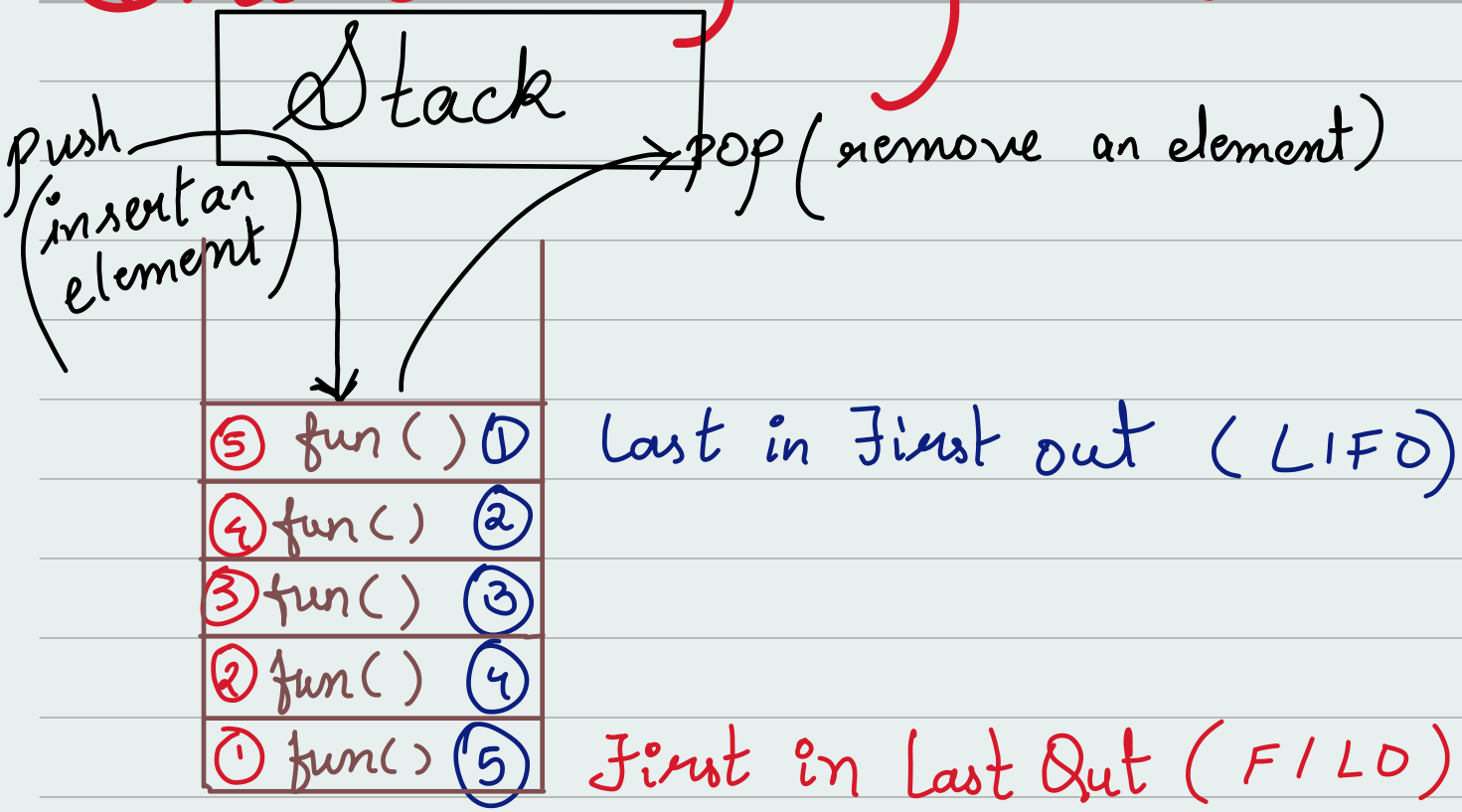
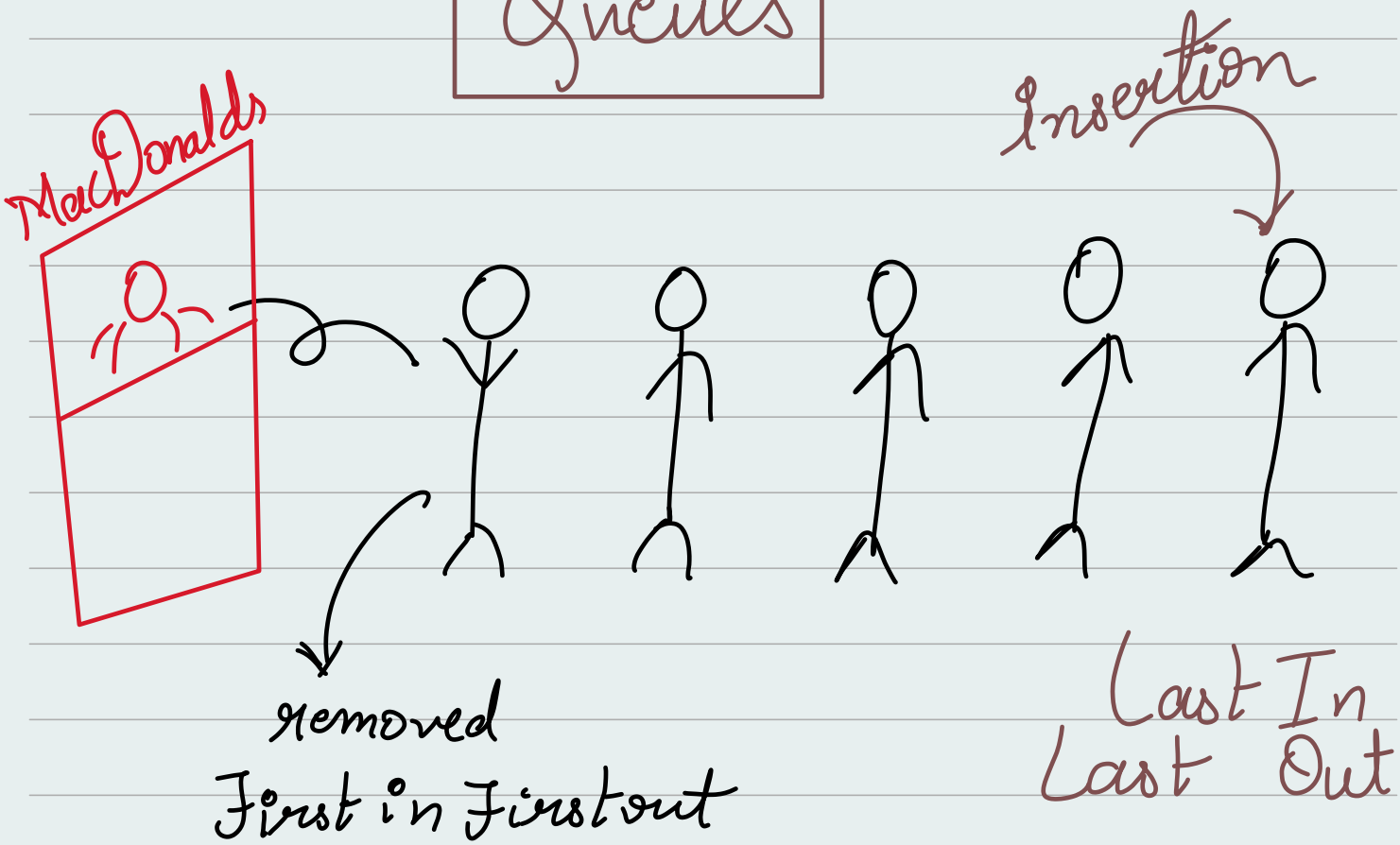
Stack & Queues :-



Last in First out (LIFO)

First in Last Out (FILO)

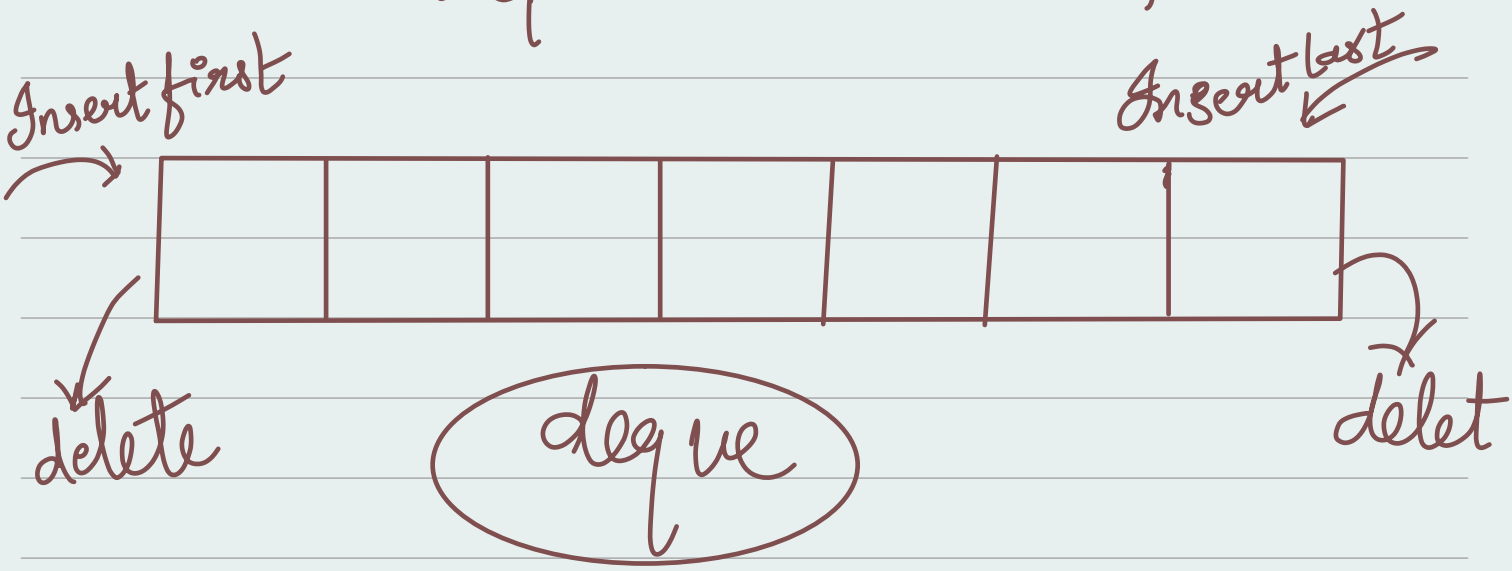
Queues



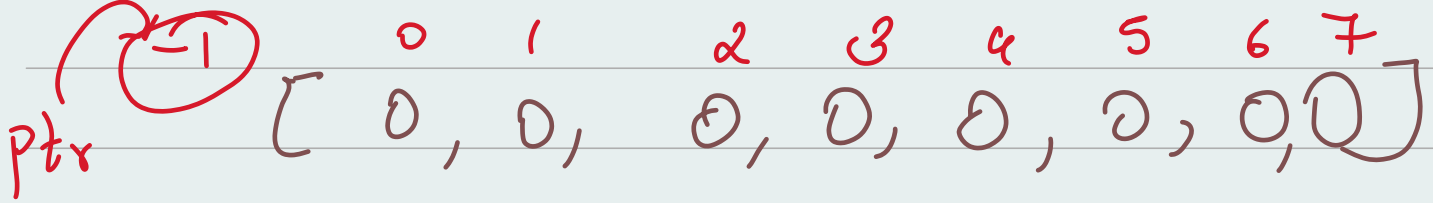

```
Queue<Integer> queue = new LinkedList<>();
```

Deque : Counter on both the sides we can insert & remove elements from both the sides.

```
Deque <Integer> deque = new ArrayDeque<>();
deque.add(10);
deque.addLast(78);
deque.removeFirst();
```



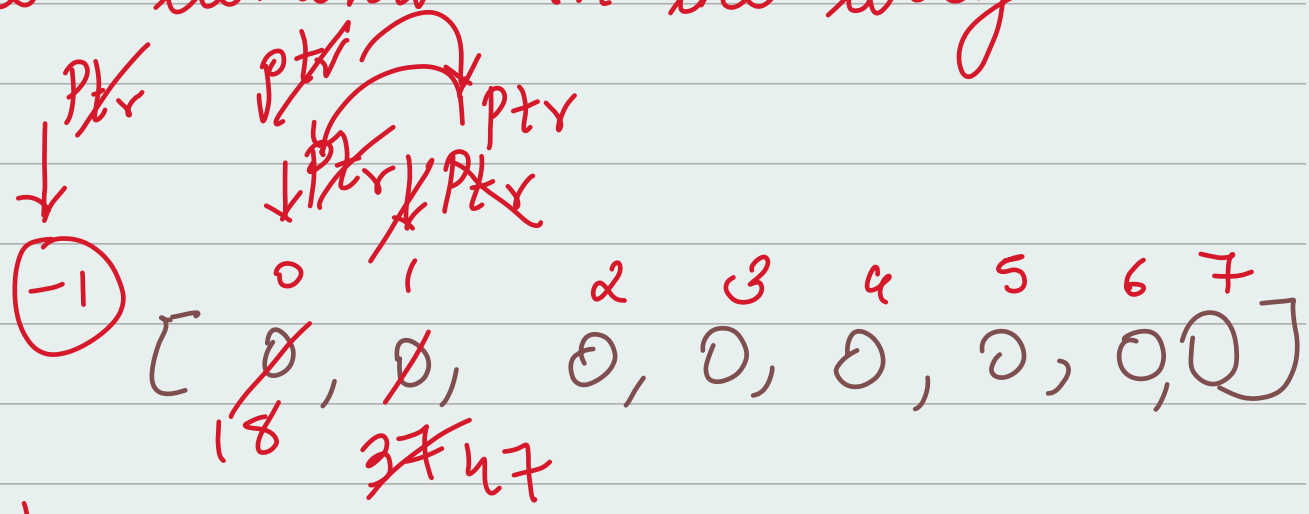
* Custom Stack.



ptr pointed to -1 index value.

So if we add or insert a item in this array we have to maintain

a pointer that would add or remove the elements in the array



Ptr		
0	add an element (18) increment	-1 + 1 = 0
1	add an element 37	0 + 1 = 1
0	delete an element 37	1 - 1 = 0
1	add an element 47	0 + 1 = 1

Using this pointer to go left and right to add or delete

when array is full { ptr == data length-1 }

Custom Queue:

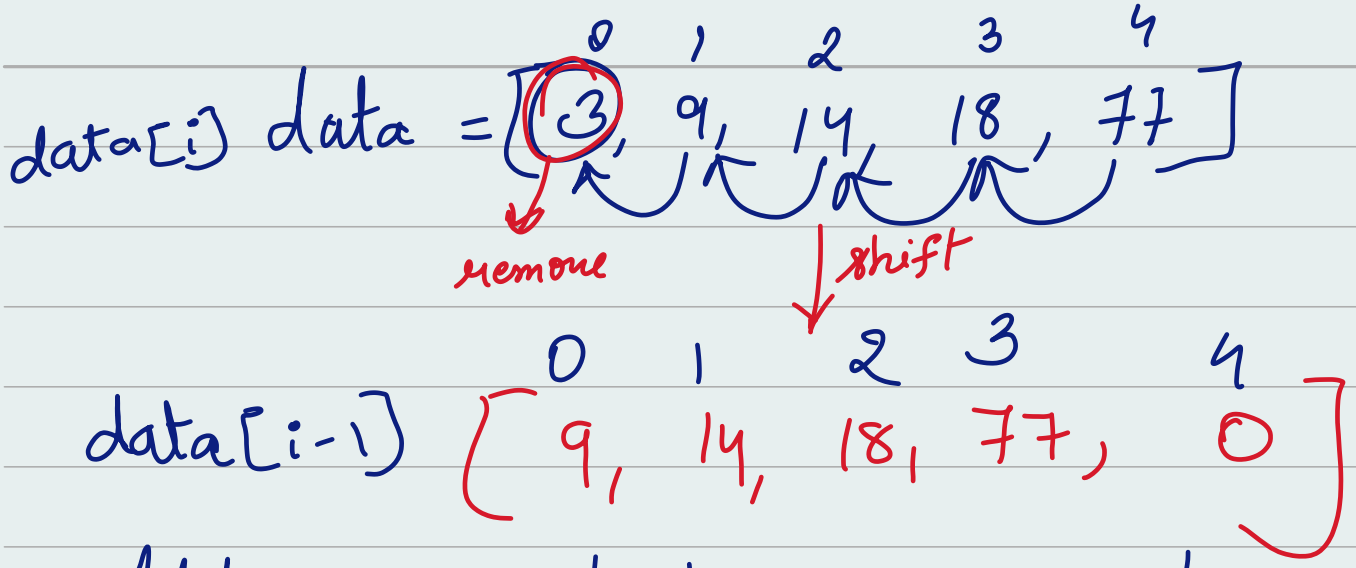
0 1 2 3 4 5

data = [34, 45, 0, 0, 0, 0]

when data reach index 5 it's full.

end

2



→ Adding complexity is O(1) because we are adding an element

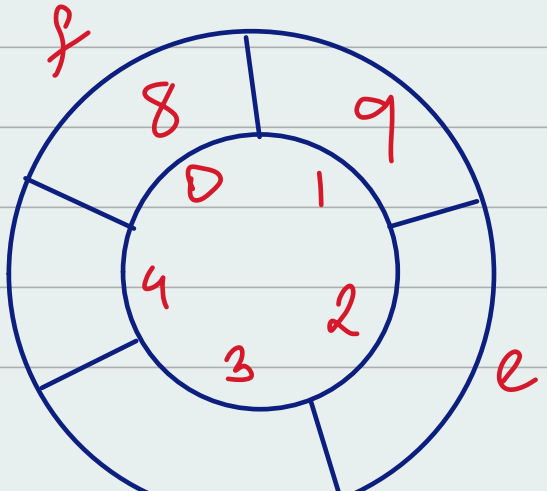
→ But removing an element changes the complexity to O(N) cause it's shifting left to O(N-1) which is O(N)

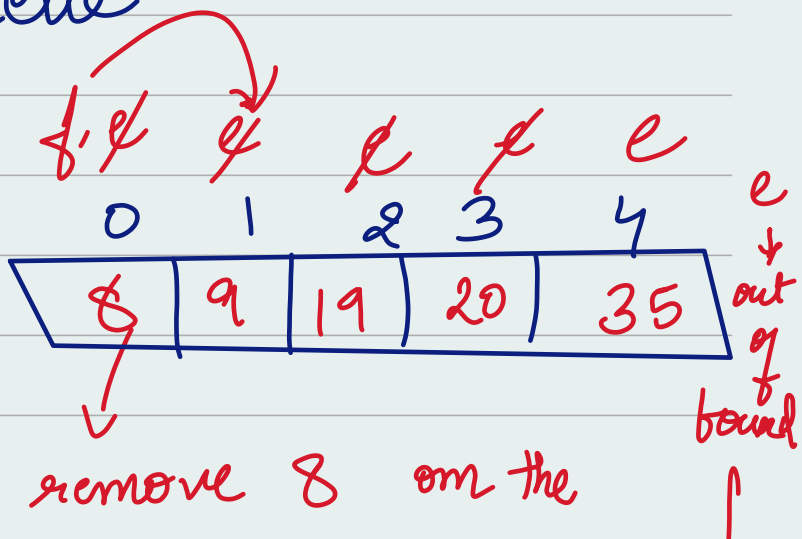
```
public int remove() throws Exception{
    if(isEmpty()){
        throw new Exception("Queue is Empty");
    }
}

int removed = data[0]
// shift the elements to left,
```

```
for (int i = 1; i < end; i++){
    data[i-1] = data[i]
}
    end--;
    return removed;
}
public int front() throws Exception {
    if(isEmpty()){
        throw new Exception("Queue is Empty");
    }
    return data[0];
}
```

* Circular Queue





index number zero.
front item is getting removed.

→ Use remainder theorem

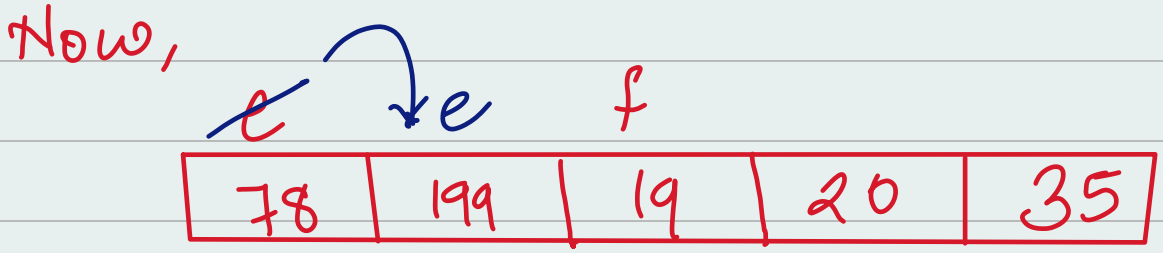
e % Size of the array.

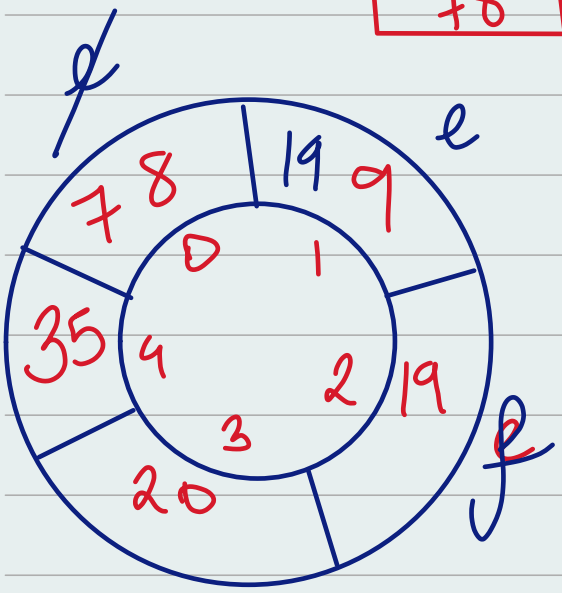
5 % 5 = 0

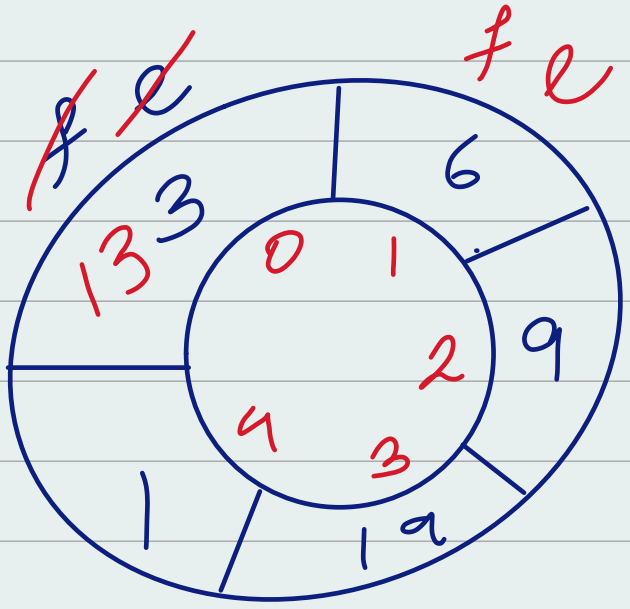
end is at 0th index again.

6 % 5 = 1

Now,







(Remove an item)

(Insert an item)

Remove 3 insert 133.

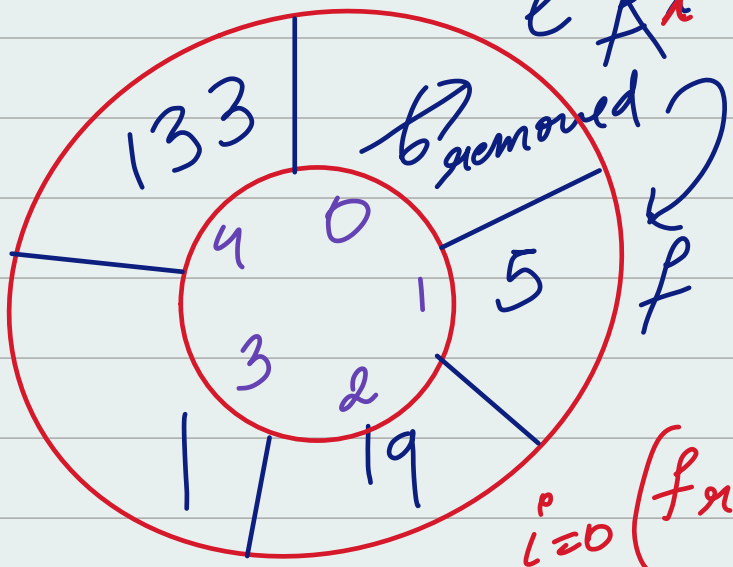
taking i pointer tell its not equal to end using do while loop

from 1 to 5 index.

when elements are copied dynamically

After for loop cond

temp[i] = data[(front + i) % data.length];



front = 2

i = 0

i=0 (front + i) % data.length

2nd index value

3rd index value

4th index value

i = 0 2 % 5 = 2

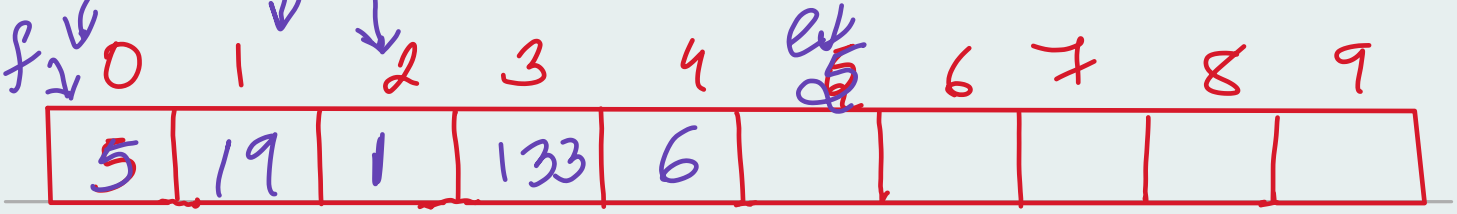
i = 1 3 % 5 = 3

i = 2 4 % 5 = 4

i = 3 5 % 5 = 0

i = 4 6 % 5

It will go data.length times



$i = 5$ $i \not< data.length$

terminate loop

So, front = 0

& end = data length