]]-1] = input_arr[i]
        count_arr[input_arr[i]] -=1

    return out_arr


print(count_sort([4,3,1,1,7,8,6]))
```

count_arr ([5, 3, 1, 5, 2, 3])

count_arr = [0, 0, 0, 0, 0, 0]

count_arr = [0, 1, 2, 0, 2]

0 1 2 3 4 5

[1, M]

count_arr[i] = count_arr[i] + count_arr[i-1]

count_arr = [0, 1, 2, 2, 4, 5]

0 1 2 3 4 5

out_arr = [1, 2, 3, 3, 5, 5]

0 1 2 3 4 5

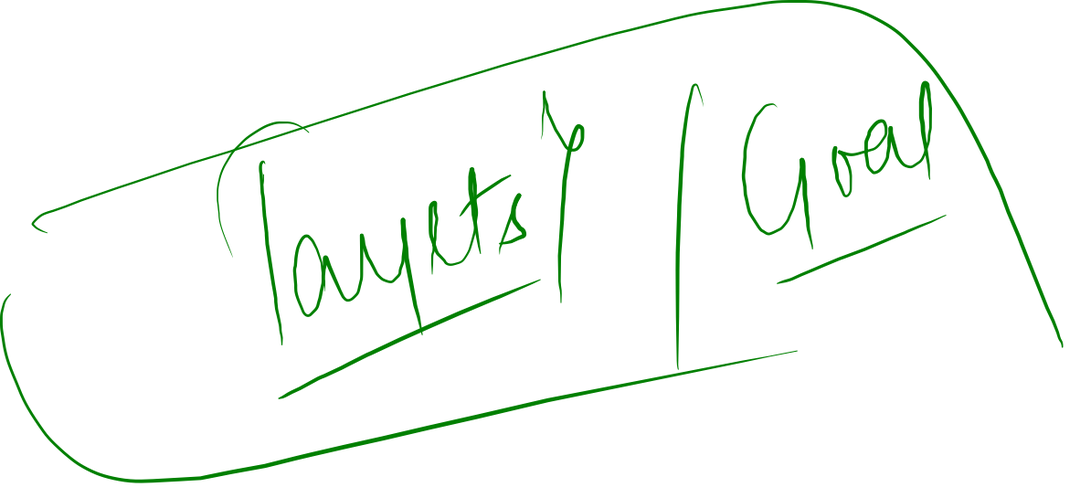
input_arr = [5, 3, 1, 5, 2, 3]

0 1 2 3 4 5

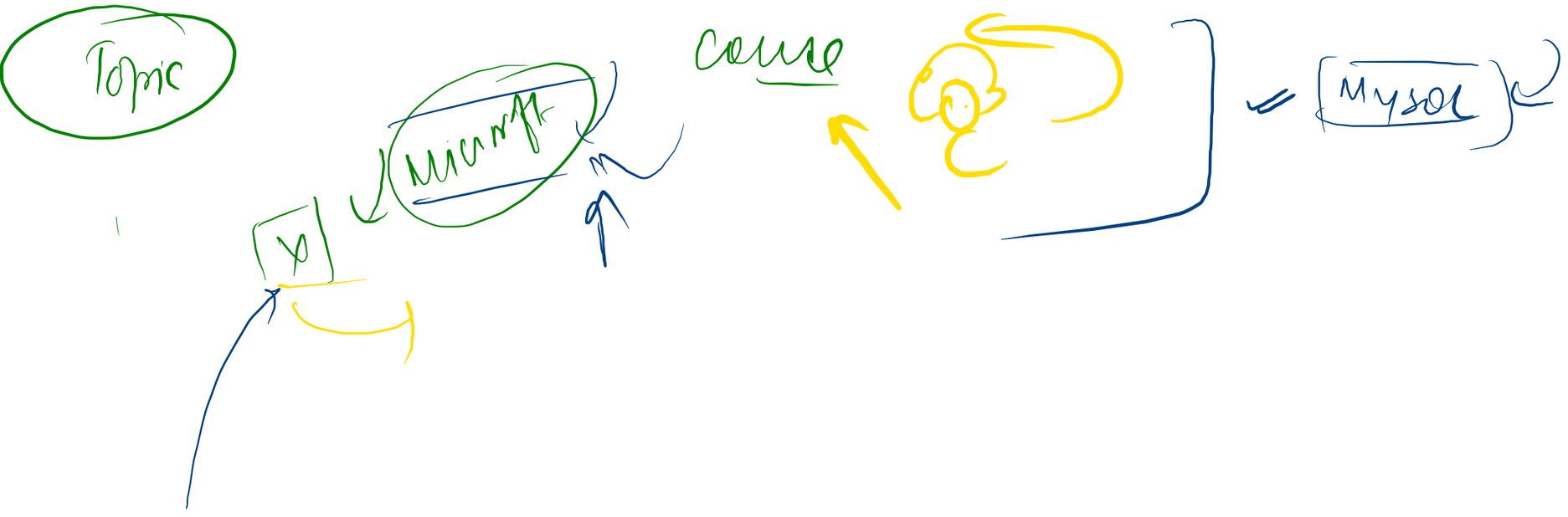
[1, 2, 3, 3, 5, 5]

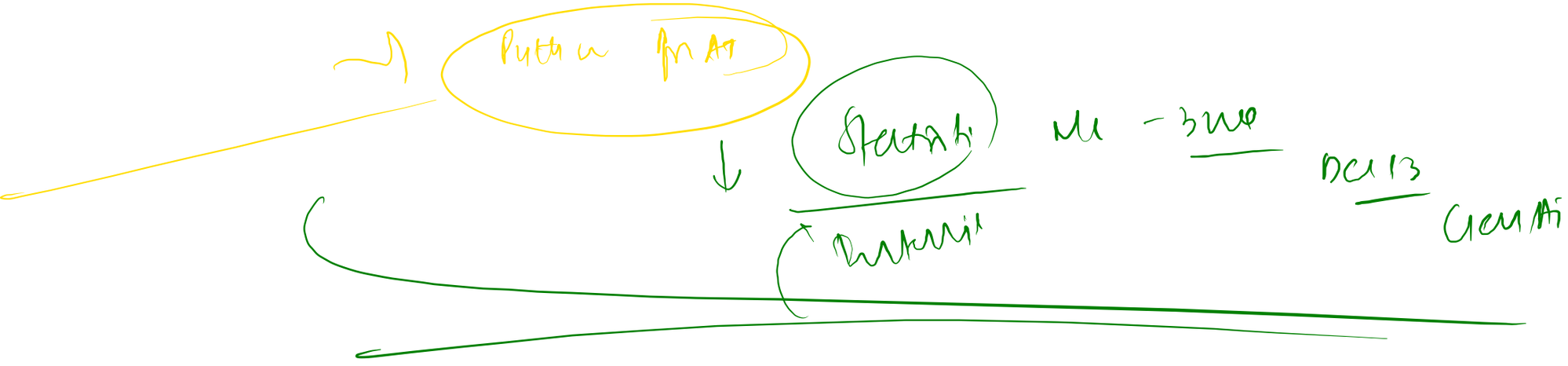
sorted

Every week } Revision DSA }


Targets / Goal

(LMS) → Curriculum


Topic
Consumer
MySQL



count

① Numbers

② we should only use if we have smaller no's

circle = 0

while (circle < N/2):

    jumps = N - 1 - 2 * circle
    count = 0
    while (count < jumps):