Stack } FILO / LIFO }

insert → push()

remove → pop()

① Array

Limitation :-

↓

fixed size

space

② Linked list

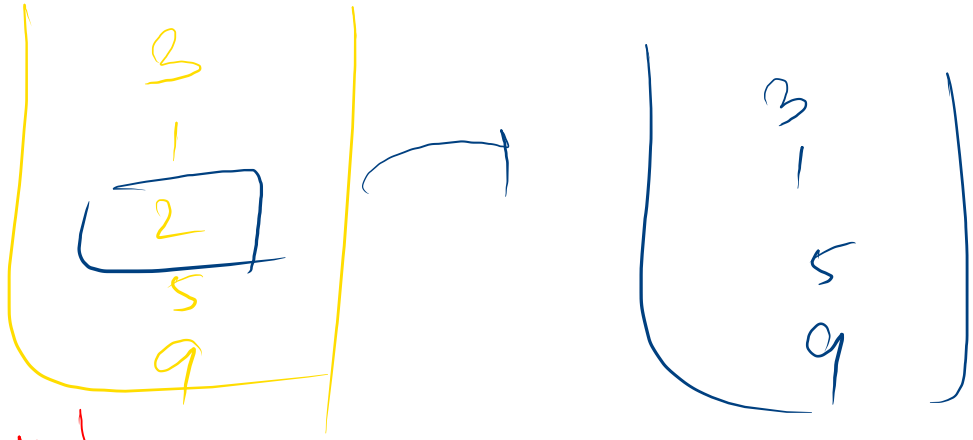
Pr 1 → Balance the parantheis

↳ Stack

Aduieve

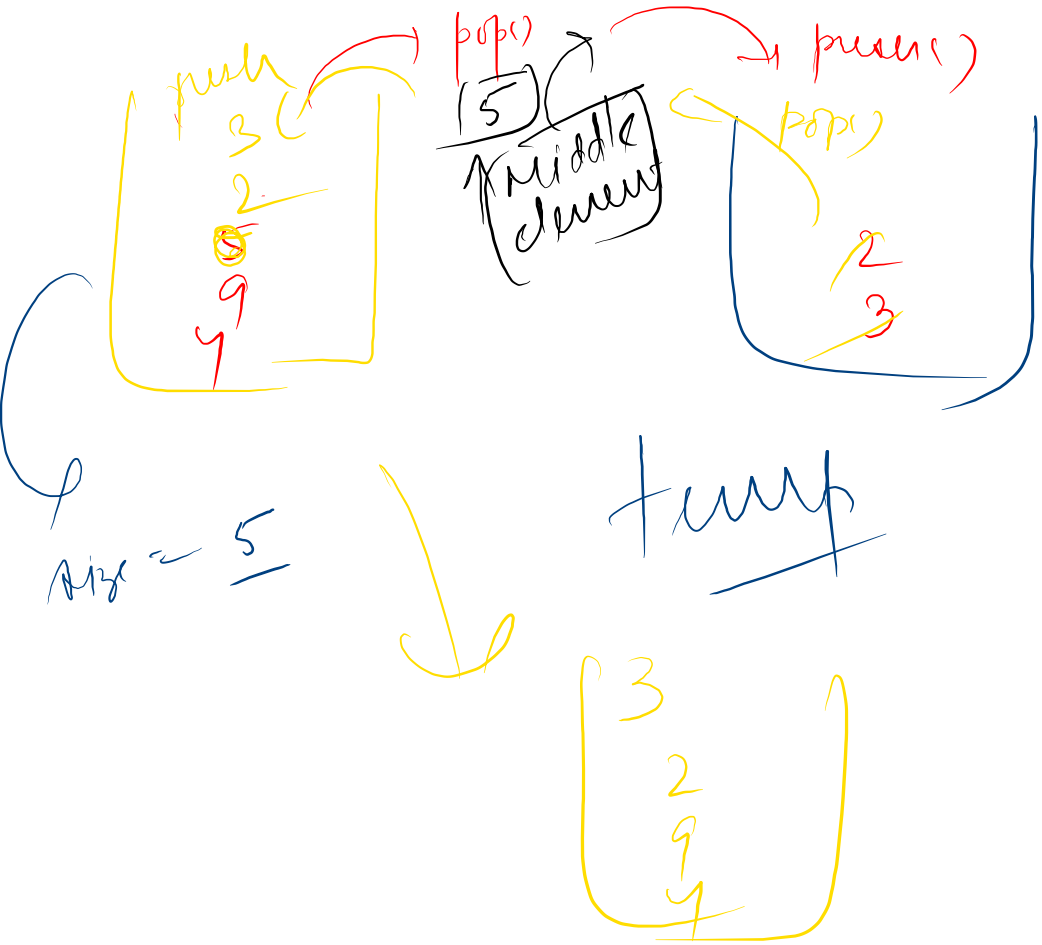
stacle

Delete the middle element

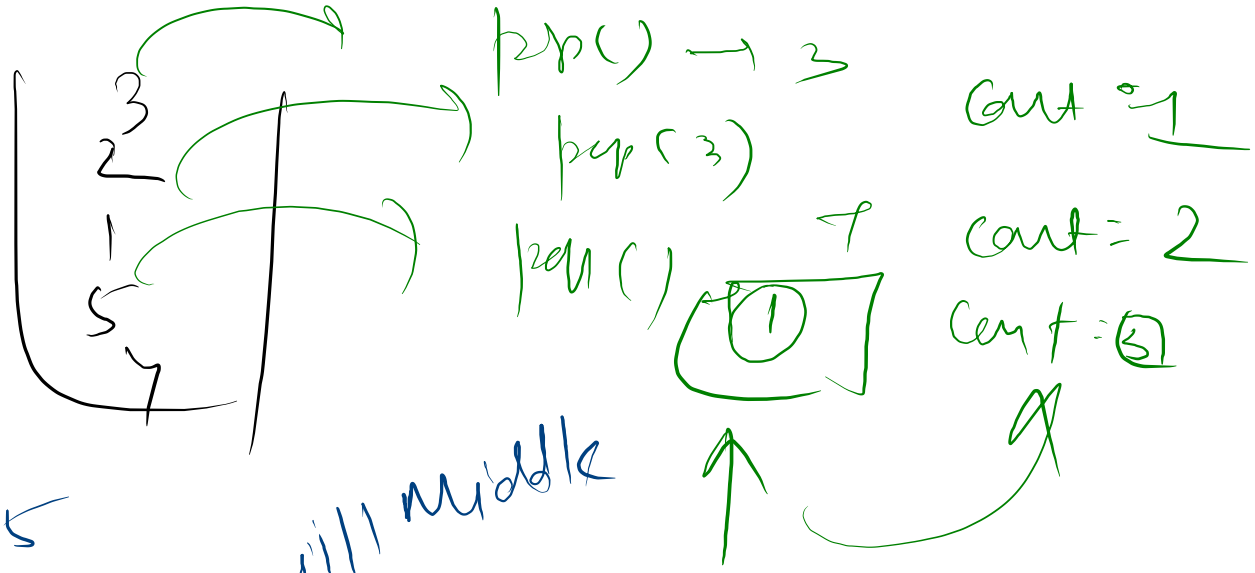


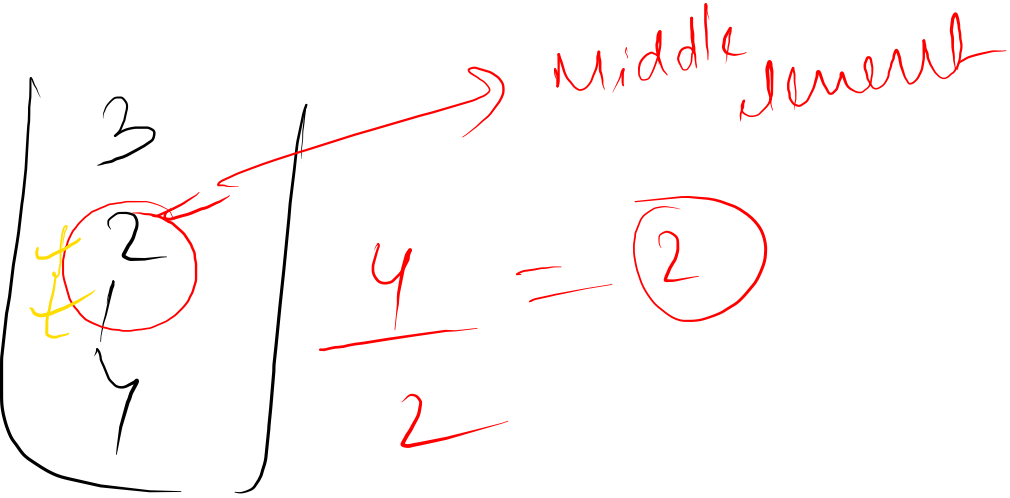
push, pop, empty, size

BS → sorted Array


push
pop()
peek()
3
2
5
9
4
5
middle element
pop()
2
3
Size = 5
temp
3
2
9
4

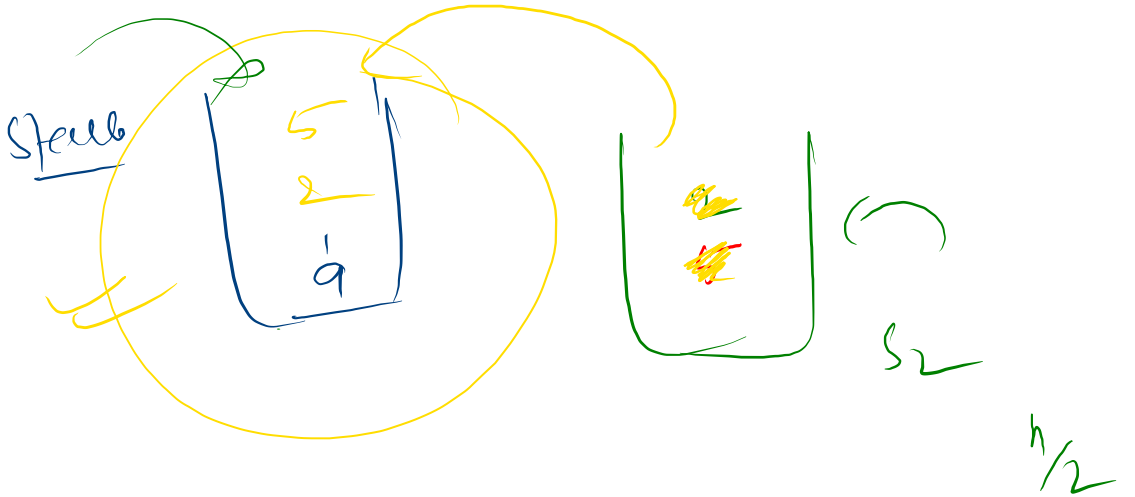
Stack
Array
LL


3
2
1
5
pop() → 3
pop(3)
pop()
1
count = 1
count = 2
count = 3
Size = 5
3rd element will middle
Midd


3
2
1
4
middle number
4/2 = 2

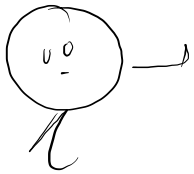
Approach 6 → using 2 stacks

```
def deleteMiddle(stack):
  #Implement this method -> push/pop/peek
  s2 = StackLL()

  size = stack.size()

  count = 0

  while(count < int(size/2)):
    s2.push(stack.pop())
    count +=1
  stack.pop()
  while(not s2.isEmpty()):
    stack.push(s2.pop())
```



TC = O(n)

SC = O(n)



Additional stack

No stack allowed

stack

push() pop()

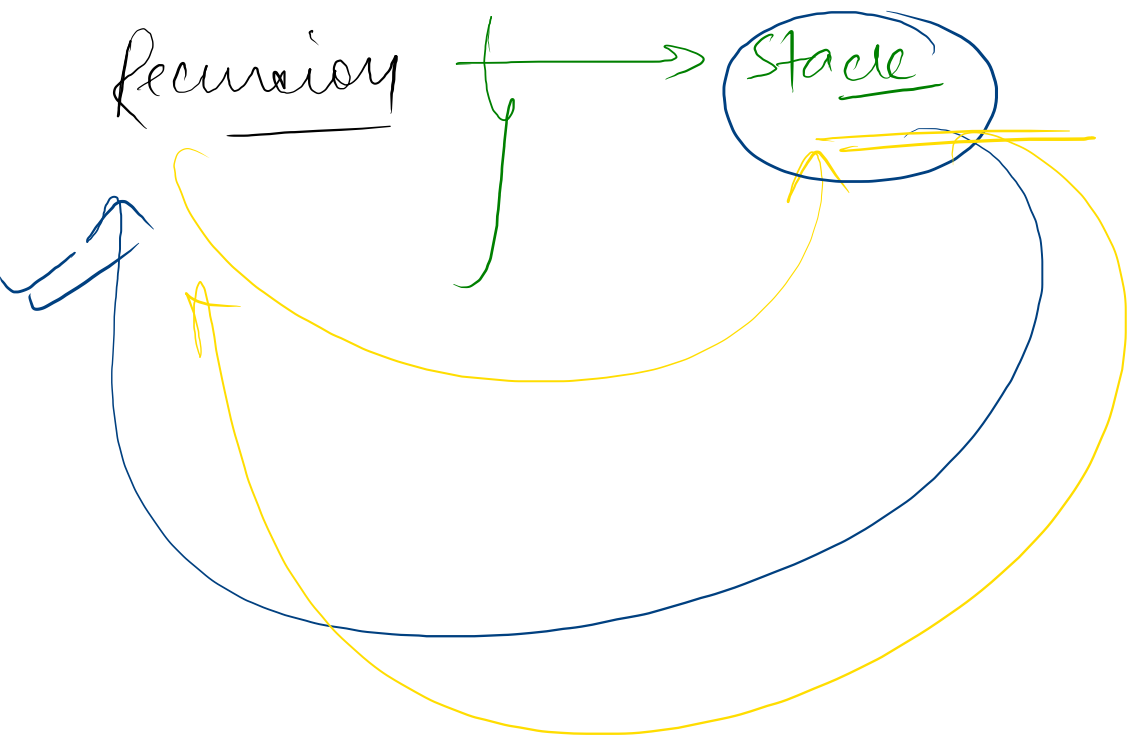
9
5
1
2
3

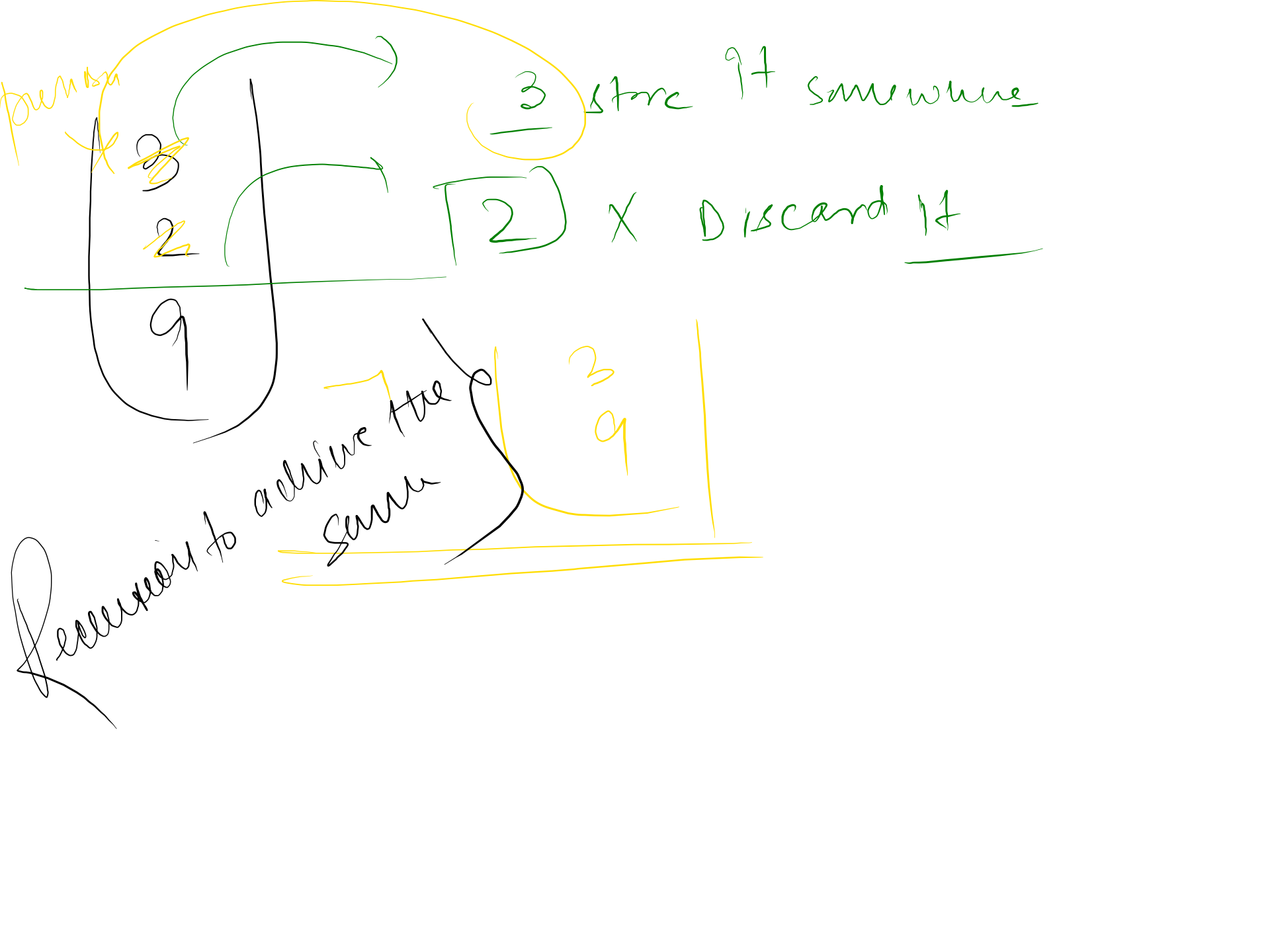
S

peek() → 9

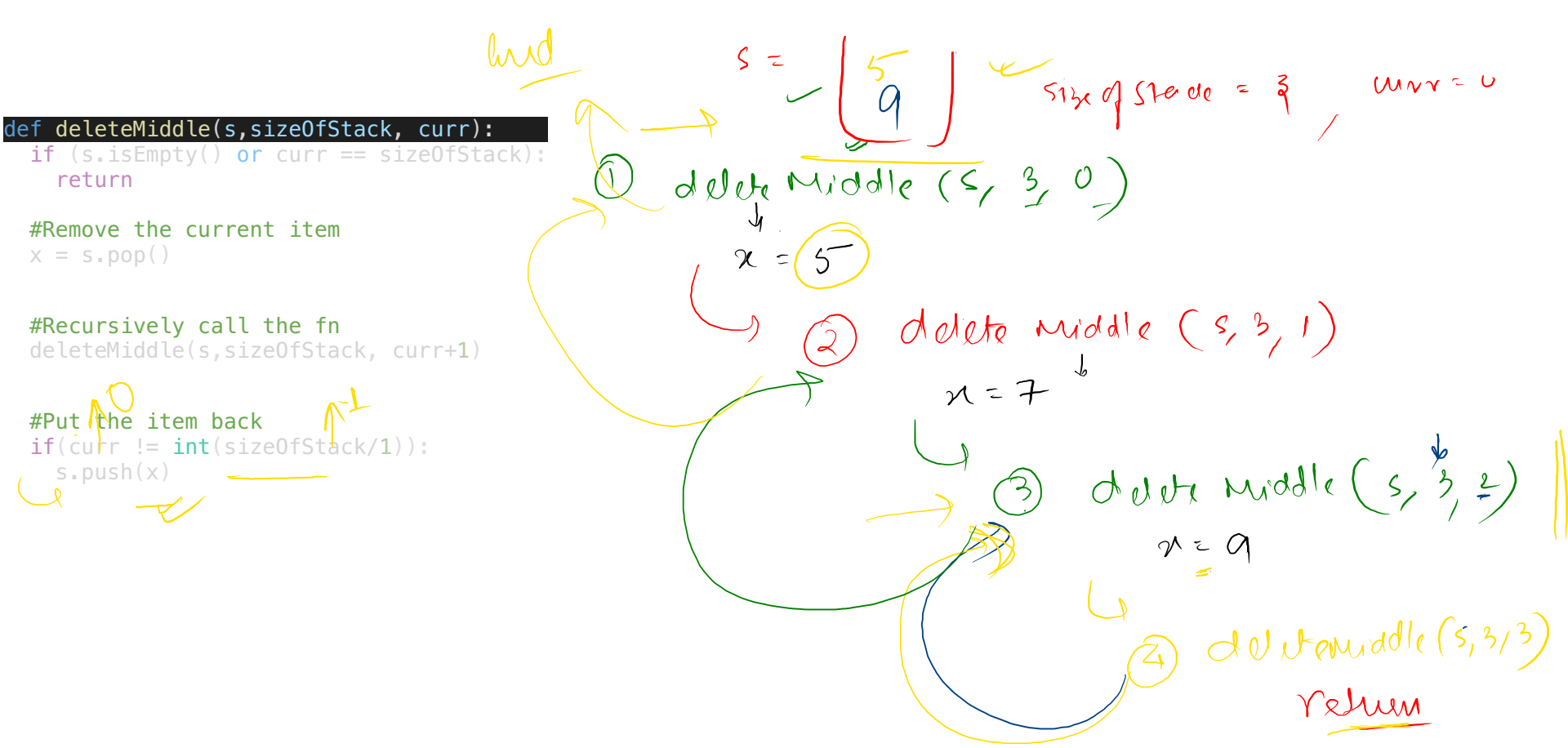
peek() → 9

peek() → 9


Recursion
Stack


3 store it somewhere
2 X Discard it
Request to acquire the semaphore

```python
def deleteMiddle(s,sizeOfStack, curr):
  if (s.isEmpty() or curr == sizeOfStack):
    return

  #Remove the current item
  x = s.pop()


  #Recursively call the fn
  deleteMiddle(s,sizeOfStack, curr+1)


  #Put the item back
  if(curr != int(sizeOfStack/1)):
    s.push(x)
```



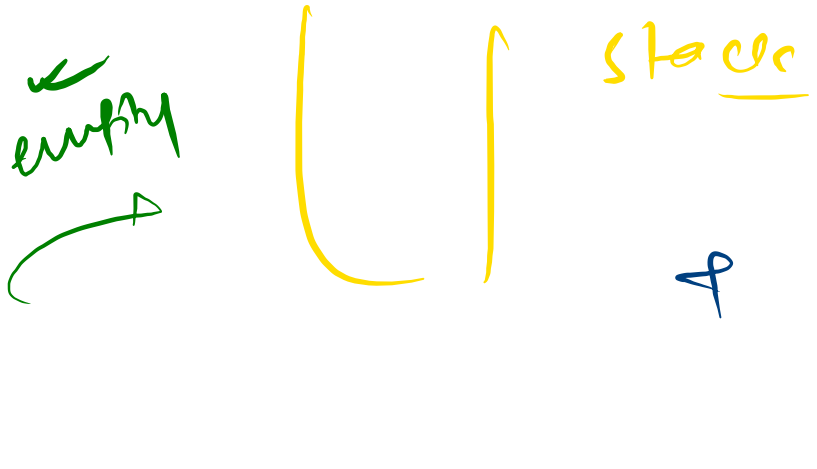
fun( ) {

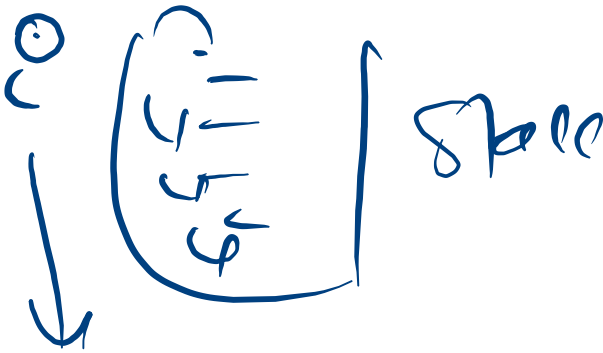
x → local variable

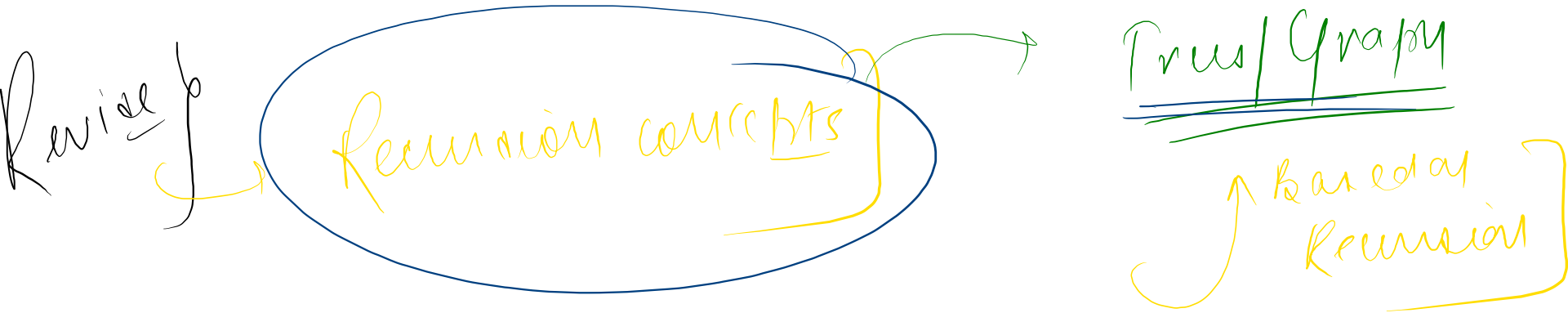

3
4

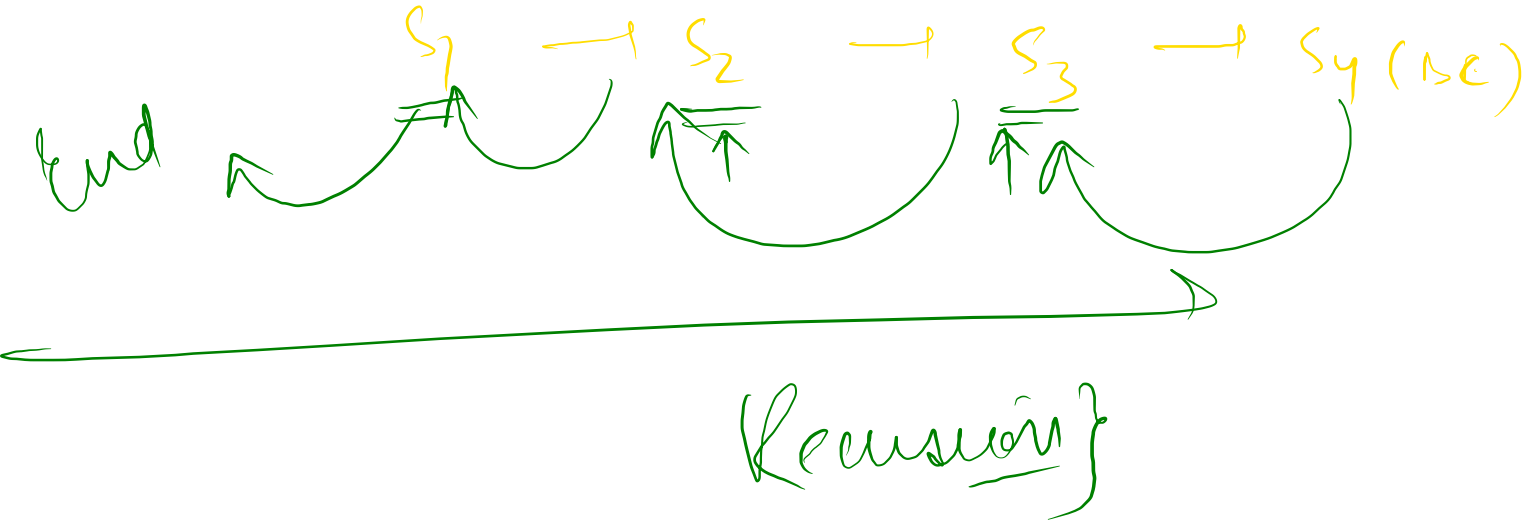
3
Middle
2
1
4

Problem

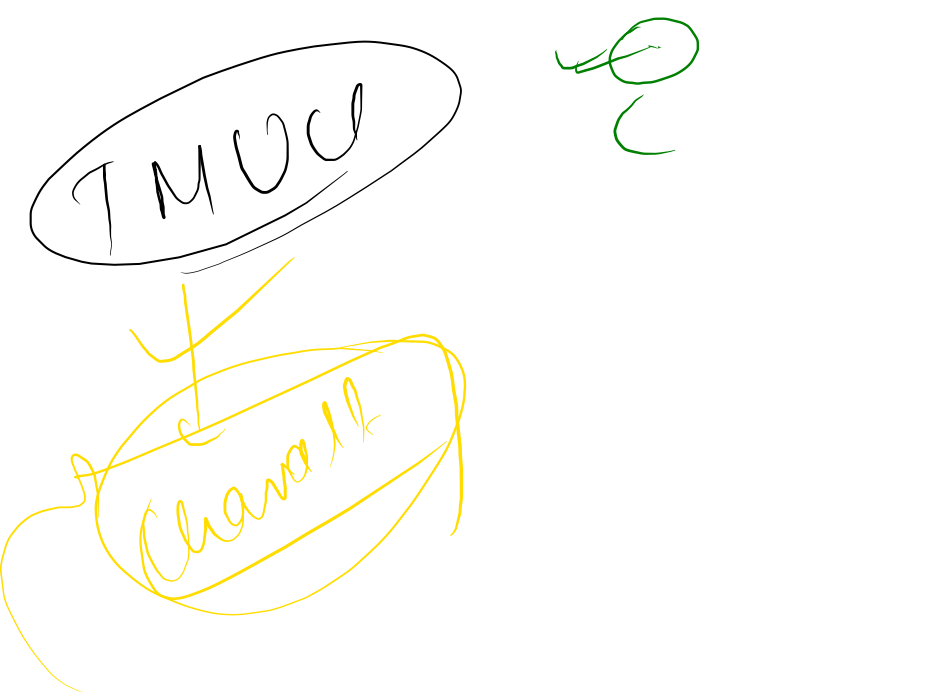





Revise
Recursion concepts
Tree/Graph
Backtracking
Recursion



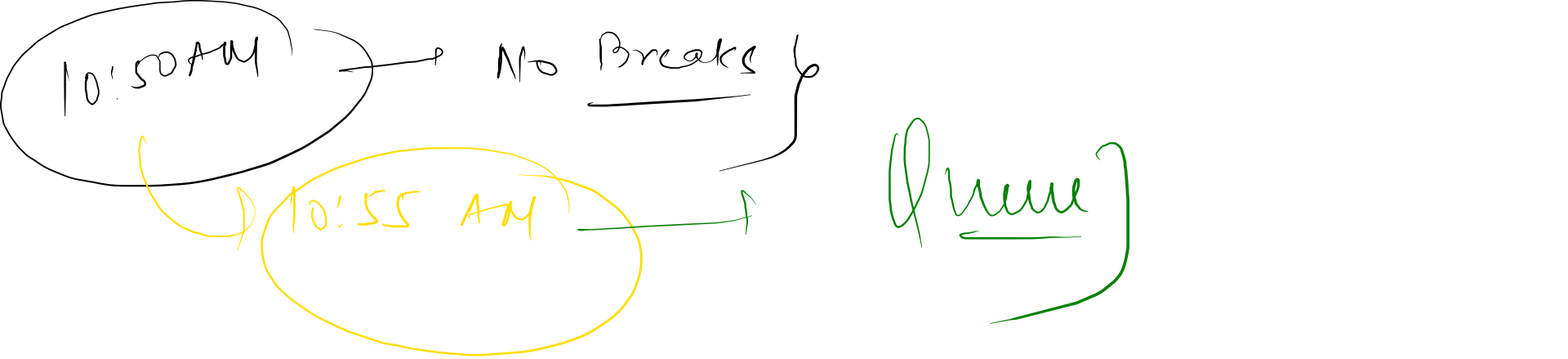
Recursion


TMUC

Stack

Array

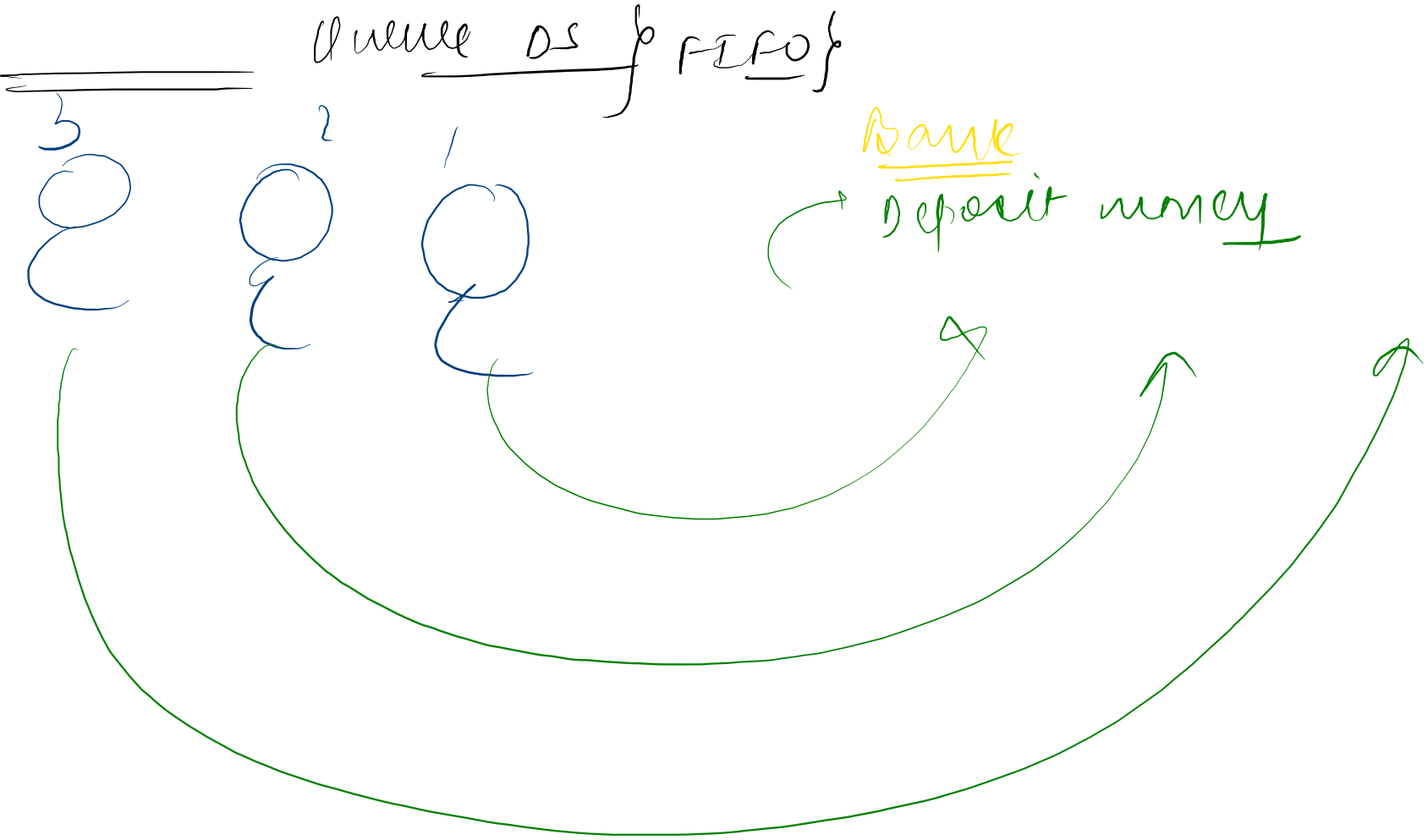
push()


10:50AM
No Breaks
10:55 AM
Queue

Queue ds

FIFO / LILO

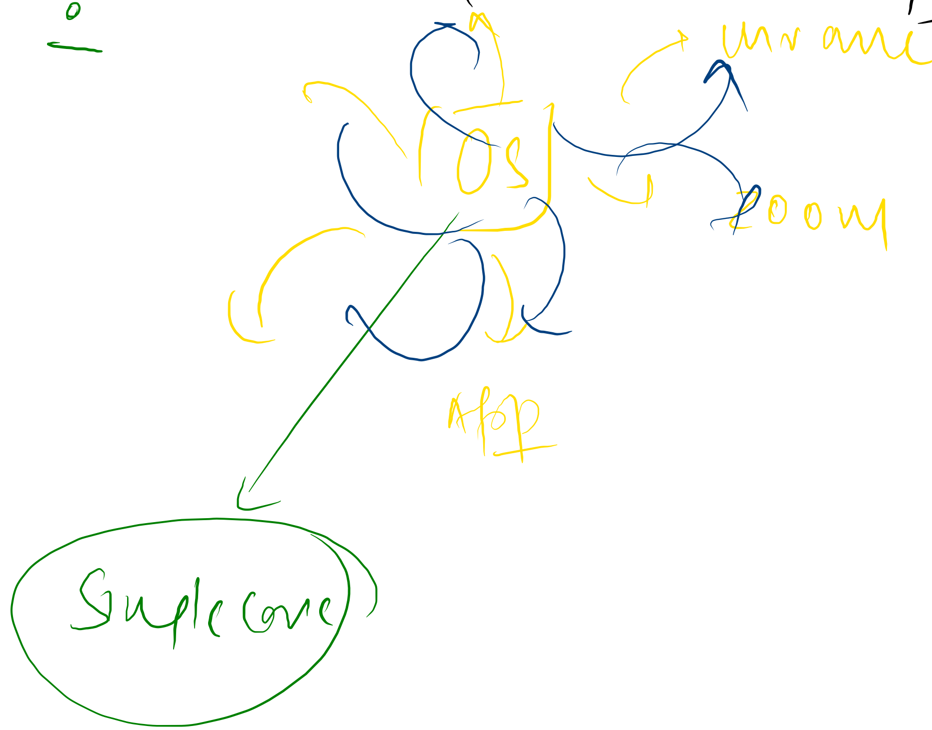
Queue

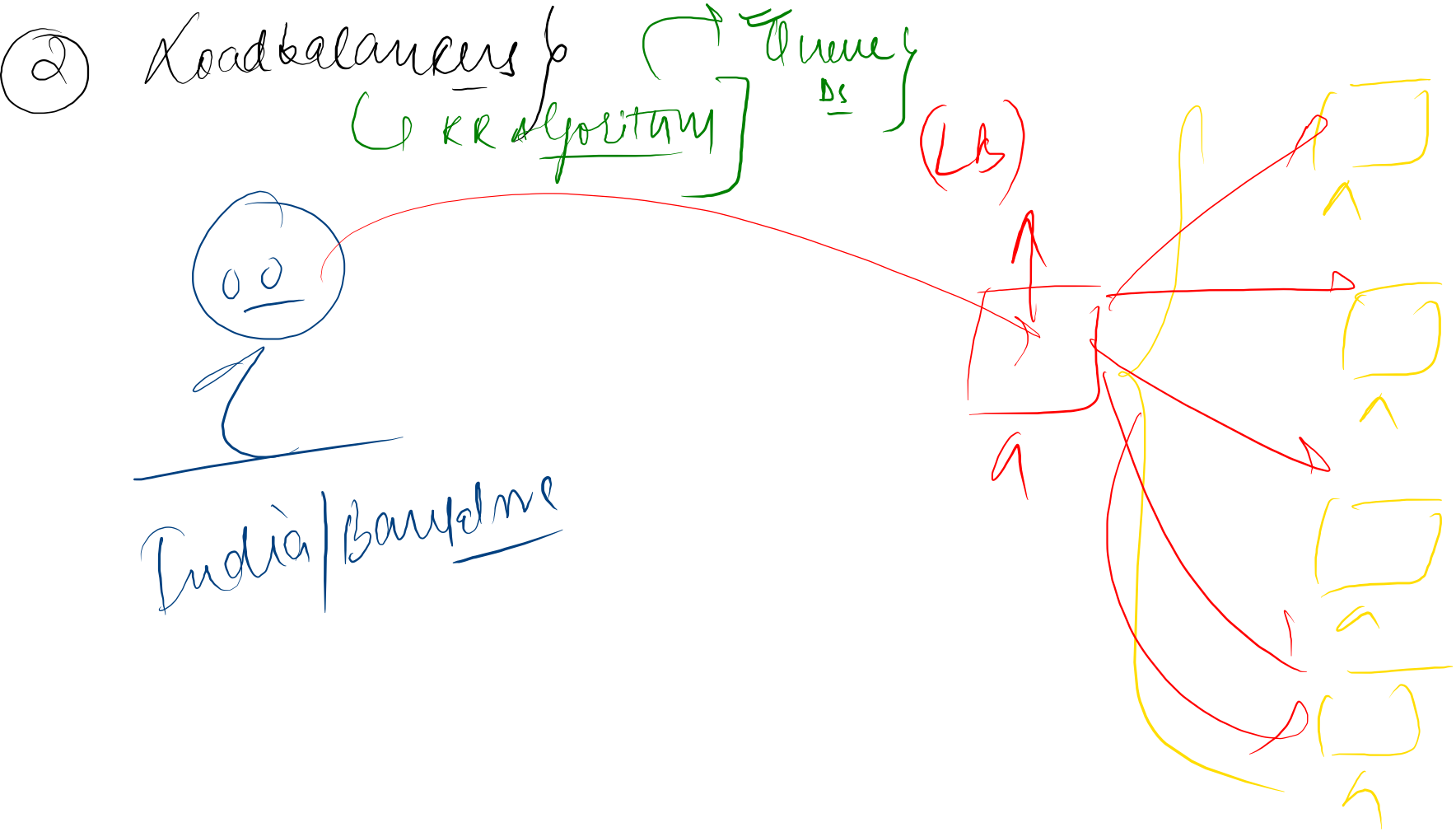

Queue DS FIFO
3
2
1
Bank
Deposit money

LILO / FIFO } → Queue]

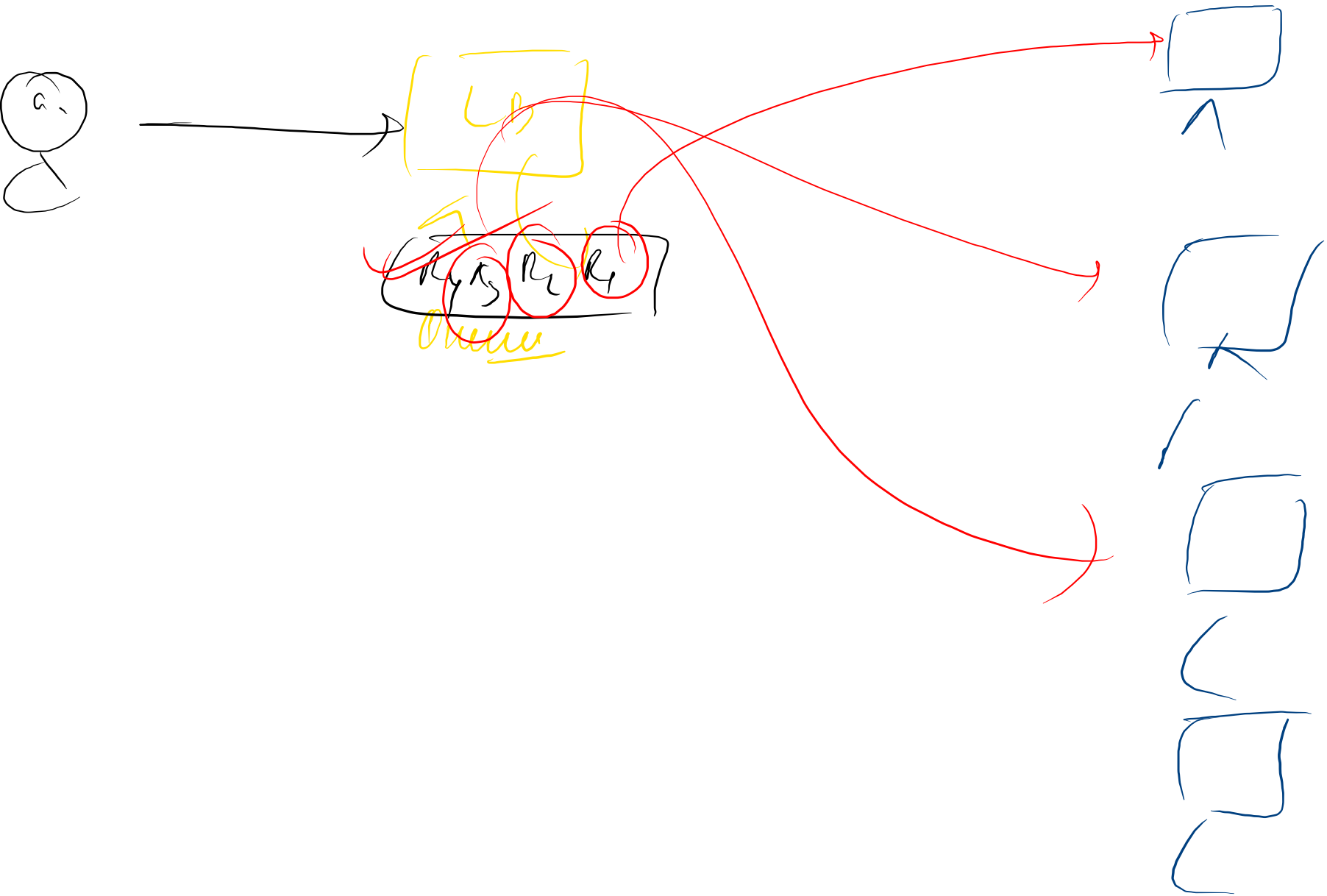
?

① OS resource scheduling]

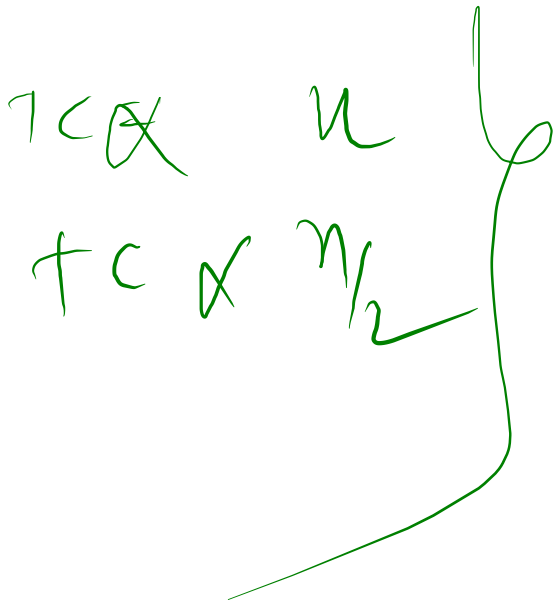



2
Loadbalancers
Client
DS
(LB)
India | Bangalore


3
Online shop
Fruits / Vegetable
Purchase items
1
2
3
Fruit1
Fruit2
Fruit3
FIFO
Queue DS




120 R/s
waiting Que
App
100 R/s
server
Kernel Queue

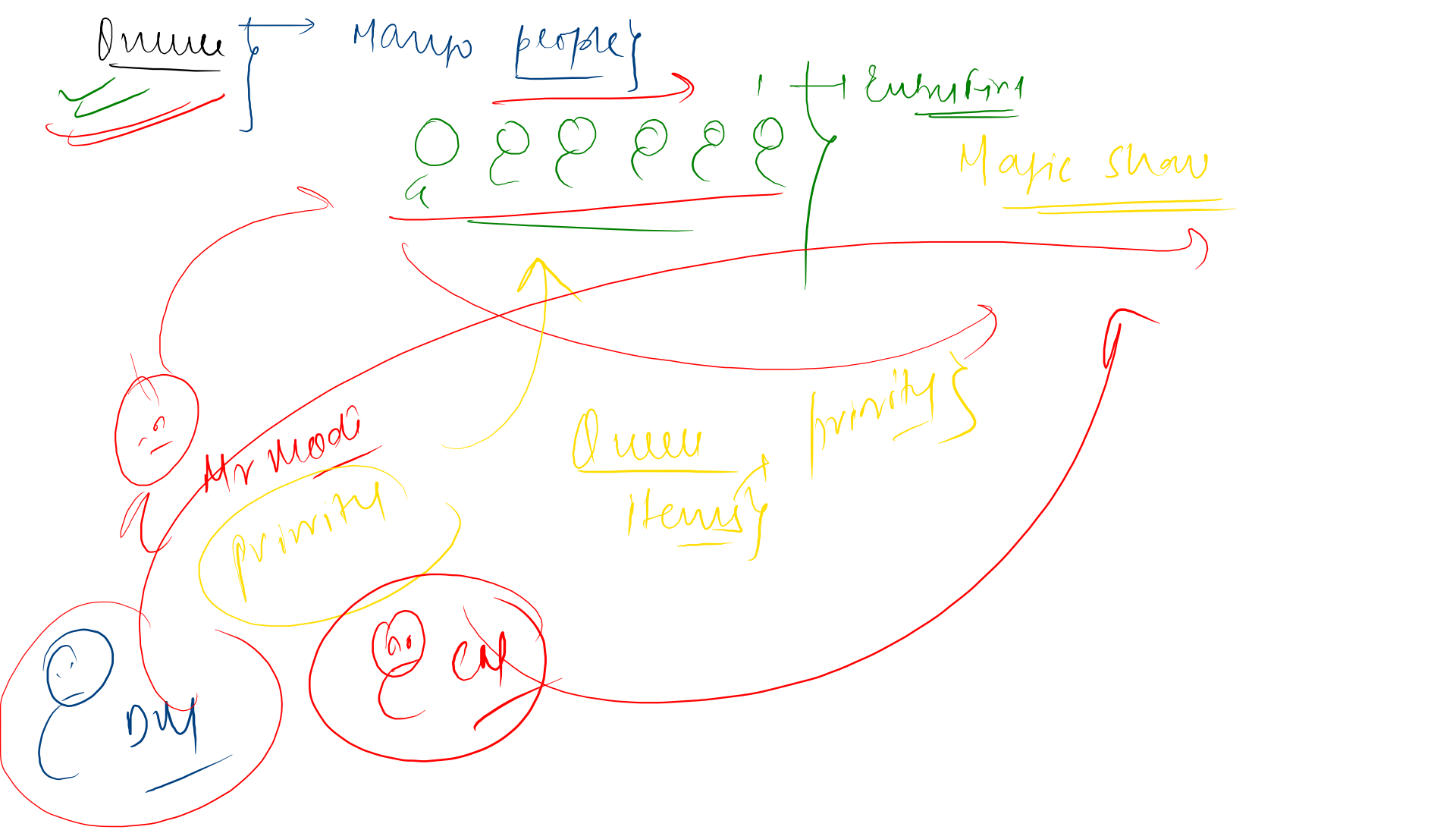

$T \propto u$
$T \propto \eta_2$

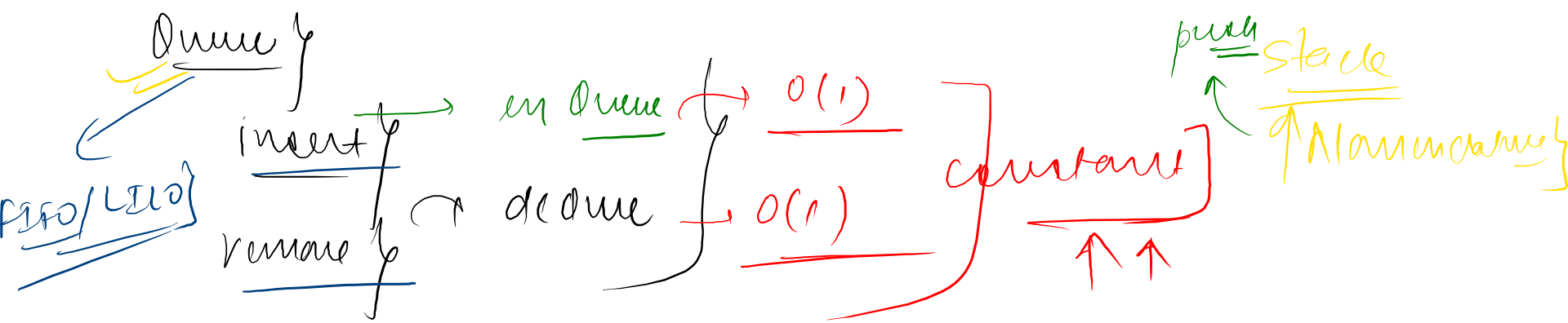
(DS) → Data

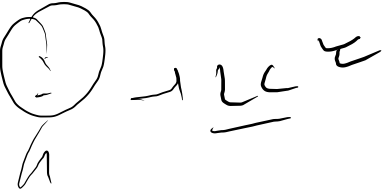
Kafka & Distributed Messaging system

Maintains the order of message

Queue data structure


Queue
Many people
Entry
Magic Show
Mr Modi
priority
Queue
items
priority
DM
CM


Queue
FIFO/LIFO
insert
remove
en Queue
O(1)
de Queue
O(1)
constant
pile
Stack
Manipulation




→ Smartly use Array → solve this

q.enque(5)

q.enque(13)

q.enque(73)

q.enque(33)

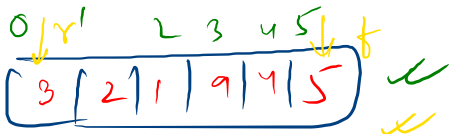
q.deque() → 5

q.deque() → 13

q.enque(23)

q.enque(72)

q.enque(23)



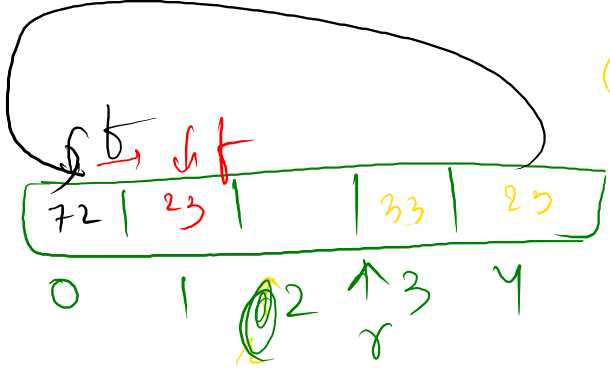
Condition

$(f+1)\%size == r$

→ $(5+1)\%6 == 0$

→ $0 == 0$

q.deque() → 73



$(f+1)\%5$

$= (4+1)\%5 = 0$

$r = 2$

front = -1 / 0 / 1 / 2 / 3 / 3 / 3 / 4 / $(4+1)\%5 = 0$ / 1

rear = -1 / 0 / 0 / 0 / 1 / 2 / 3

No element

if it's full ?

$(f+1)\%size == r\%size$

$= (1+1)\%5 = 2\%5 = 2 == 2$

q.enque(51)

Queue

(f) → enque

(r) + deque

② Empty → f == r == -1

③ Full → (f+1) % size == r % size

→ facts

r = (r+1) % size

↳ r < size

(so that index is always in correct range)

front = 3

capacity = 5

for f%c = 3

3%5 = 3

↳ 3

f = 4

c = 5

f = f%c

4 = 4%5 = 4

f = 5

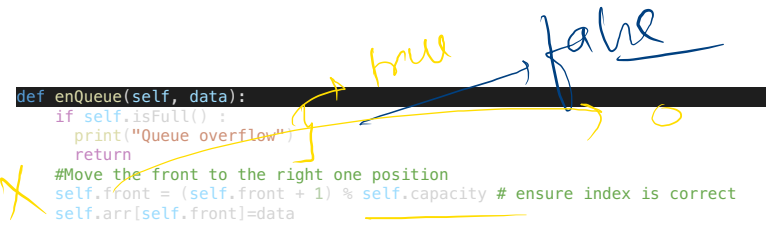
c = 5

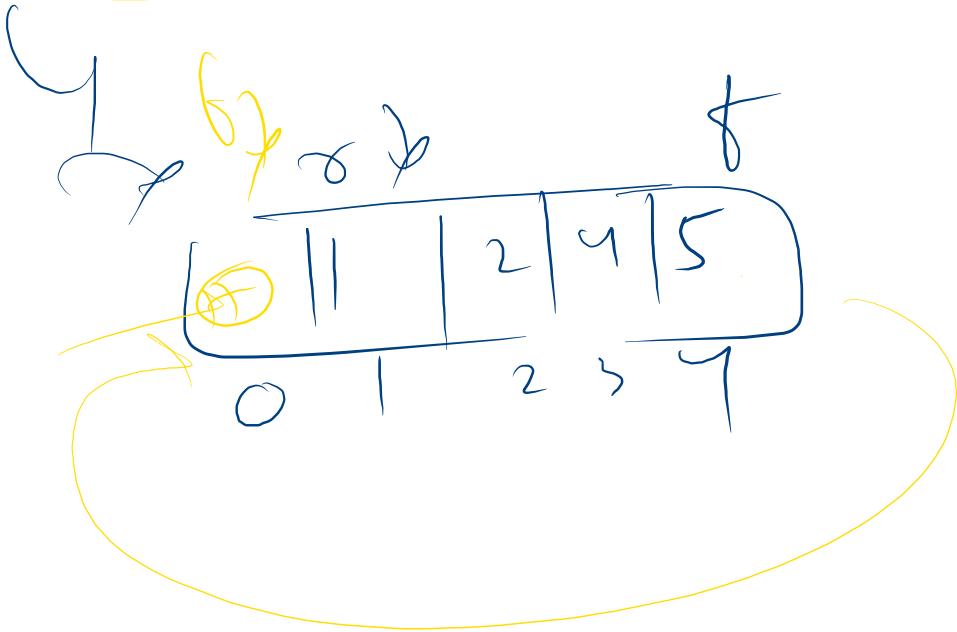
f%c = 5%5 = 0

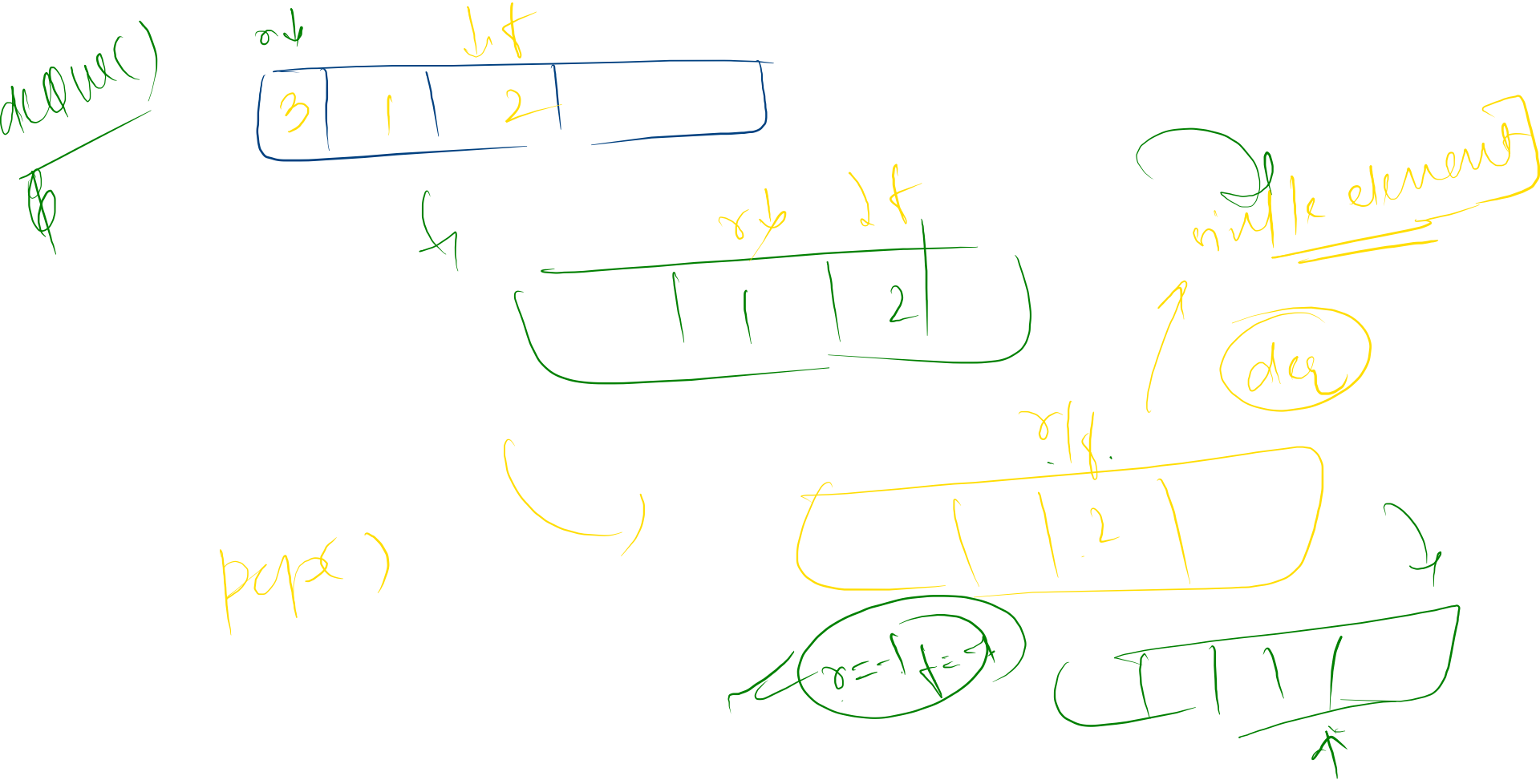
0th index

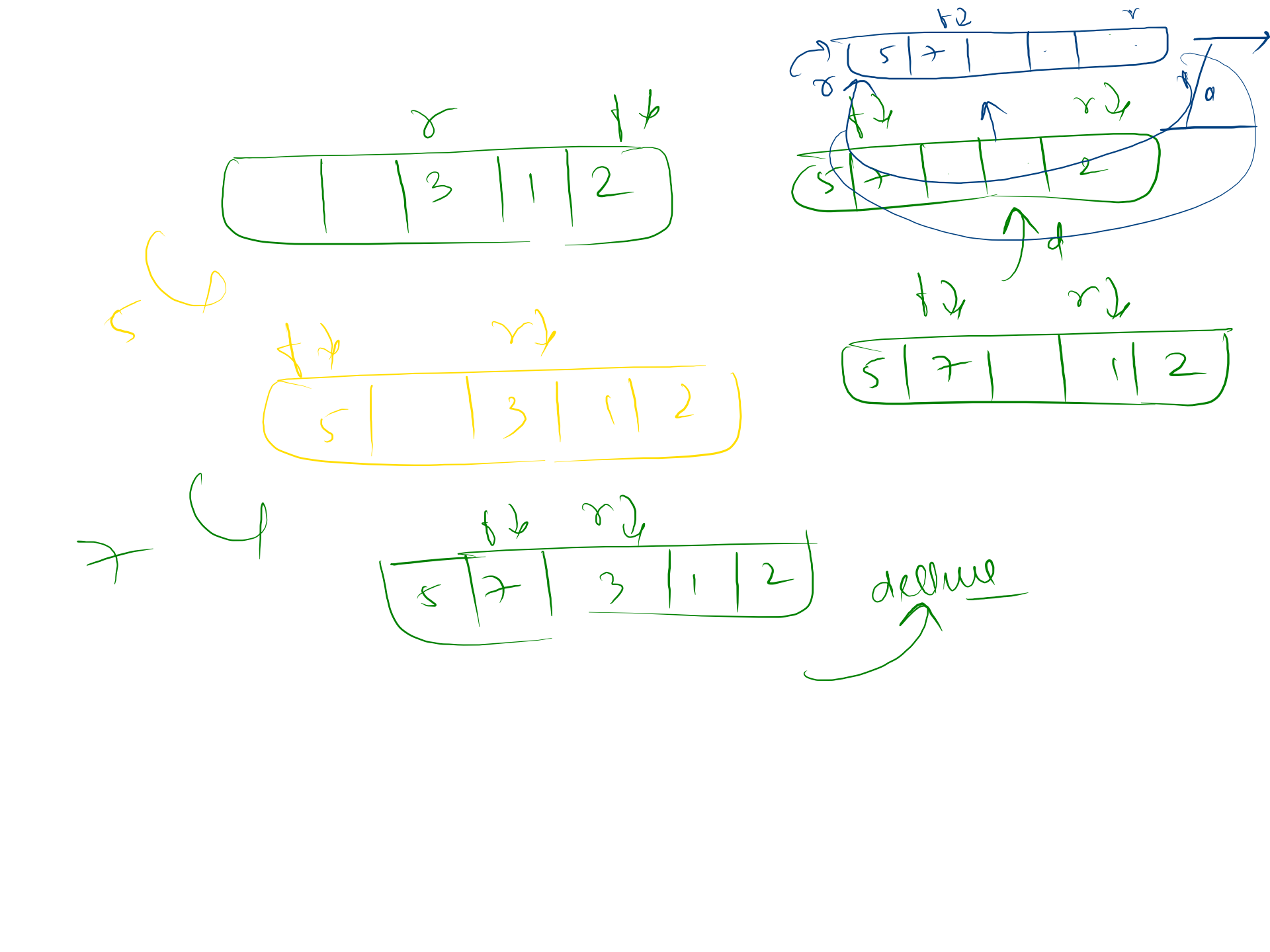
More than 1st index

```
def enQueue(self, data):
    if self.isFull() :
      print("Queue overflow")
      return
    #Move the front to the right one position
    self.front = (self.front + 1) % self.capacity # ensure index is correct
    self.arr[self.front]=data
```



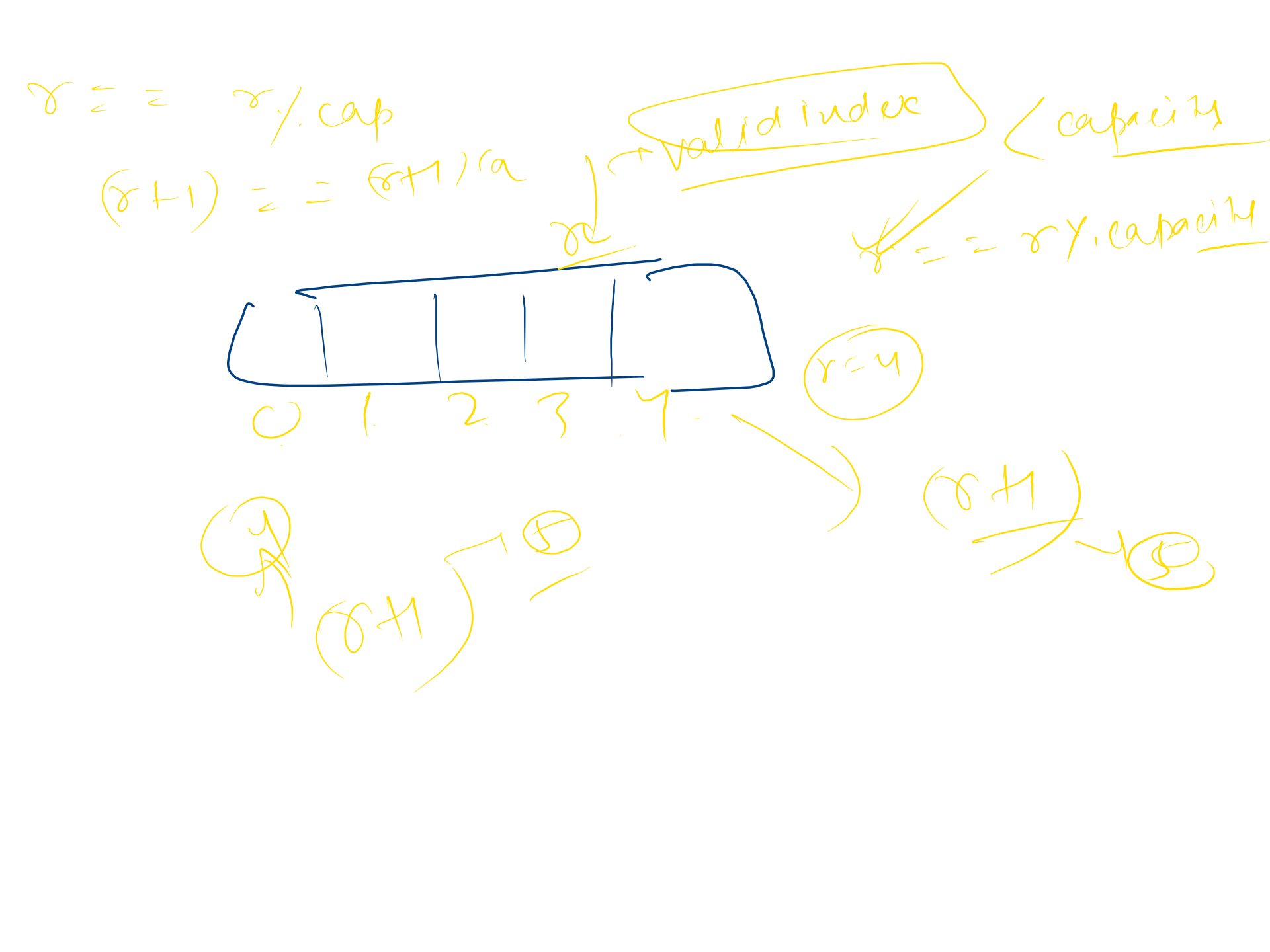



deque()
3
1
2
middle element
deq
pop()
x==-1 || x==7


5
7
3
1
2
delete

$r = 2\%$ capacity

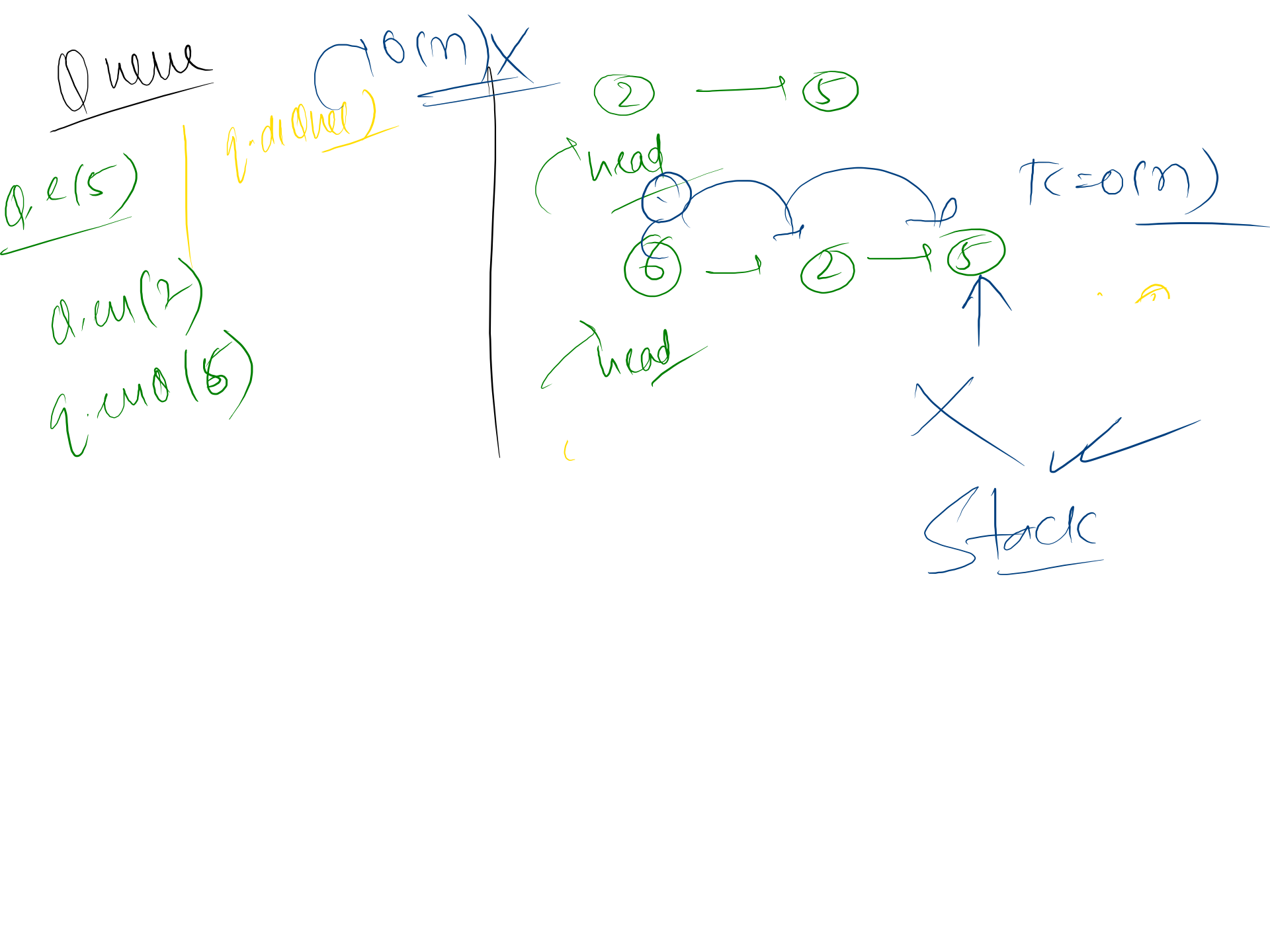
when $r <$ capacity


r == r % cap
(r+1) == (r+1) % a
valid index
capacity
y == ry.capacity
0 1 2 3 4
r = 4
(r+1)
(r+1)

```
class Queue :
  def __init__(self, capacity):
    self.arr = [None]*capacity
    self.capacity = capacity
    self.front = -1
    self.rear = -1 #in the begining both front and rear should be -1
    self.size =0

  def isEmpty(self):
    return self.front == -1


  def isFull(self):
    return (self.front +1)%self.capacity == self.rear


  def enQueue(self, data):
    if self.isFull() :
      print("Queue overflow")
      return
    #Move the front to the right one position
    self.front = (self.front + 1) % self.capacity # ensure index is correct
    self.arr[self.front]=data

    if self.rear == -1:  #when the first element is inserted
      self.rear =0
    self.size +=1


  def deQueue(self):
    if self.isEmpty():
      print("Queue underflow")
      return
    data = self.arr[self.rear]
    #Check if this was the only element
    if self.rear == self.front:
      self.rear = -1
      self.front = -1
      #Because it was the only element, and after removal it's empty
    else :
      self.rear = (self.rear+1)%self.capacity #ensure we are in correct index range
    self.size -=1
    return data


  def size(self):
    return self.size


  def traverse(self):
    for i in range(self.size):
      print(self.arr[self.rear+i]%self.capacity)
```

Implement Queue with LL :-


3D
Treat Me :)


Queue
O(n)
q.dequeue()
q.e(5)
q.en(2)
q.en(6)
2
5
head
6
2
5
head
TC=O(n)
Stack

Queue

tail → enqueue
head → dequeue
→ O(1)

Q.enq(5)

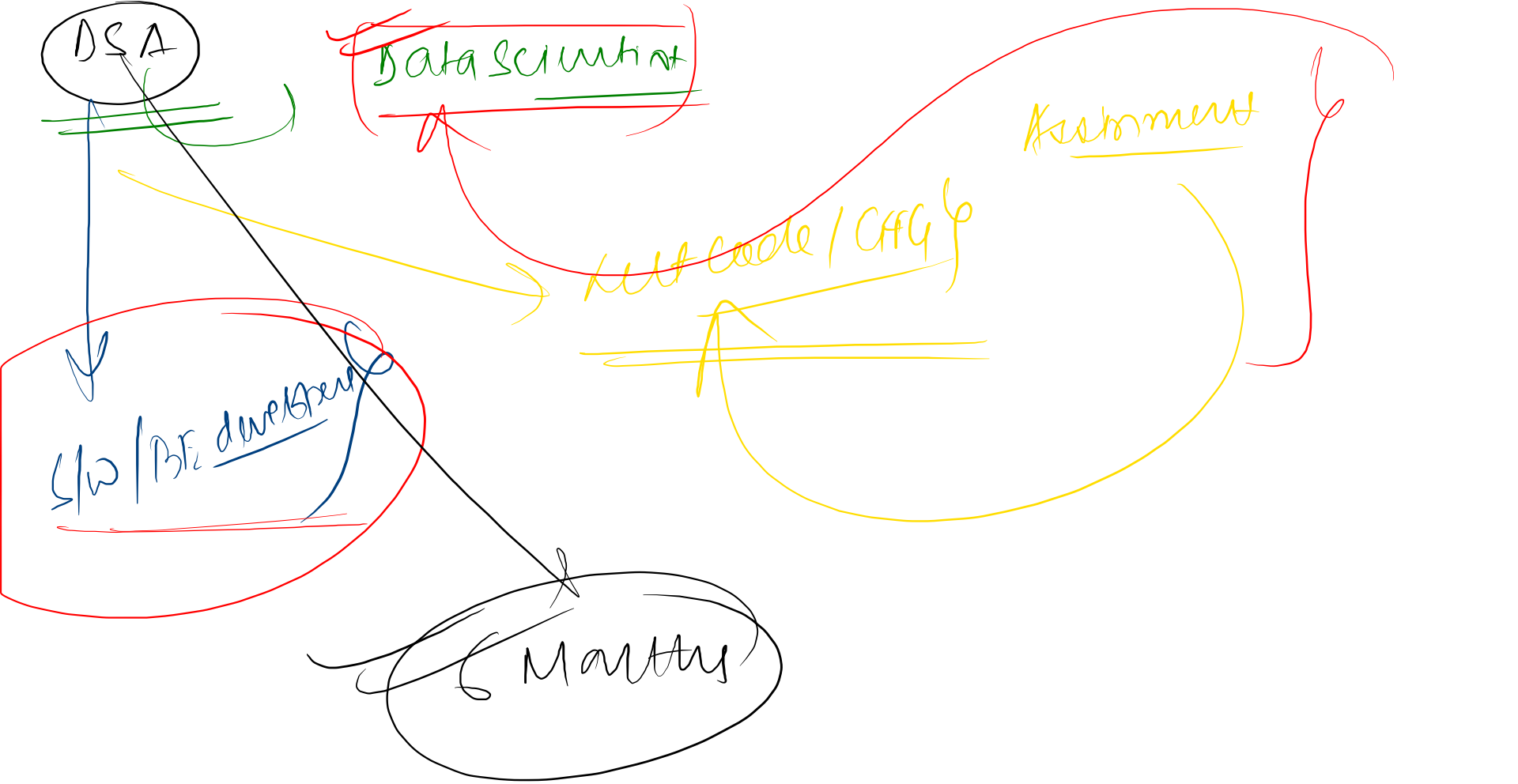
Q.enq(7)

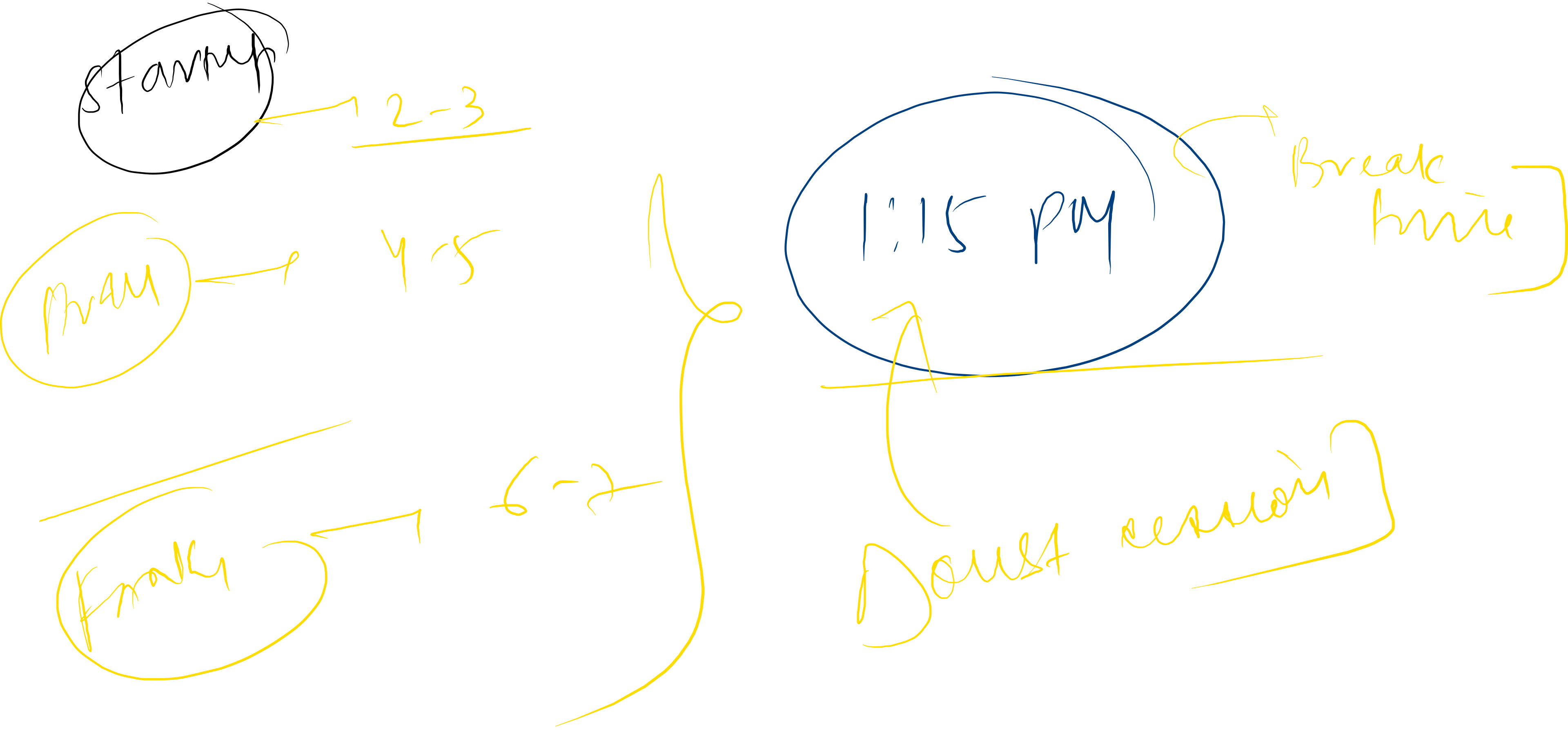
Q.enq(9)

→ 5

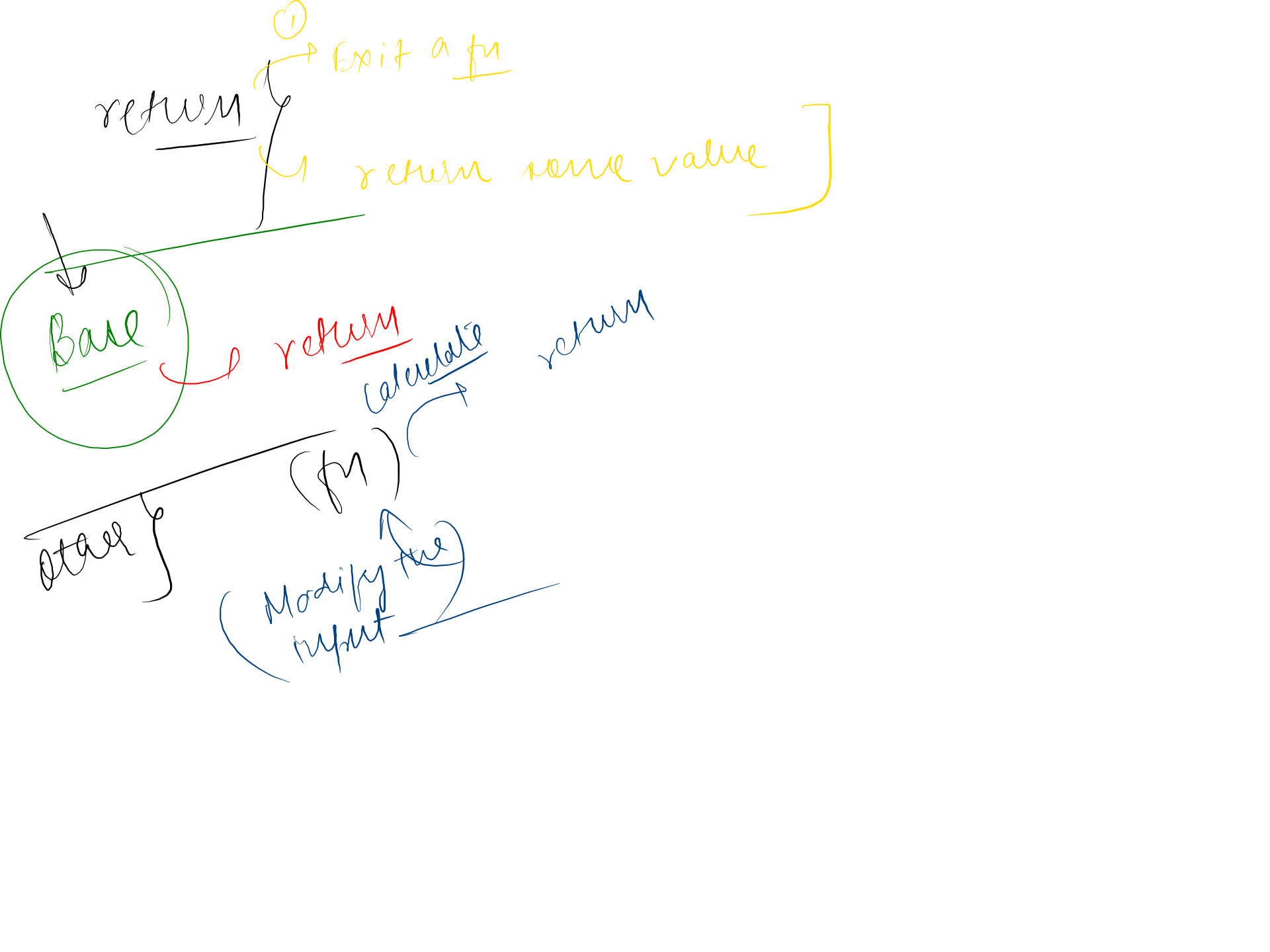
Q.deque()

Q.deque()


DSA
Data Scientist
Assignment
Test code / CHG
S/w / BE developer
Maths


2-3
4-5
6-7
1:15 PM
Break time
Doubt session

```
def revv_rec(arr,si,ei):              #Why this is not working
    if si>ei:
        return
    else:
        return revv_rec(arr,si+1,ei-1)
        arr[si],arr[ei]=arr[ei],arr[si]
```

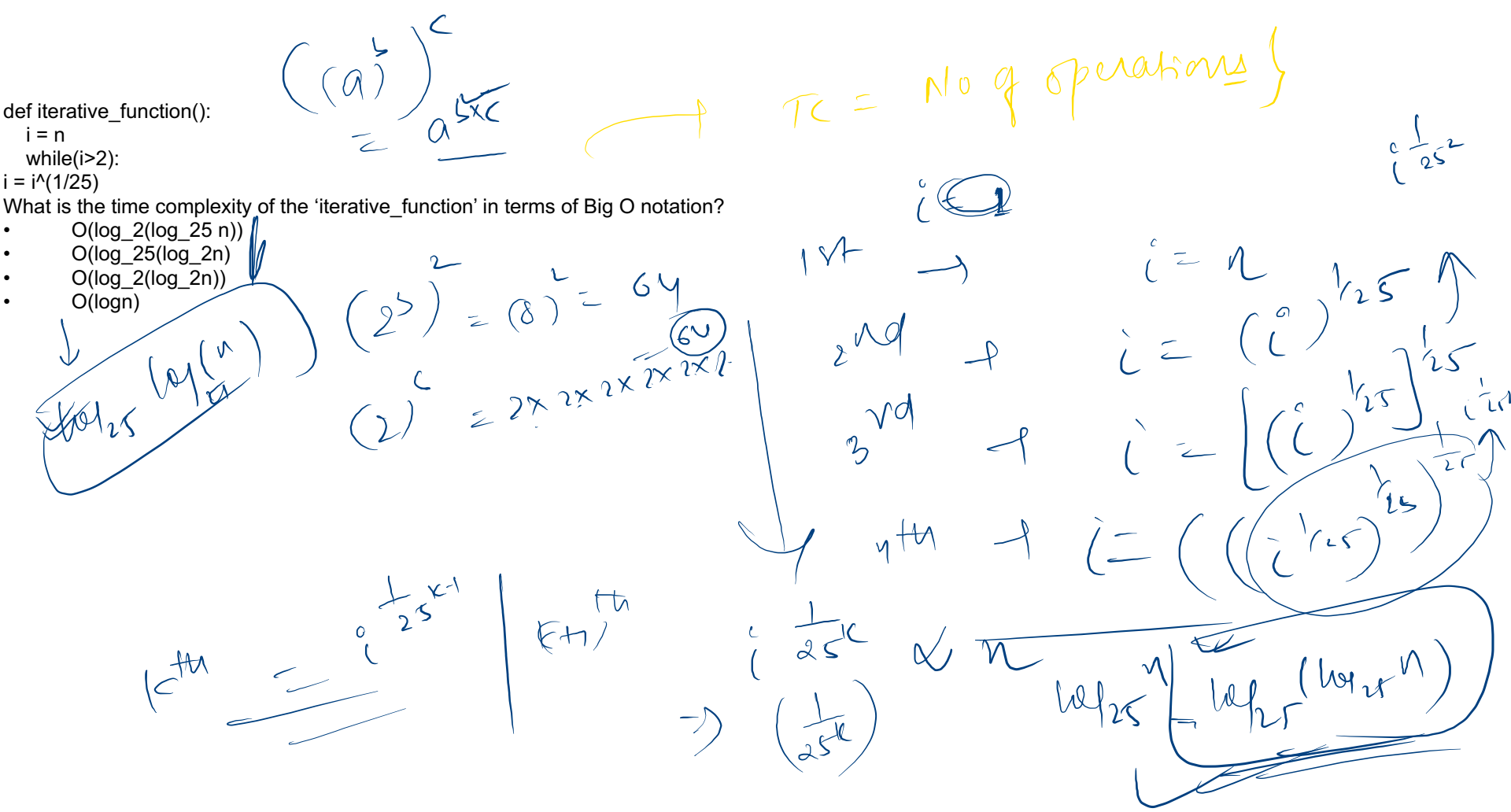

1
Exit a fn
return
return some value
Base
return
calculate
return
(fn)
other
Modify the input

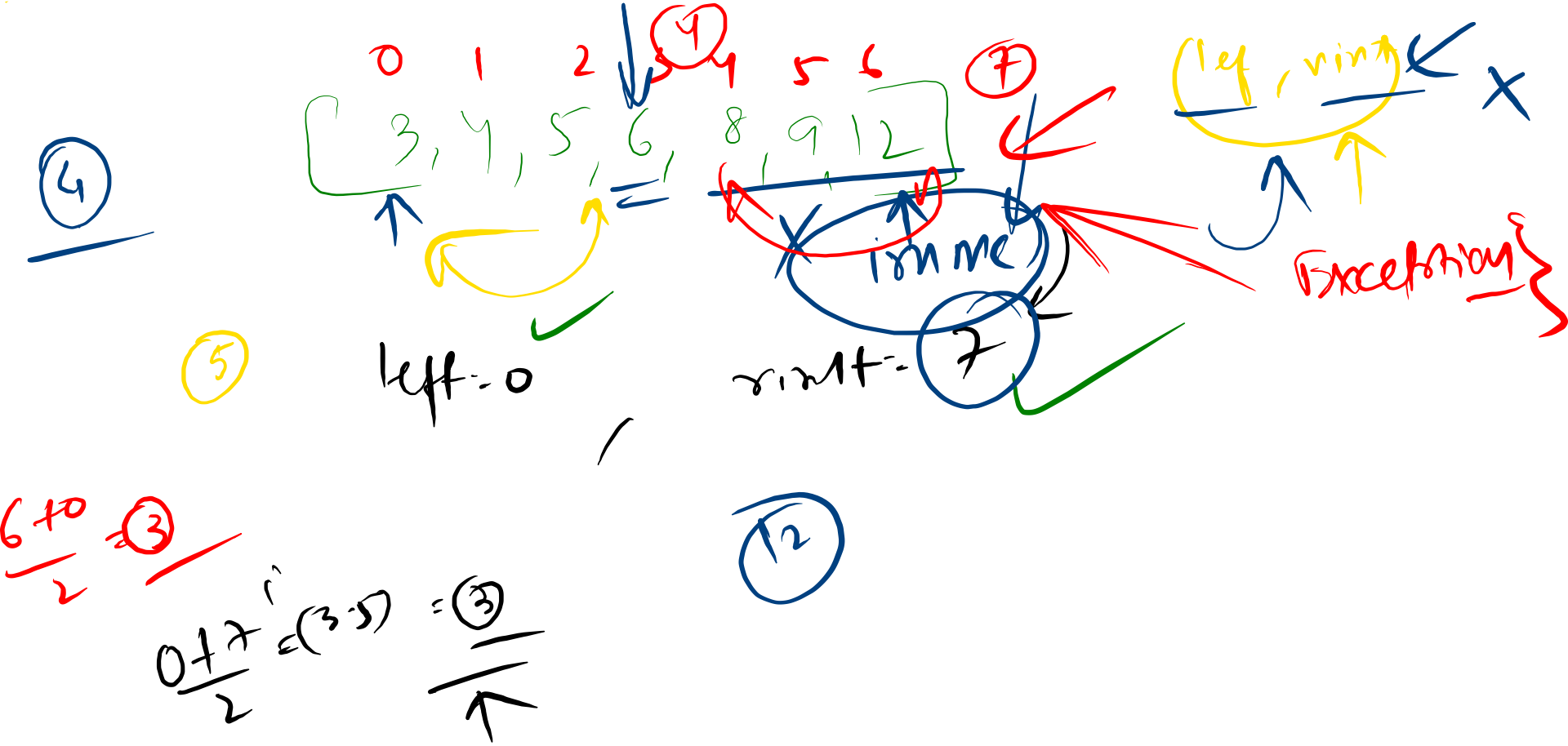
```
def findrotate_ii(arr):
    start=0
    end=len(arr)-1

    if arr[start]<arr[end]:
        return 'arrey is not rotated'
    while start<=end:
        mid=(start+(end-start)//2)

        if arr[mid]<arr[mid+1] and arr[mid]<arr[mid-1]:
            return mid
        elif arr[mid]<arr[end]:
            #start=mid+1
            end=mid-1
        else:
            arr[mid]>arr[start]
            #end=mid-1
            start=mid+1
    return
```

```
def iterative_function():
    i = n
    while(i>2):
i = i^(1/25)
```

What is the time complexity of the 'iterative_function' in terms of Big O notation?

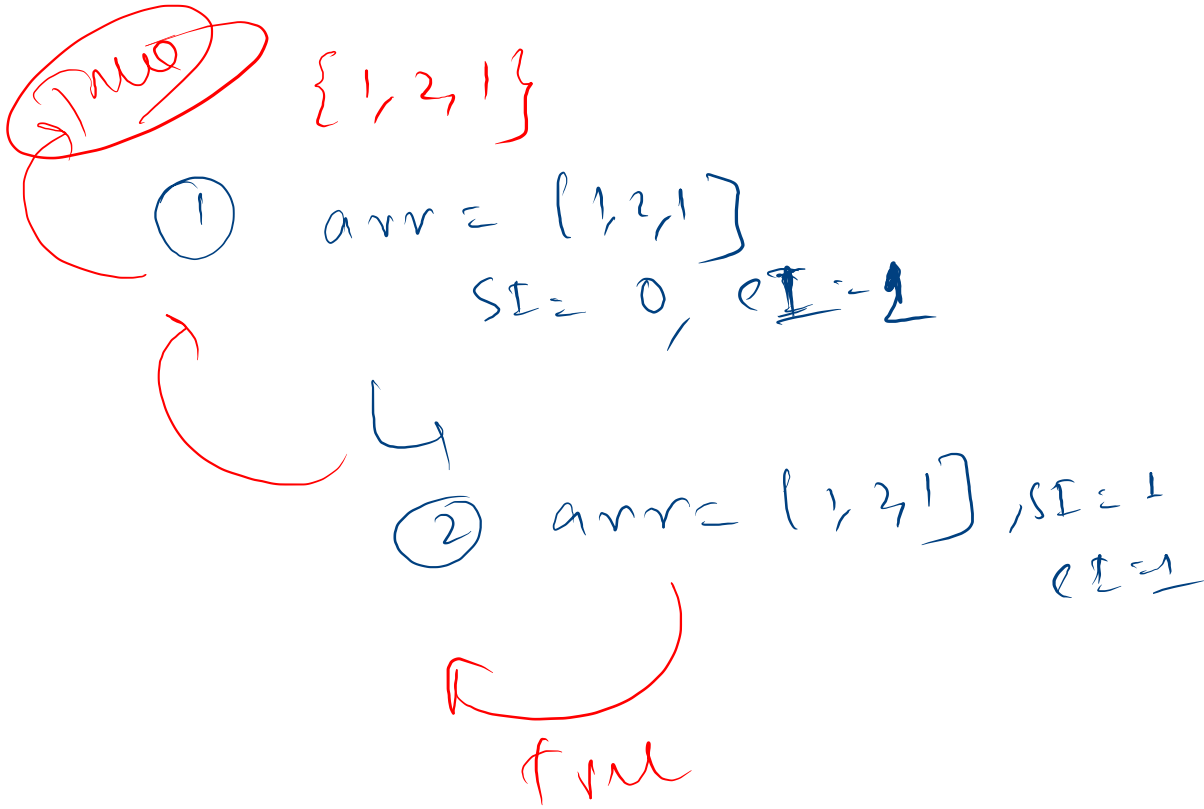
- O(log_2(log_25 n))
- O(log_25(log_2n)
- O(log_2(log_2n))
- O(logn)




0 1 2 3 4 5 6 7
3, 4, 5, 6, 8, 9, 12
left, right
X
4
5
left = 0
right = 7
mid
Exception
6+0/2 = 3
0+7/2 = (3.5) = 3
12

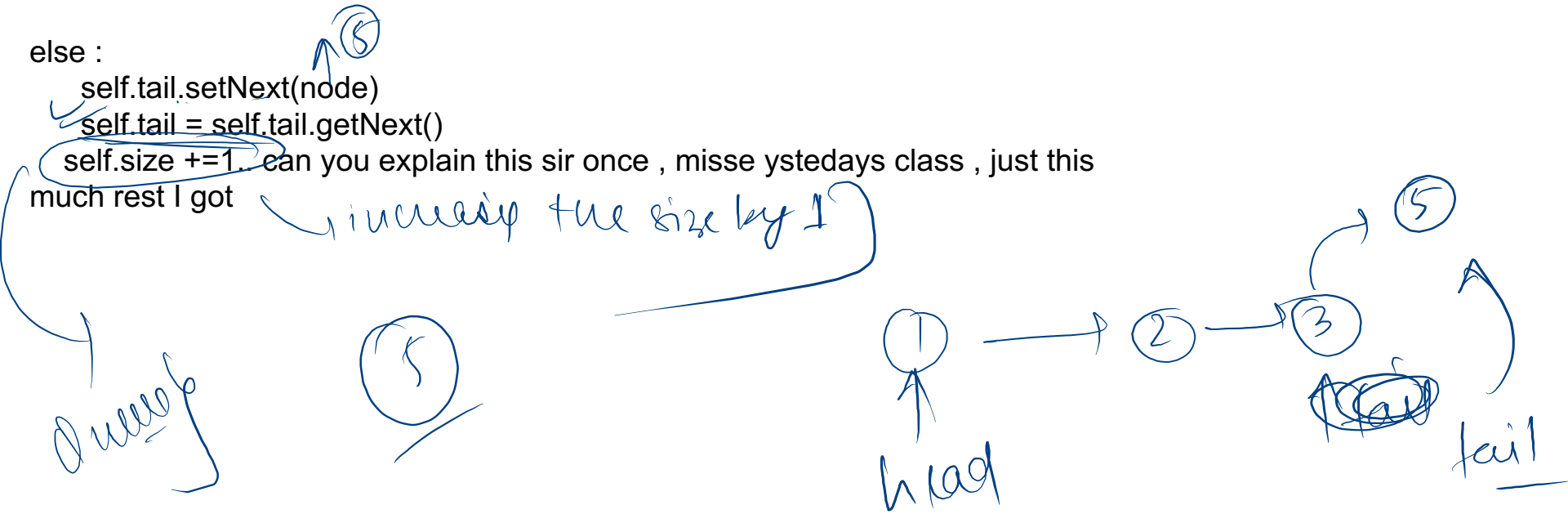
fn(left, right)

```python
def isPalindrome_rec(arr,sI,eI):
    # Base Case
    if sI>eI:
        return False
    if sI==eI:
        return True
    # Logic
    if arr[sI]!= arr[eI]:
        return False
    # Recursive Call
    return
isPalindrome_rec(arr,sI+1,eI-1)
```

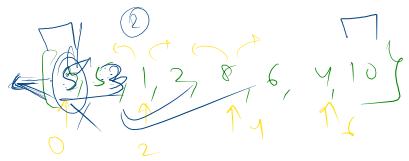


```
else :
    self.tail.setNext(node)
    self.tail = self.tail.getNext()
    self.size +=1
```

can you explain this sir once , misse ystedays class , just this much rest I got



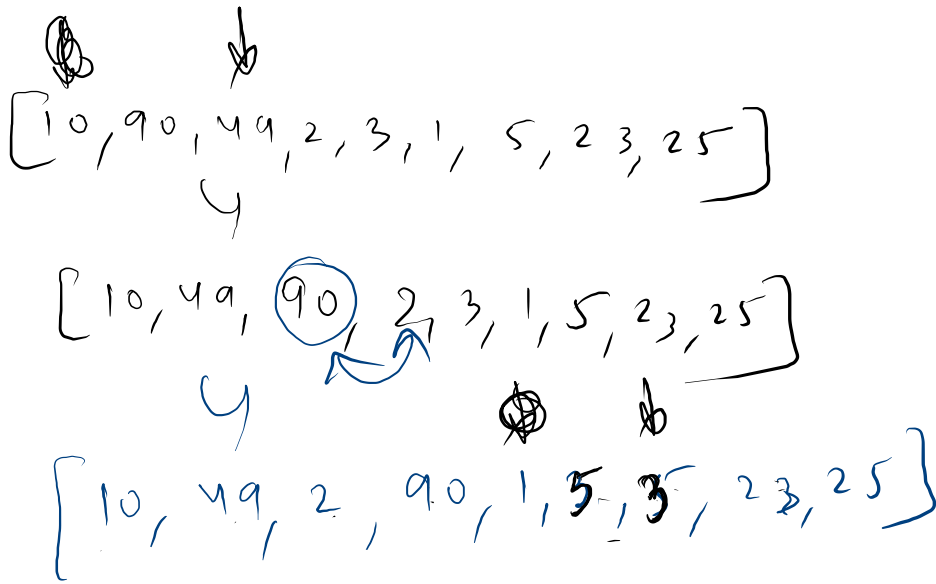
```
arr=[3,5,1,2,8,6,4,10]

def sortinwave(arr):
    for i in range (0,len(arr)-1,2):          #Rather than mentioning can we
start range from 2
        if i>0 and arr[i]>arr[i-1]:                #even if do not add condition i>0 and
i<len(arr)-1 I am not getting
            arr[i],arr[i-1]=arr[i-1],arr[i]         #out of index error why? see below
code and output
        elif i<len(arr)-1 and arr[i]>arr[i+1]:       #code is not working see 15 and
26
            arr[i],arr[i+1]=arr[i+1],arr[i]
    return arrr
```

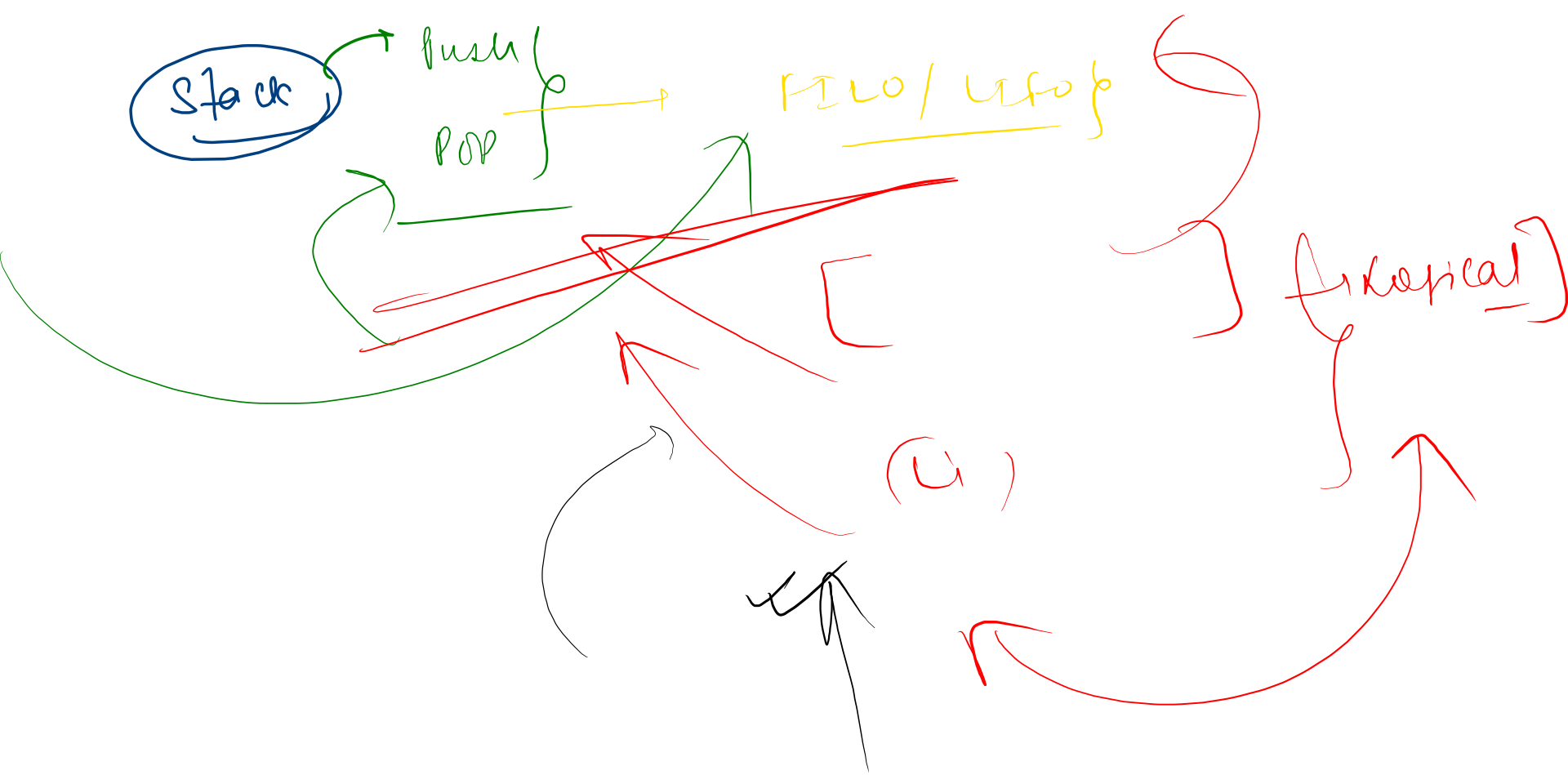


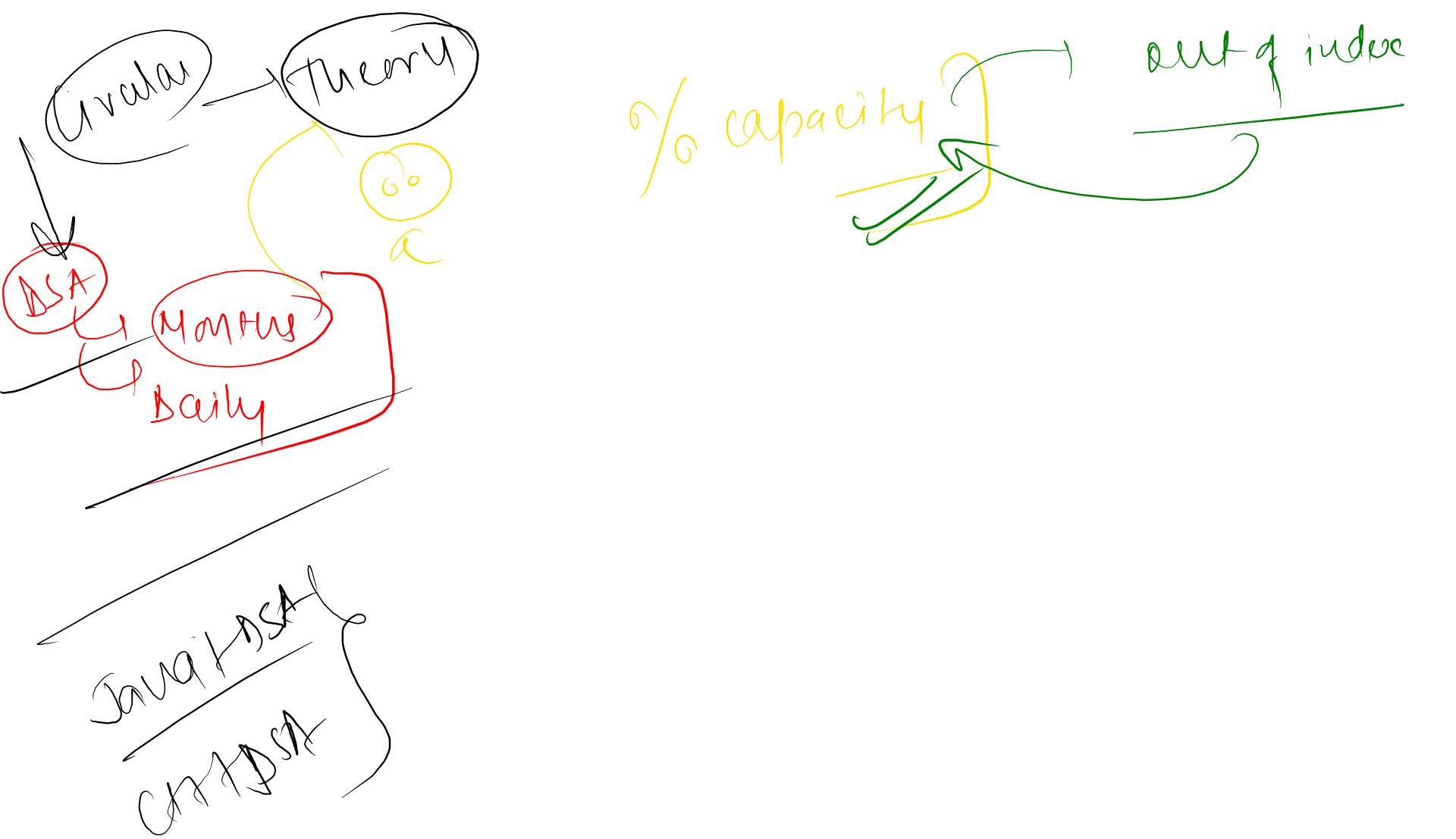
~~Node~~ node = Node(5)

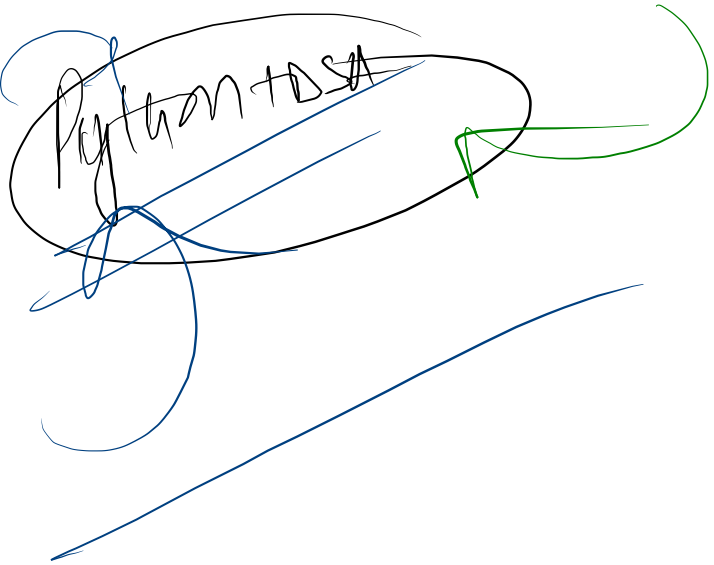
node.setData(6)

```
def  sortInWave(arr):

    for i in range(0,len(arr),2):


        #Comparing with left eleement
        if i>0 and arr[i] > arr[i-1]:
            arr[i]  ,arr[i-1]=arr[i-1],arr[i]


        #Compare with the right element
        if i<=len(arr)-1 and arr[i] > arr[i+1]:
            arr[i]  ,arr[i+1]=arr[i+1],arr[i]


arr = [10,90,49,2,3,1,5,23,25]
```



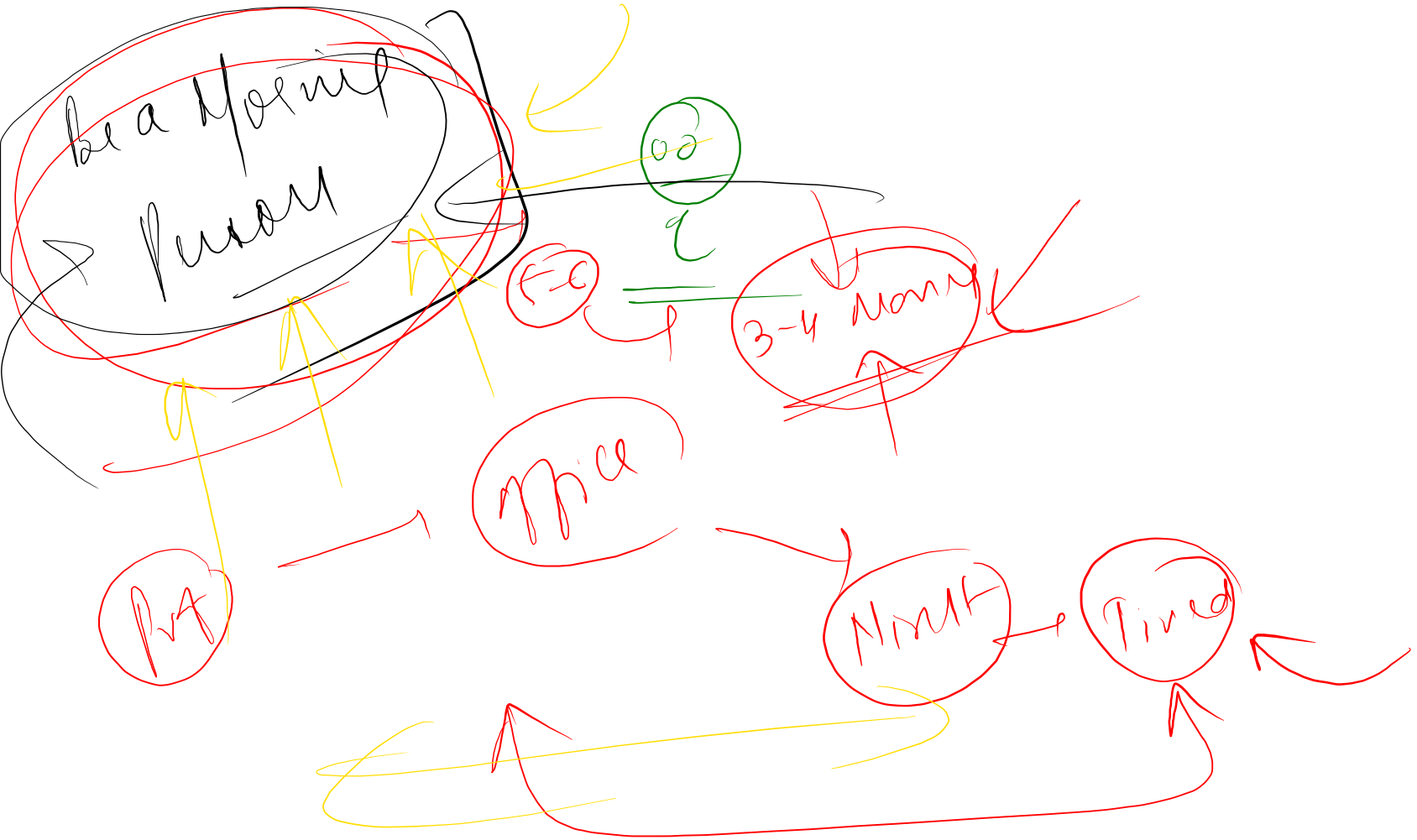

Stack
Push
POP
FILO / LIFO
Logical

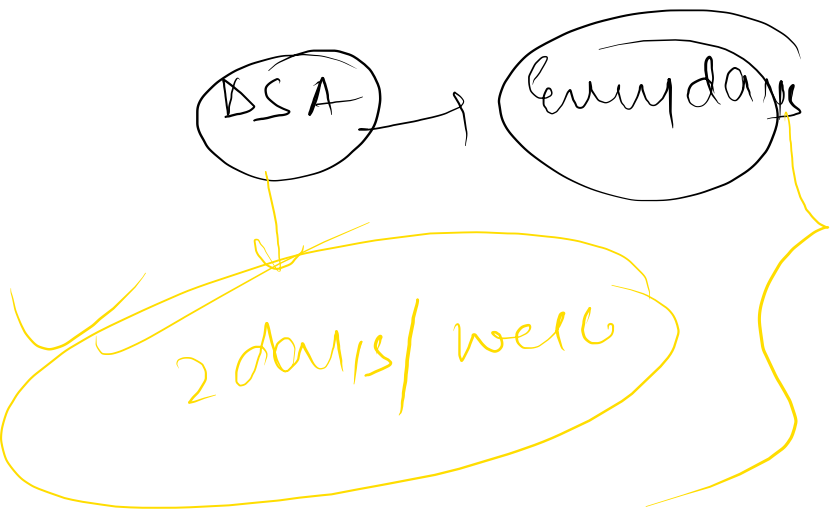

Uvalu
Theory
DSA
Months
Daily
Java + DSA
C++ + DSA
% capacity
out of index



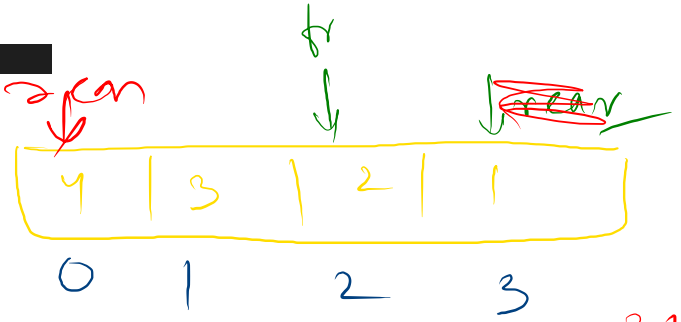
Big Product Based


Stack
Queue
push / pop
enqueue
dequeue




DSA
Everyday
2 days/ week

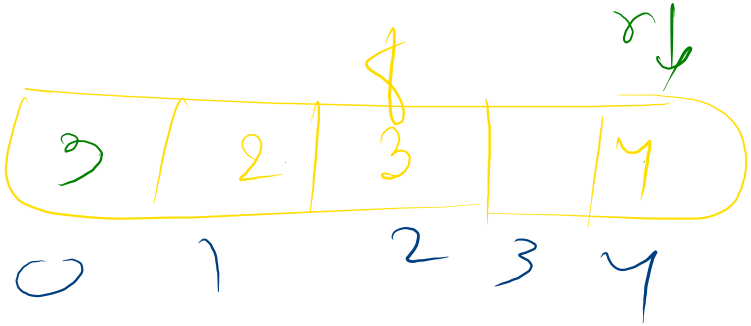
```
def deQueue(self):
    if self.isEmpty():
        print("Queue underflow")
        return
    data = self.arr[self.rear]
    #Check if this was the only element
    if self.rear == self.front:
        self.rear = -1
        self.front = -1
        #Because it was the only element, and after removal it's empty
    else :
        self.rear = (self.rear+1)%self.capacity #ensure we are in correct
index range
    self.size -=1
    return data
```



3 + 1
= 4
Beyond valid index

4 % 4
= 0

```
def traverse(self):
    for i in range(self.size):
        print(self.arr[(self.rear+i)%self.capacity], end=" ,
```

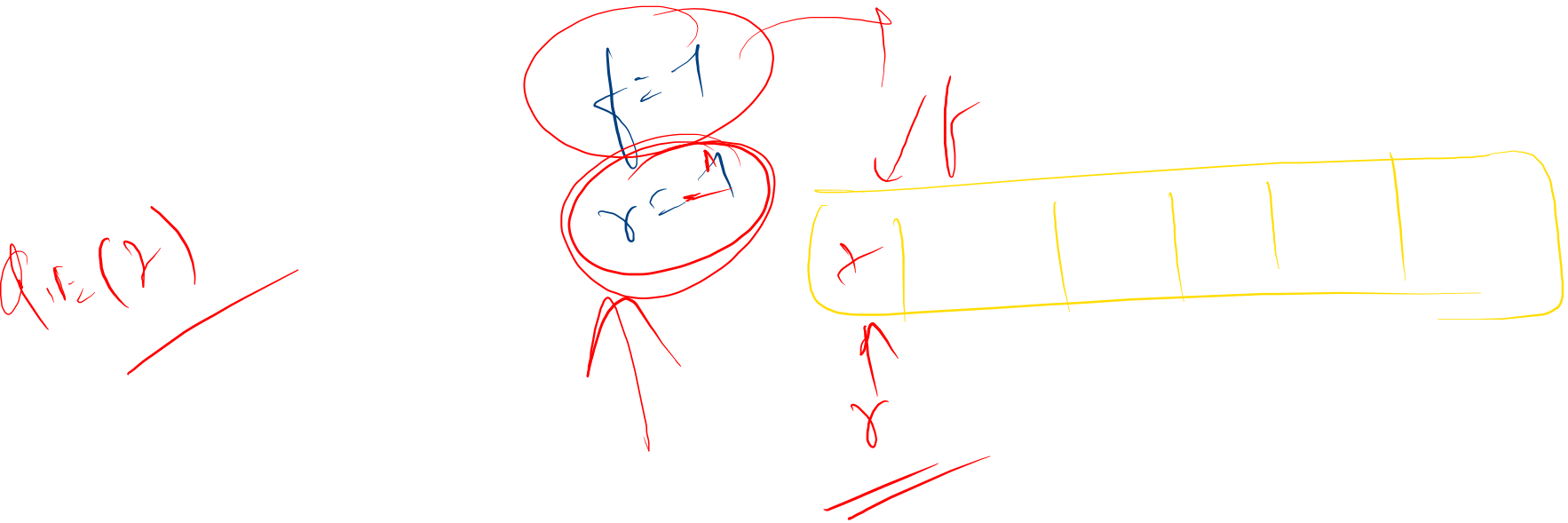


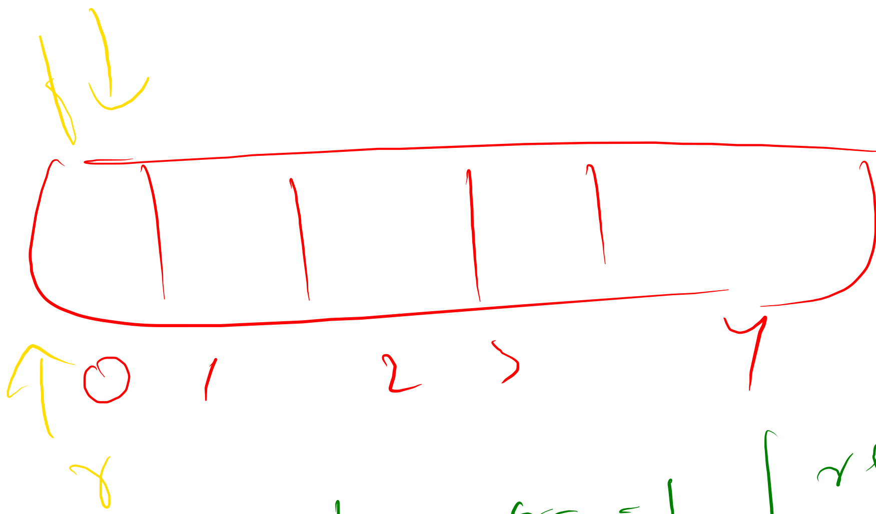
i=0 arr[4+0%5] = arr[4]

i=1 arr[4+1%5] = arr[5%5] = arr[0]

i=2 arr[4+2%5] = arr[6%5] = arr[1]

i=3 arr[4+3%5] = arr[7%5] = arr[2]

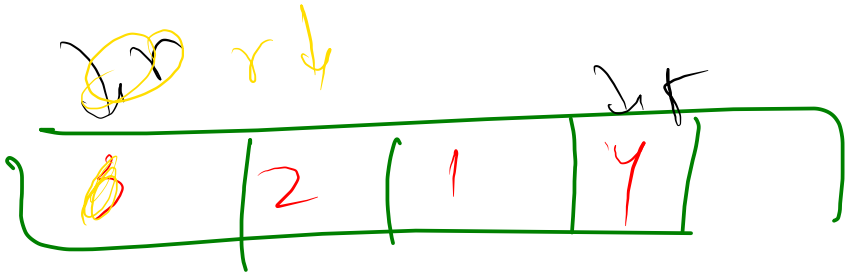




dequeue

f == r == -1 [reset to -1]

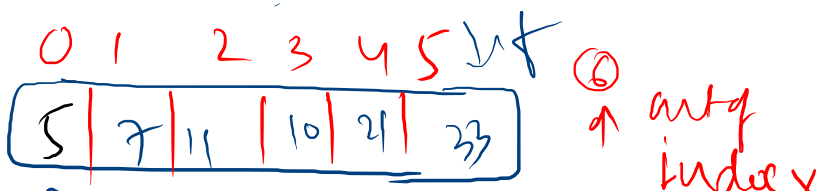
f == r → only 1 element present



r → moving point

Circular array implementation

Case 1



q.Fe(5)

q.Fe(7)

q.Fe(11)

q.Fe(10)

q.Fe(21)

q.Fe(33)

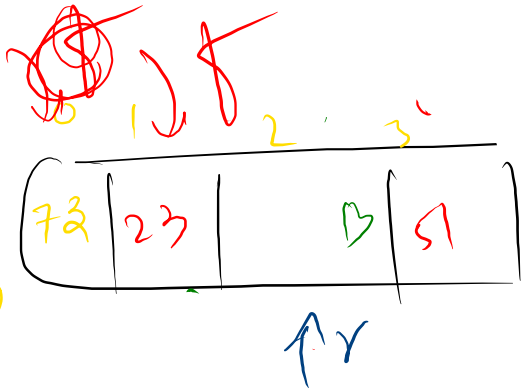
f+1 == r

6 == 0

(f+1) % size == r % size

=> 5+1 % 6 == 0 % 6

=> 0 == 0 ↑ full



q.E(7)

q.E(11)

q.E(13)

q.E(73)

q.D

q.E(51)

q.D

f+1 = 4th invalid

4 % size

f+1 == r

1+1 = 2

q.E(23)

q.E(32)

Conditions to check

full → $(r+1) \% capacity == r$] → if 0 it's full

Queue

- ① Arrays
- ② Linked List

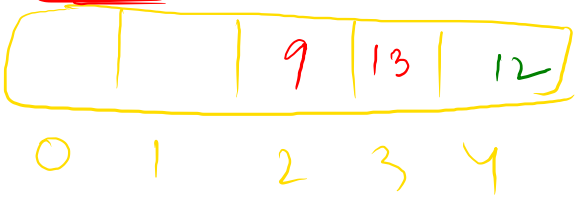
Array

Queue q

store data in array

rear

front



q.enque(5)

q.enque(7)

q.enque(9)

q.enque(13)

q.enque(12)

→ q.enque(31) } Q is full

space left

Still not able to use it

q.deque() → 5

q.deque() → 7

Circulation of whole array the way

Queue

front → index where we enqueue

rear → index where we dequeue

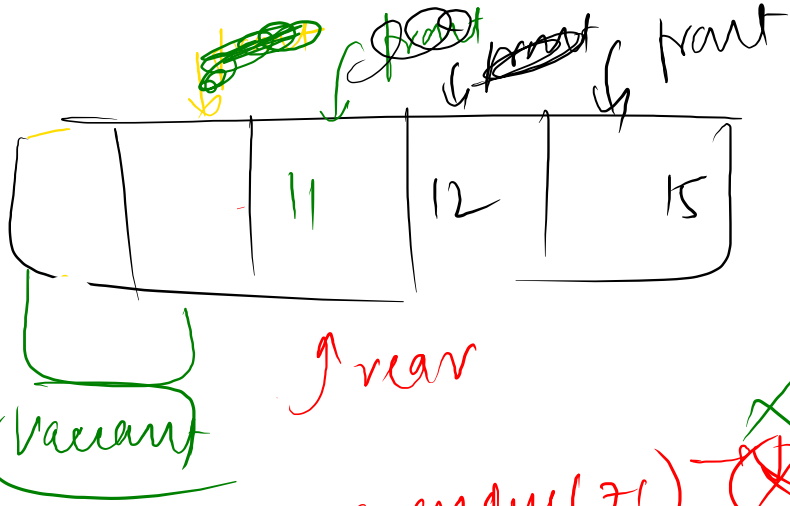
q.enqueue(5)

q.enqueue(7)

q.enqueue(11)

q.dequeue() → 5

q.dequeue() → 7



q.enqueue(12)

q.enqueue(15)

q.enqueue(71)

enqueue → O(1)

dequeue →

TC