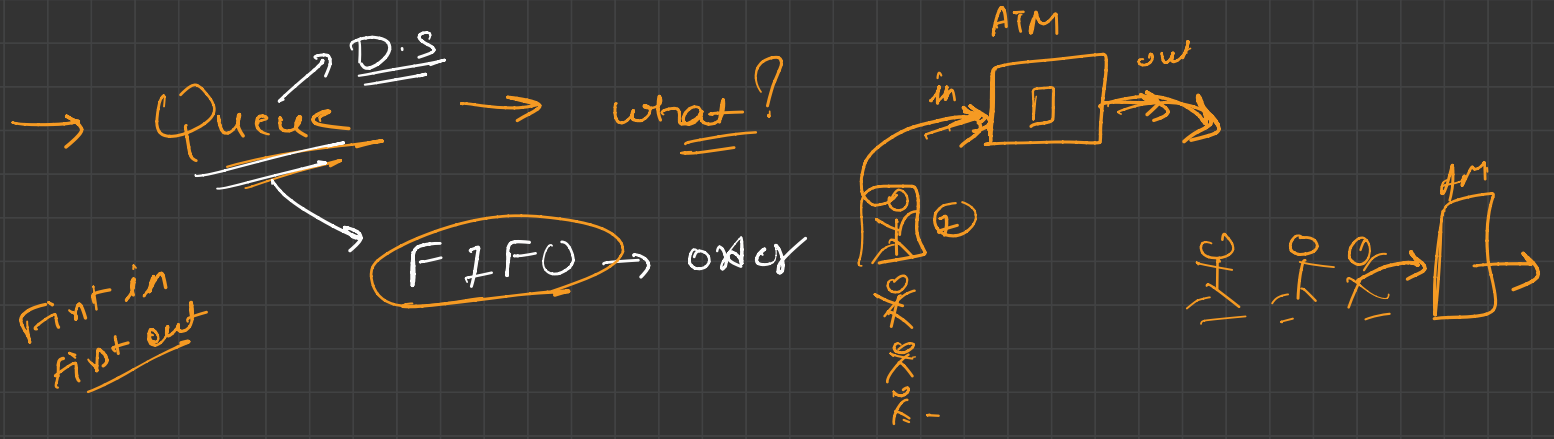
Queue



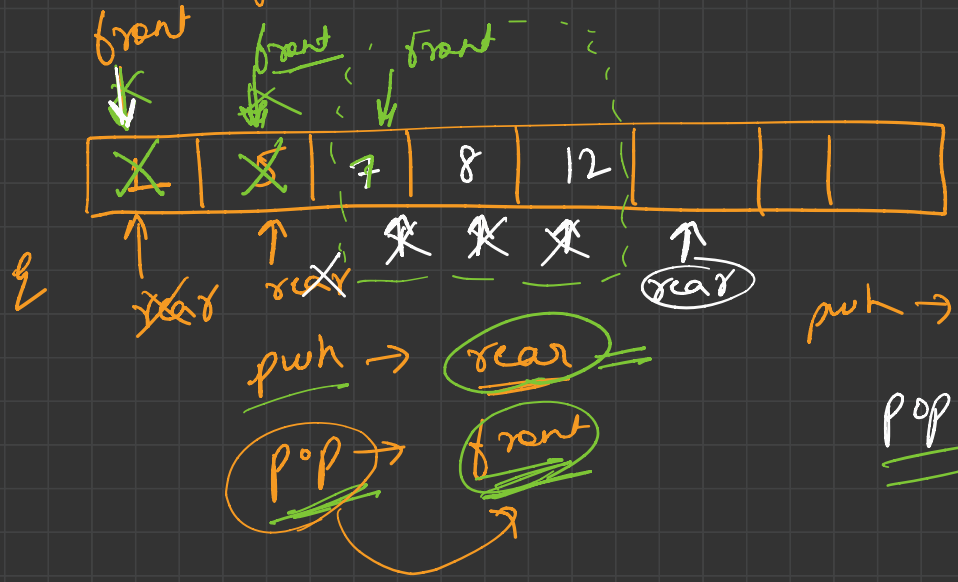
queue.push(5)

7
8
12

FIFO

pop → FIFO

pop

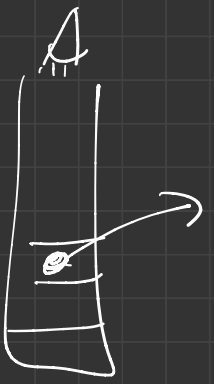


push → 1, 7, 7, 8, 12

pop(), pop()

queue → push / insert
pop / remove
size
isEmpty
} Operation

STL:



#include <queue> →

```
queue<int> q;
```

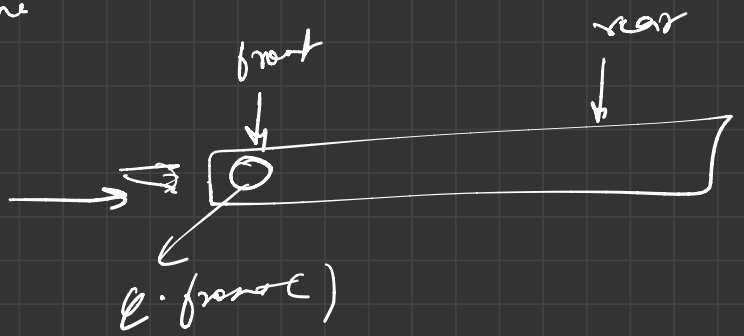
push → q.push(17);

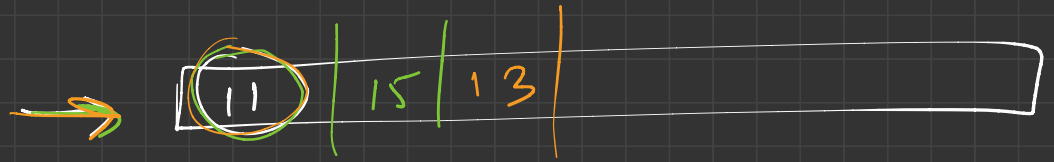
pop → q.pop();

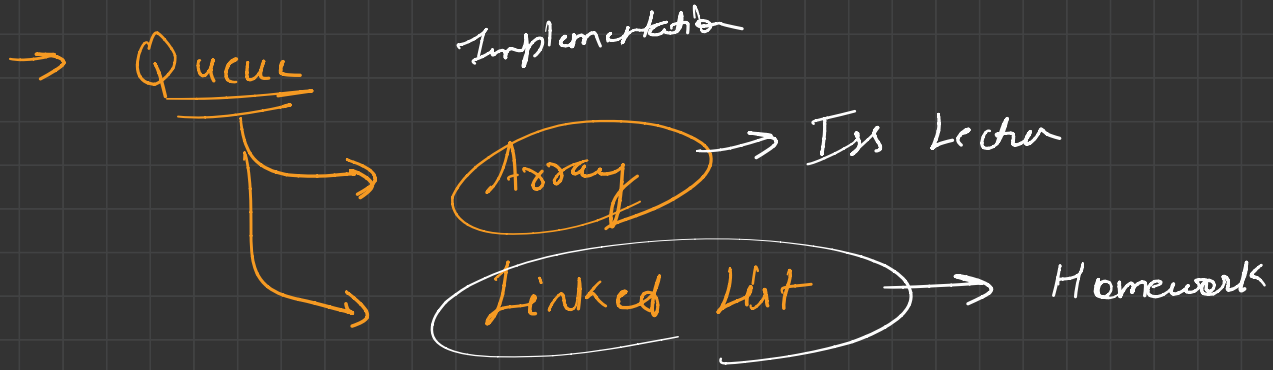
size → q.size();

isEmpty → q.empty() → T/F

sabse aage front me Kona elem → q.front()

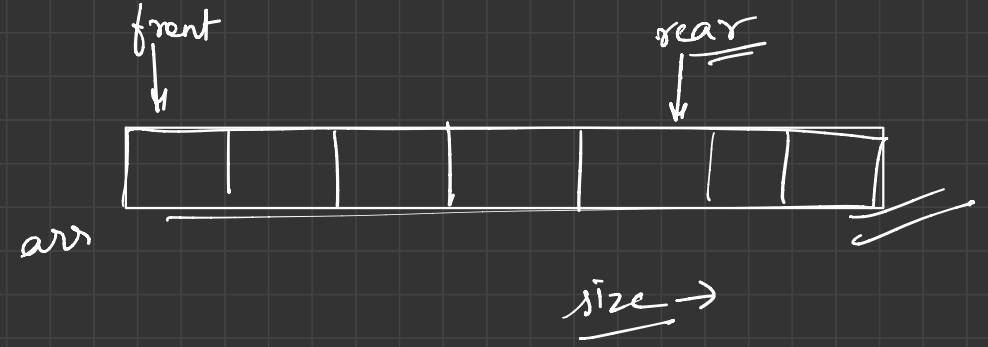






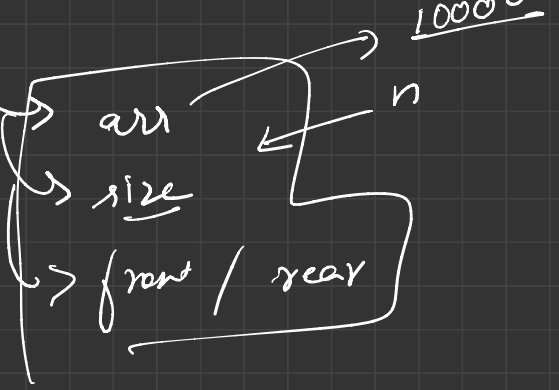
insert / push → enqueue

dequeue → pop

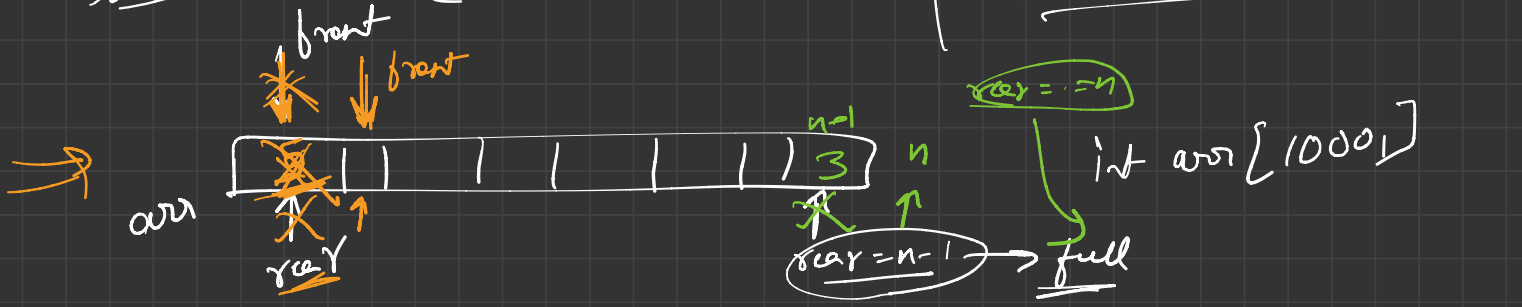


class Queue

→ Data Members



starting → front = 0
rear = 0



pop()

push:-

if (full) → Queue is full

else

arr[rear] = element

rear++;

empty

front == rear

(5)

pop:-

if (empty) → Queue is Empty

front == rear
↓
empty

isEmpty
↓
O(1) → T.C

arr[front] = -1

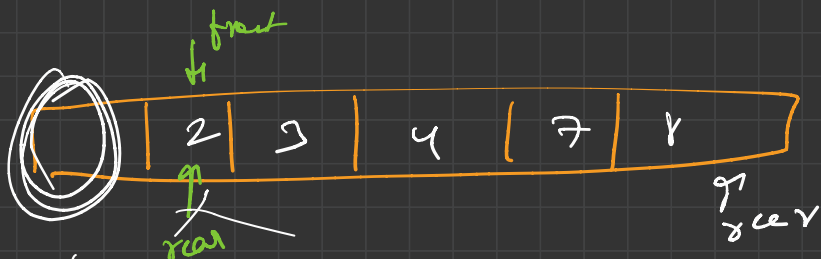
front ++;

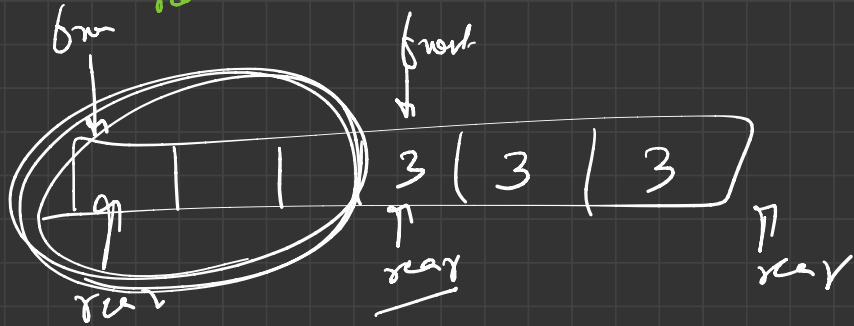
fail jaa tak hai

if (front == rear)
{ front = 0; }

rear = 0;

}



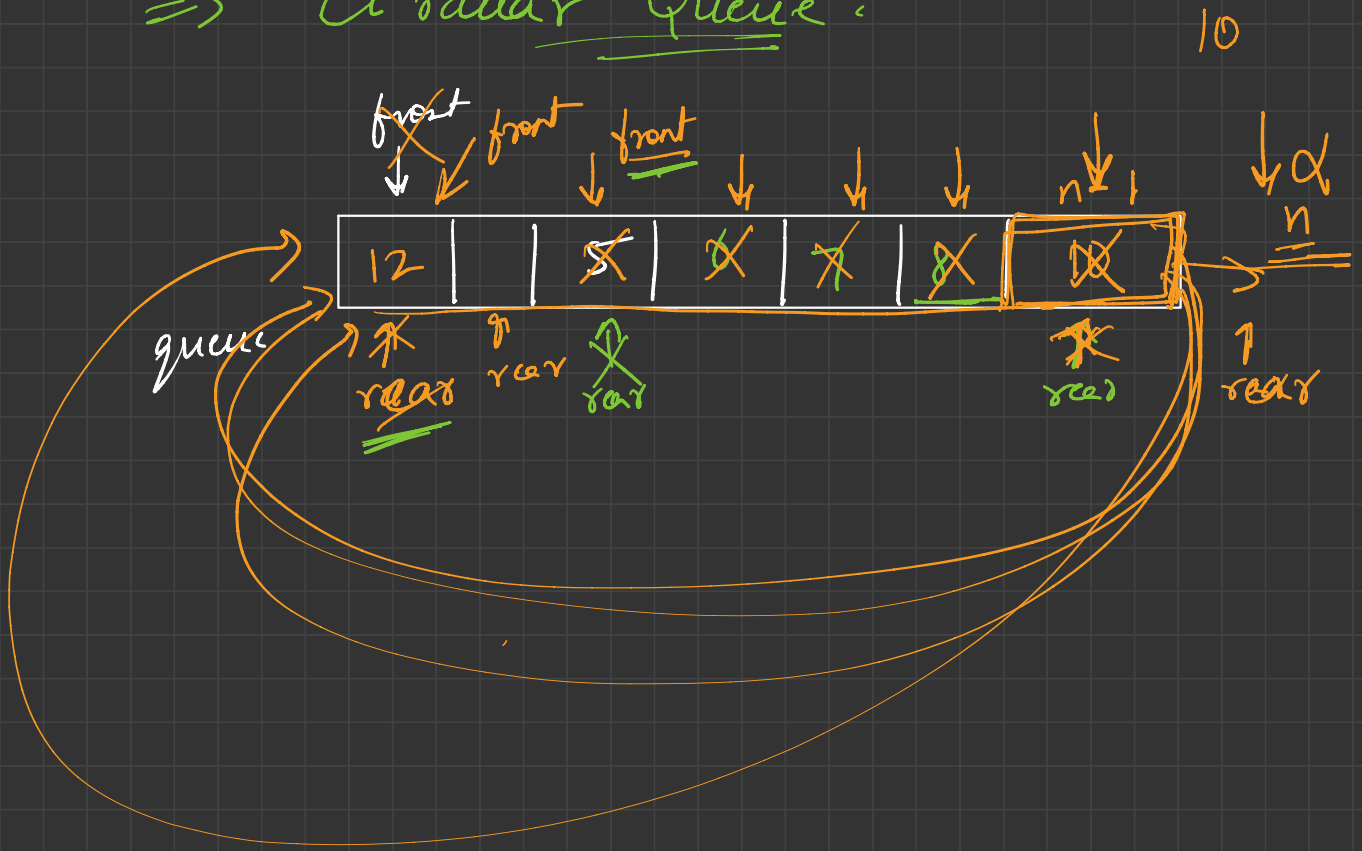


isEmpty → front = rear

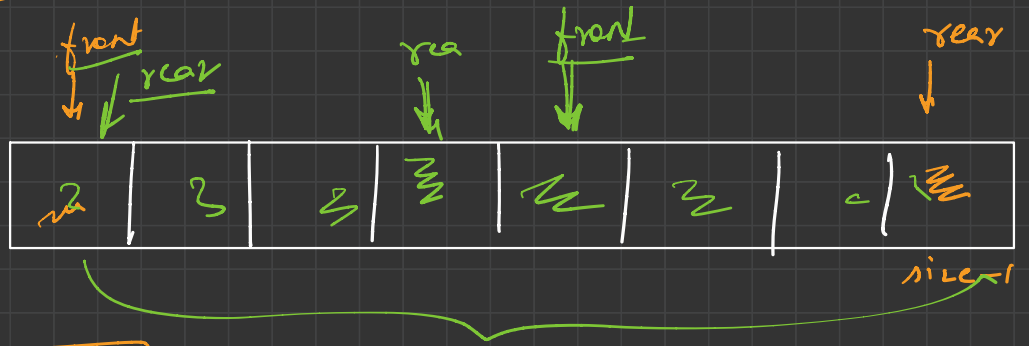
Front()
→ if (empty) → return -1

return arr[front];

⇒ Circular Queue :-



push :-



queue

front = -1

rear = -1

push → if (full) → "Queue is Full"

(front == 0 && rear == size - 1)

or

rear == (front - 1) % (size - 1)

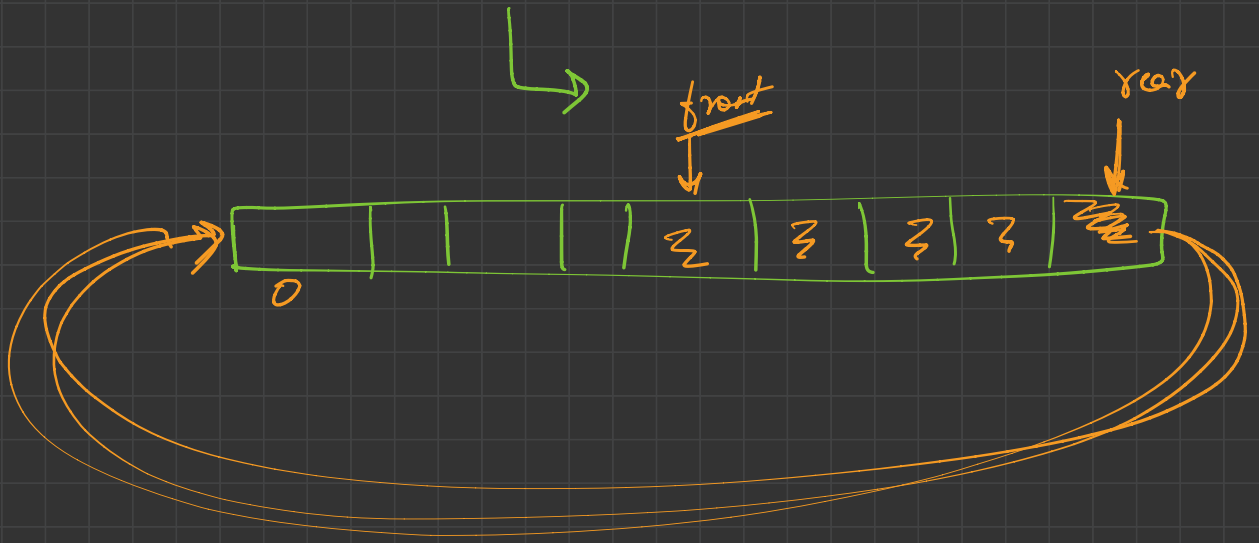
↳ if (front == -1) // first element

{

front = rear = 0;

arr[queue] = data;

}



if (rear == size - 1 && front != 0)

{

rear = 0

arr[rear] = data;

}

dir

```
year++
arr[year]= data;
```

Dequeue / pop :-

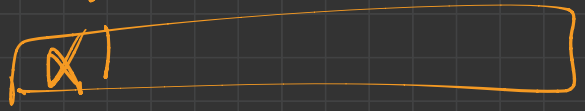
→ if (empty) → Queue is Empty

front == -1

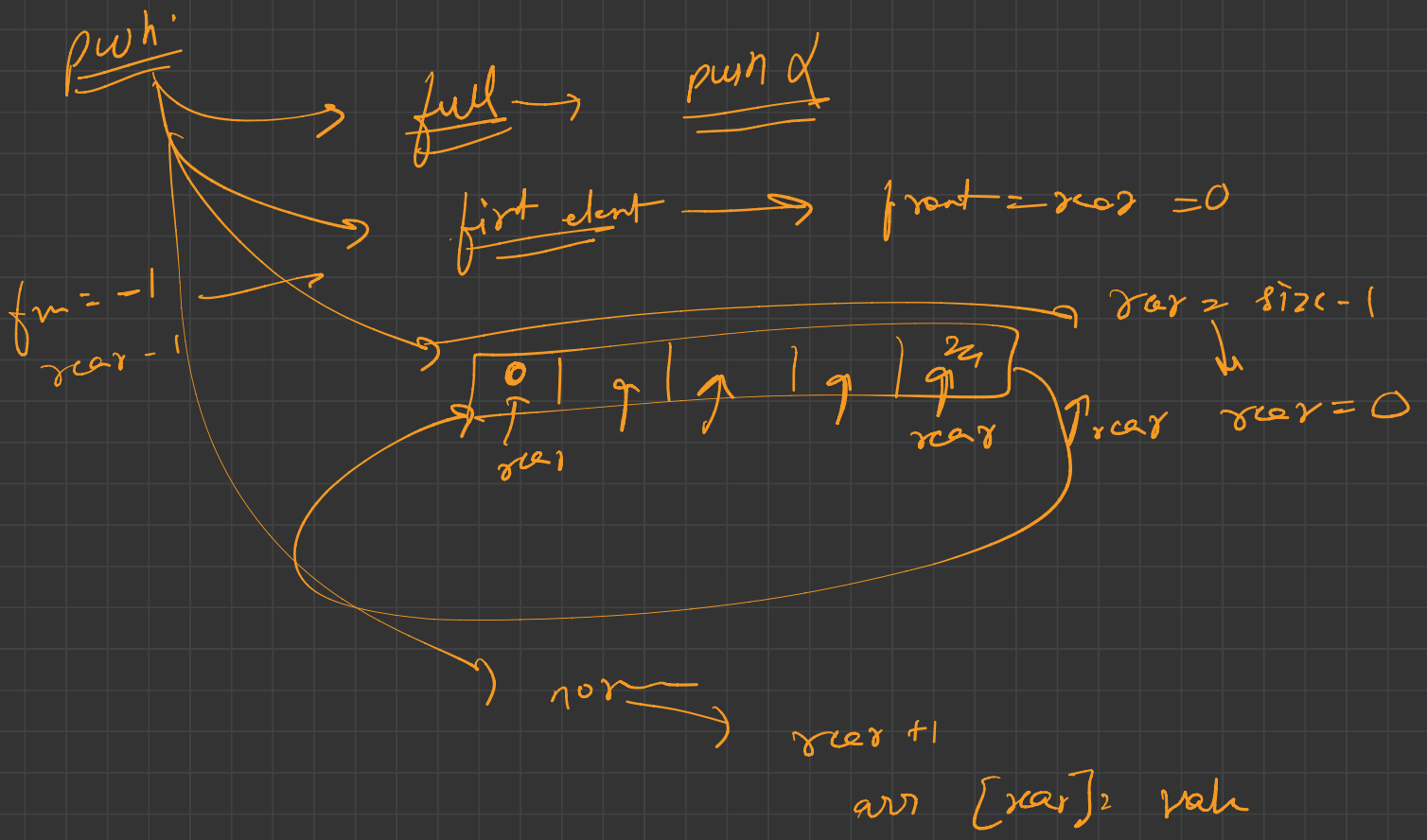
→ // single element in queue

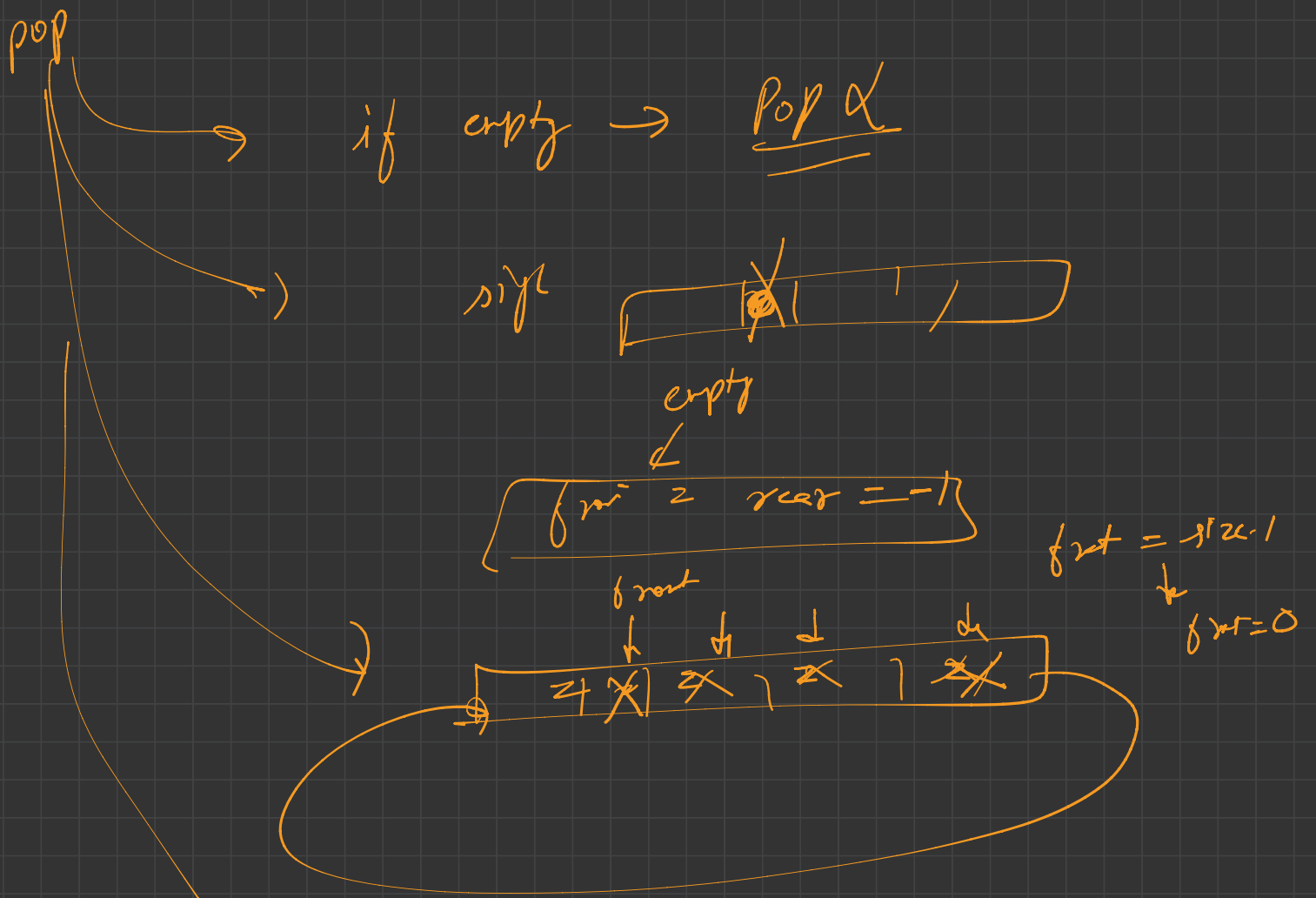
if (front == rear)

front = rear = -1;




```
if (front == size - 1)
    front = 0;
```


push
full
push ✗
first elem
front = rear = 0
fr = -1
rear = -1
rear = size - 1
0
rear
rear
rear = 0
rear
nor
rear + 1
arr [rear] = val


pop
if empty → Pop X
size
empty
front = rear = -1
front = size-1
front = 0

↳ normal

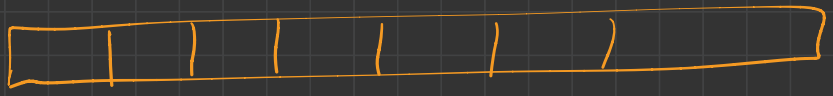
front ++

Normal Queue

Circular Qu

i/p restricted qu → i/p → only on one side (rear)

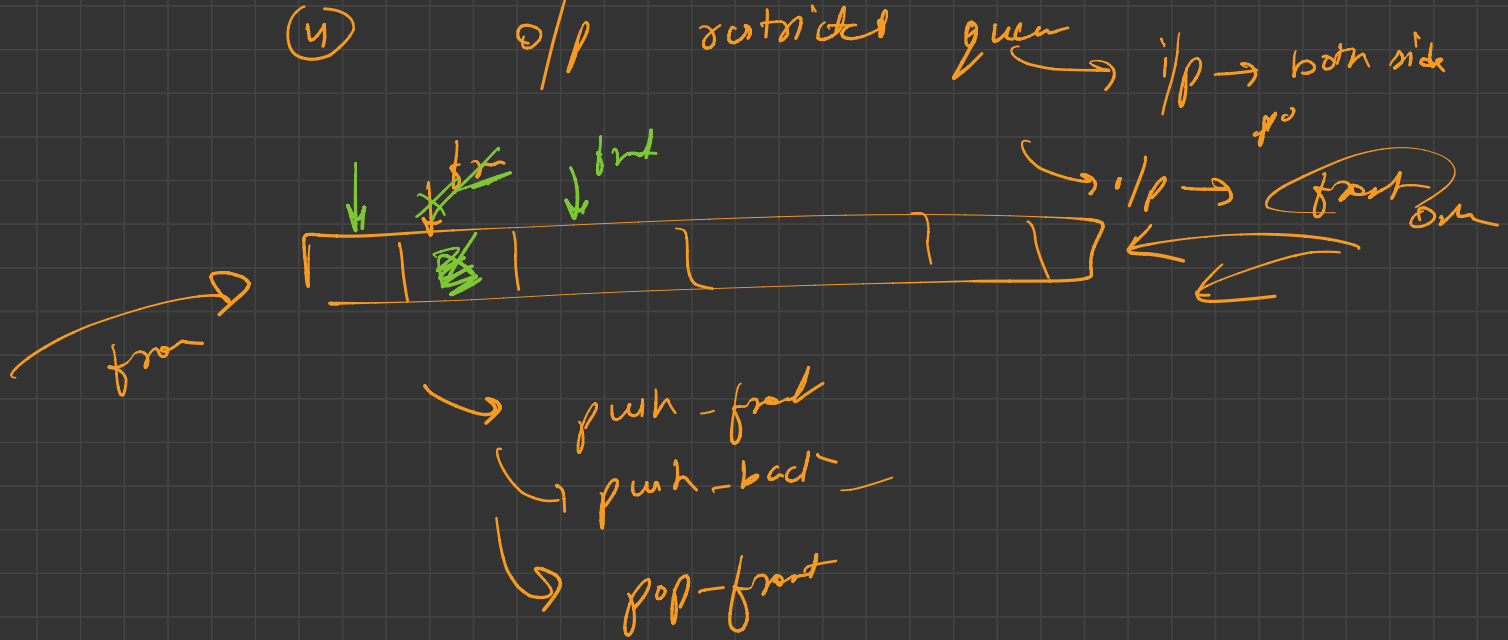
→ o/p → both side possible



push_back (rear)

pop_front (front)

pop_back (rear)

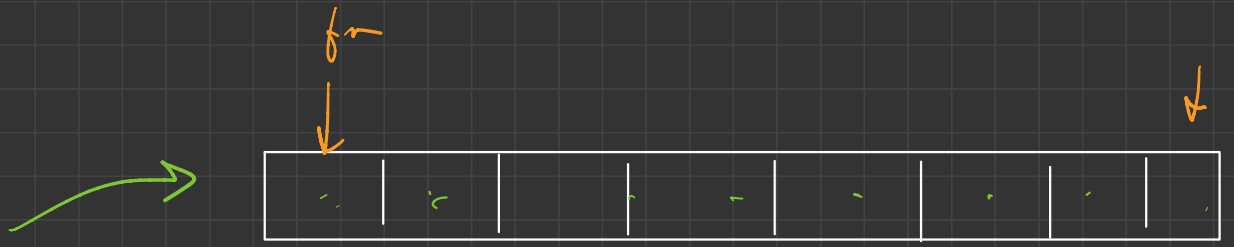

(4) o/p restricted queue
i/p → both side
i/p → front
front
front
front
push_front
push_back
pop_front

Doubly Ended Queue → (Stack / Queue)

STL :- → Use

Implementation → Let's do it

rear



front = -1

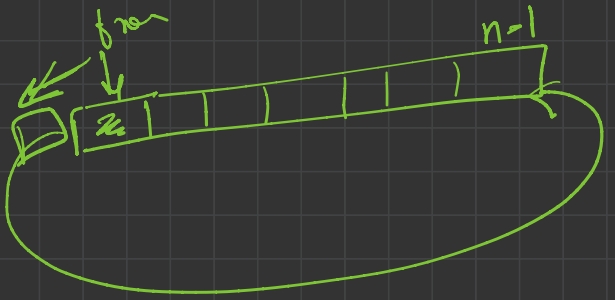
rear = -1

push front

if full → Queue is full

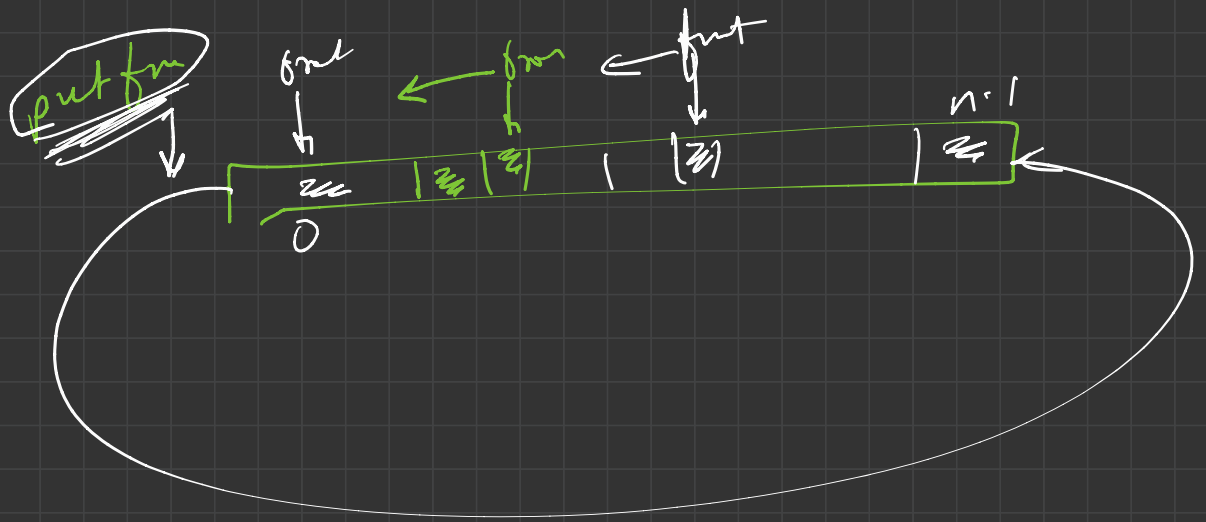
if // first element

front == -1 → front = rear = 0



if (front = 0) → front n-1

normal flow → front --;



push rax → same

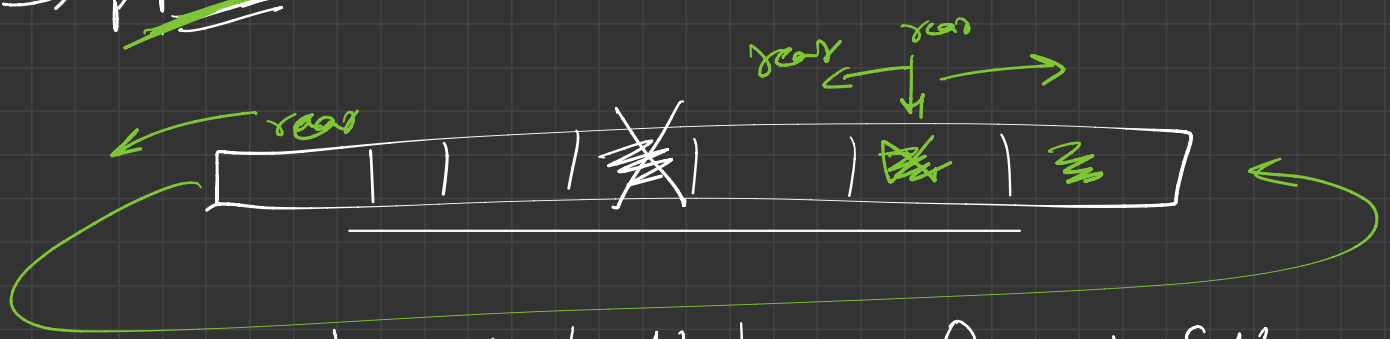
→ same as

pop front

→ pop_back

→ pop-back ← (rear

put



↳ if (empty) → Queue is Empty

↳ || single element

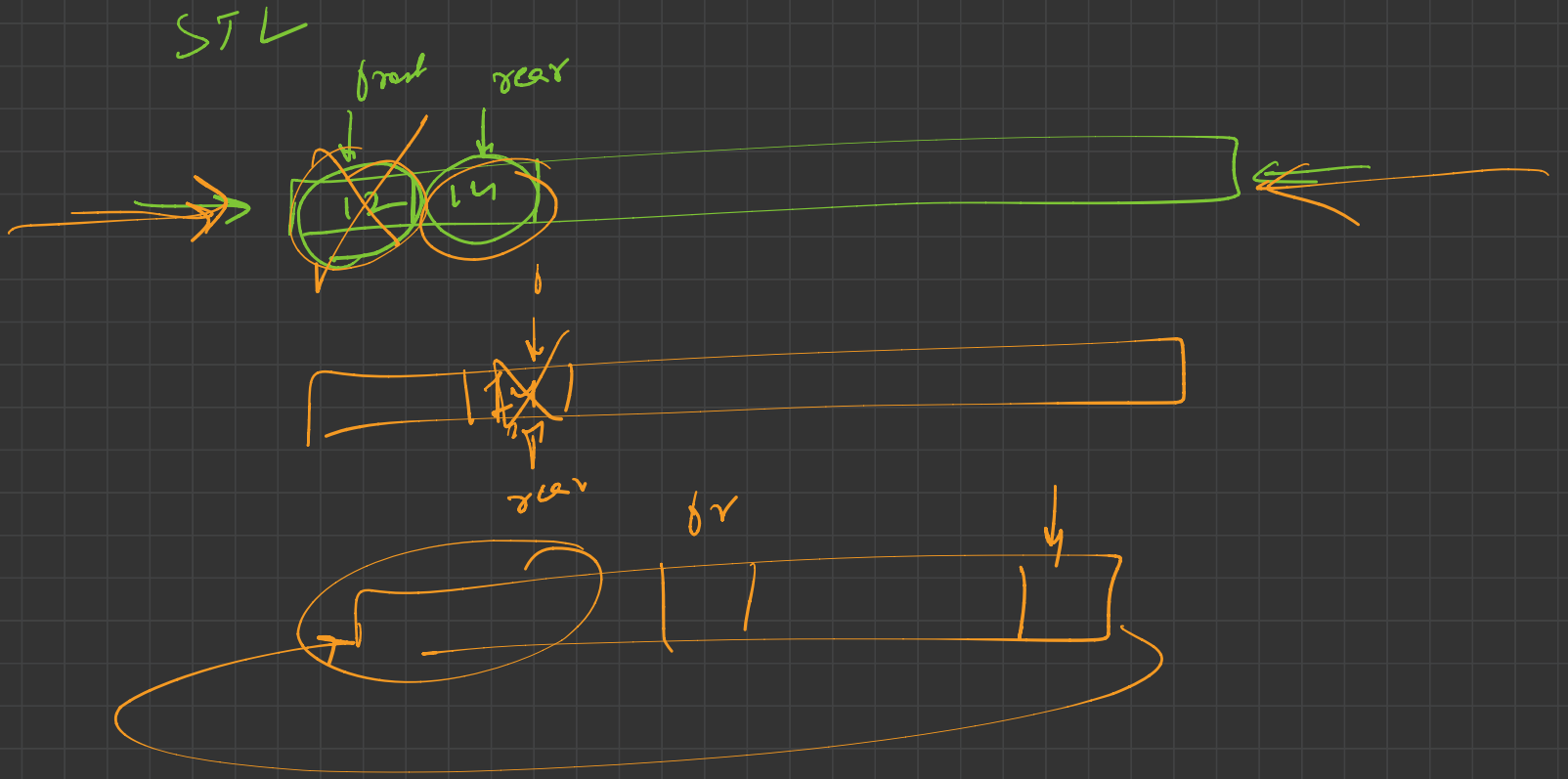
↳ front = rear = -1

↳ || cyclic nature

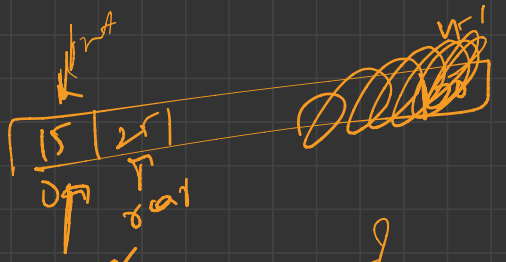
if (rear == 0) → rear = n-1

↳ normal flow

rear --;


STL
front
rear
rear
fr

(-10/01)

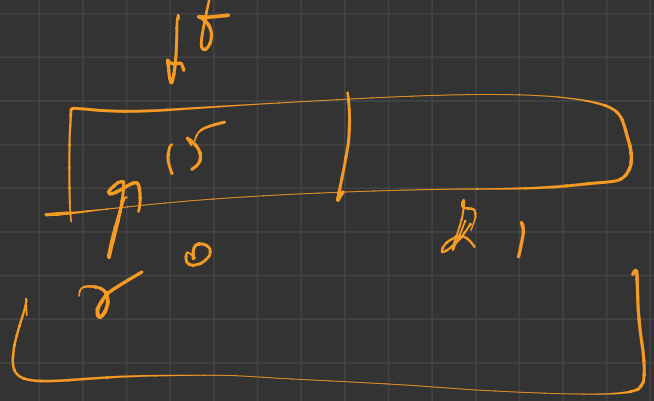


size 2

push front 20 → fail

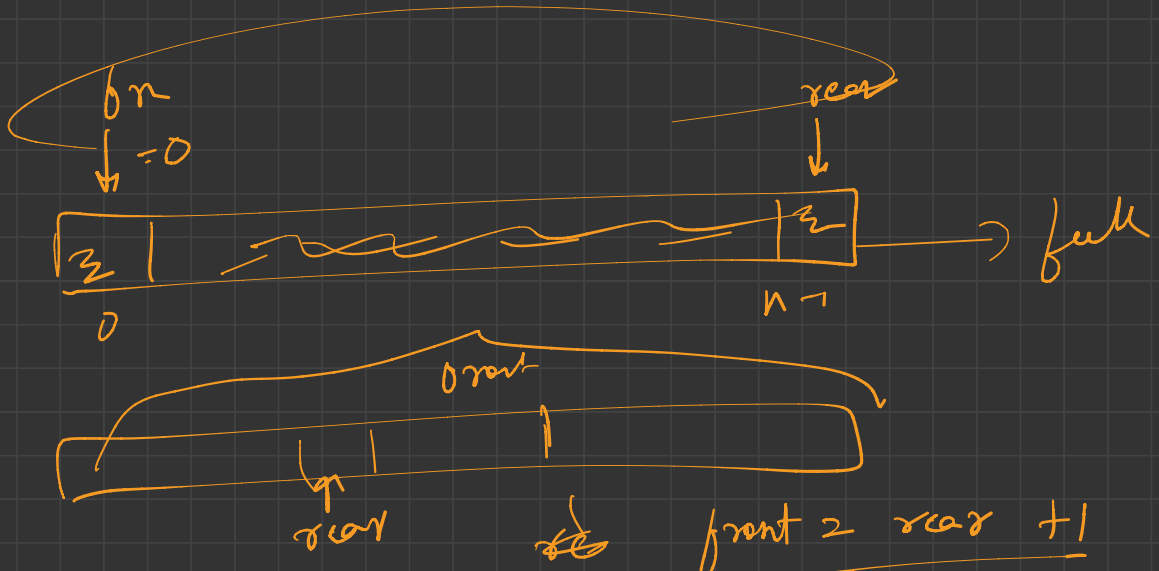
isfull → TRUE

getrear → 25



F // 0 == (-1)%(1)

0



front = rear +1

front = 0

(front - 1)