@Veves

→ first in, first out

→ enqueue (insert an element at end of queue)

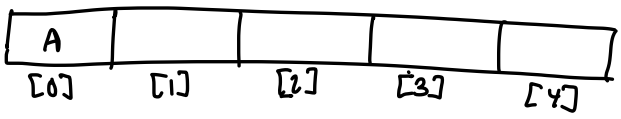
→ dequeue (removes & returns the element at front of queue)

```
Public interface Queues<E>{
    int size();
    boolean isEmpty();
    E first();
     void enqueue(E e);
    E dequeue();
```

→ Array Linear Queues

→ front of queue is fixed at first slot in the array whereas the rear of the queue moves down with each Enqueue. Now we dequeue the front element in the queue.

(a) queue.enqueue('A')



front = 0
rear = 0

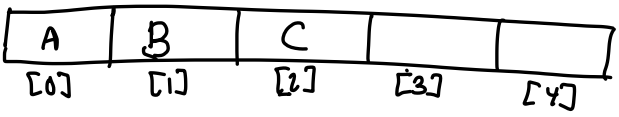
Enqueue: O(1)

Dequeue: O(1)

(b) queue.enqueue('B')

front = 0
rear = 1

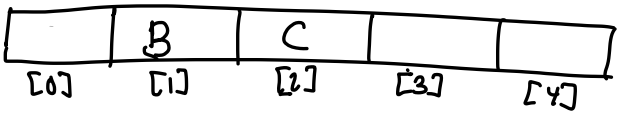
(c) queue.enqueue('C')



front = 0
rear = 2

* In dequeue, instead of moving elements up to beginning of array, it merely just increments the index of front indicator to next slot

(d) queue.enqueue(item)

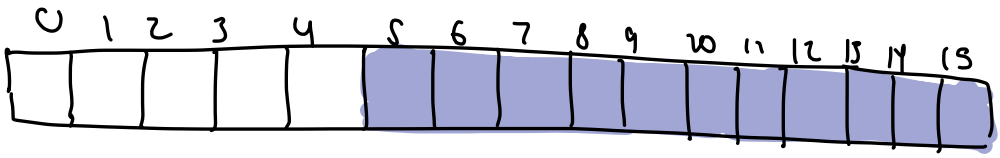


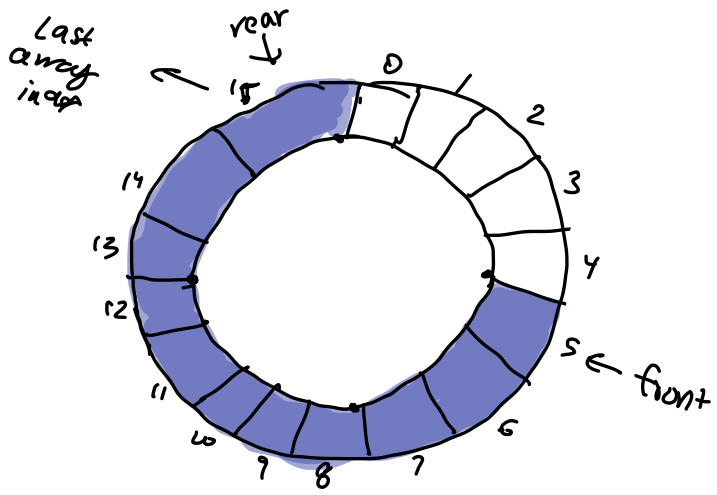
front = 1
rear = 2

→ Queue is full when if rear == Max_Items - 1

But when removing elements, we have empty slots at the low array index

→ Circular Array Queue

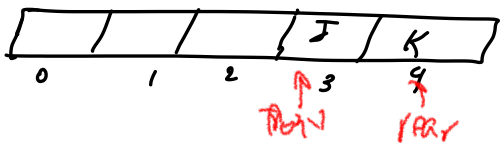




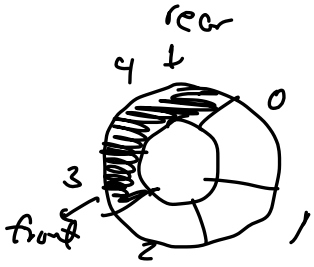
→ When rear reaches the last array index, move rear to index 0.

```
rear = (rear + 1) % maxQue;
```

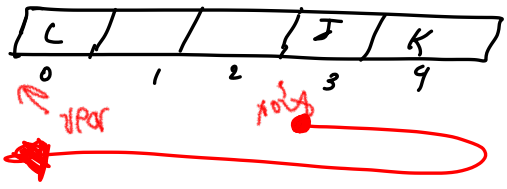
(a) queue.enqueue('L')



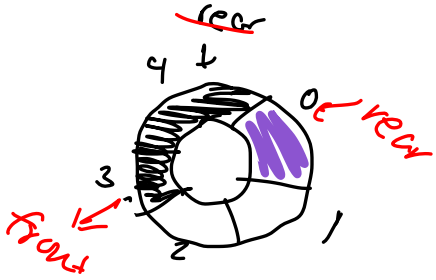
front = 3
rear = 4



(b) can wrap the queue around to top of array

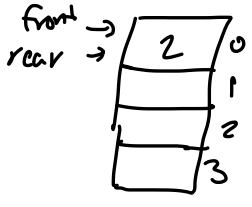


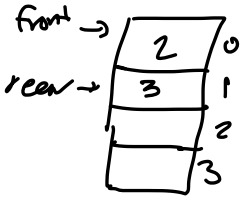
front = 3
rear = 0

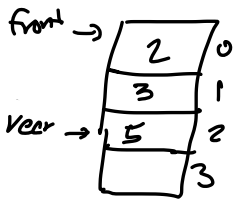


→ Example

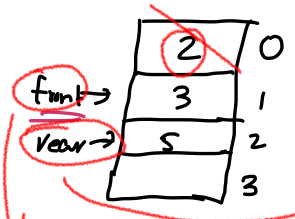
- q.enq(2)
- q.enq(3)
- q.enq(5)
- q.deq(item)







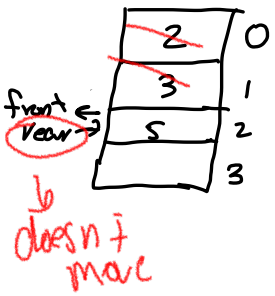
item = 2



doesn't move

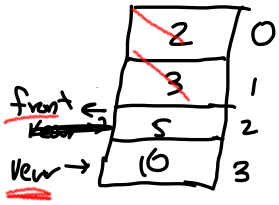
→ front moves to next spot & 2 is deleted

• q.deq(item)
item = 3

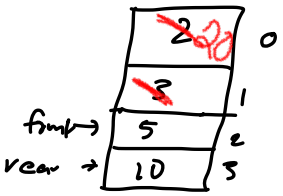


doesn't work

• q.enq(10)



• q.enq(20) ??



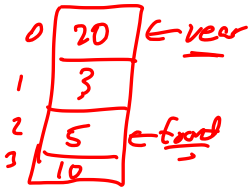
Let queue elements wrap around

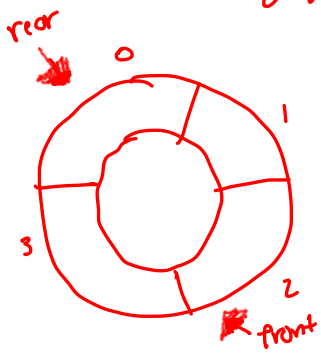
```
if (rear == maxQue-1)
    rear = 0;
else
    rear = rear - 1;
```

or

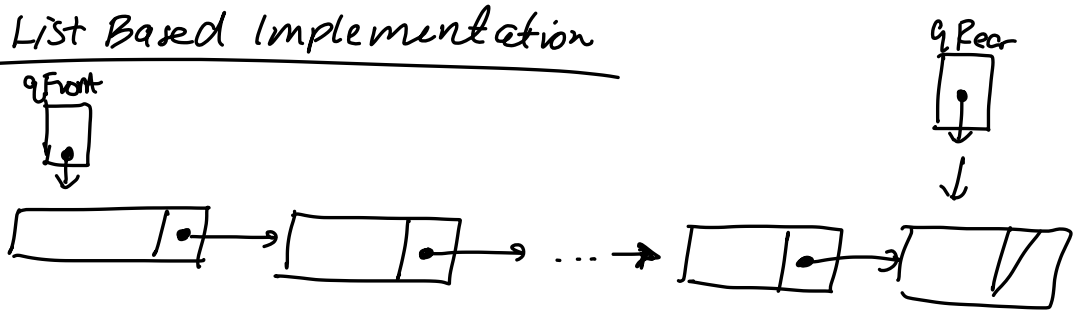
```
rear = (rear + 1) % maxQue;
```

wrap Ard:





→ Linked List Based Implementation

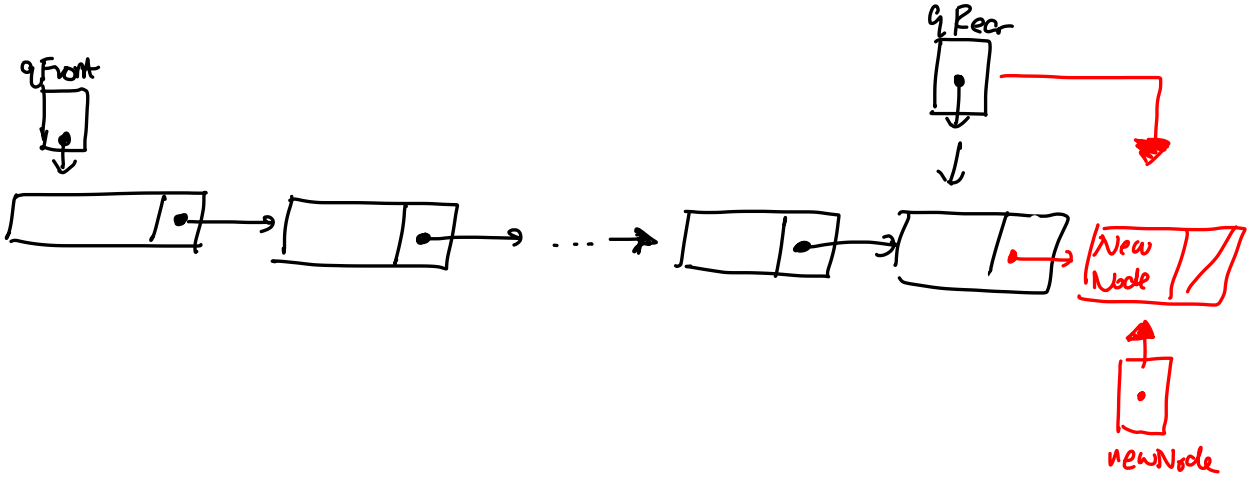


• enqueue(ItemType newItem)

- function: adds newItem to rear of queue
- Precond: queue has been initialized & is not full

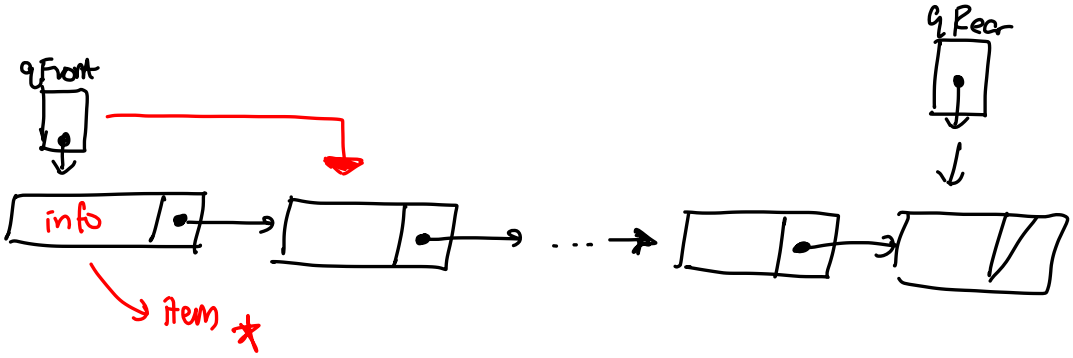
→ rear.next = newNode

→ rear = newNode

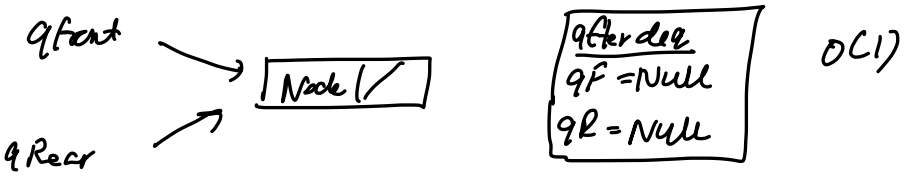


- dequeue:

  frontPointer = front.next

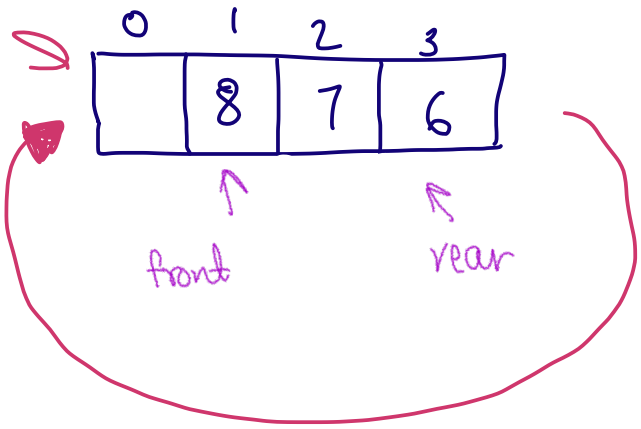


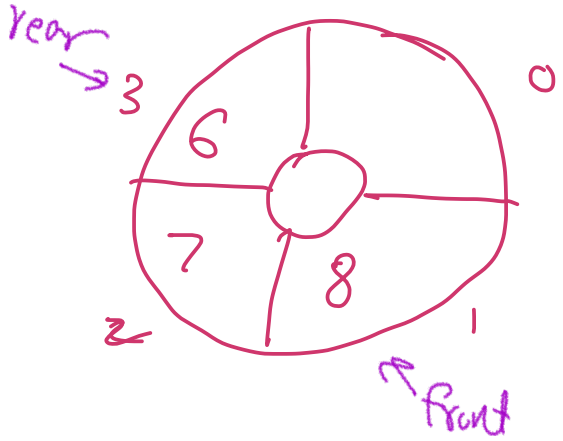
1 Special Case/ queue contains only one element



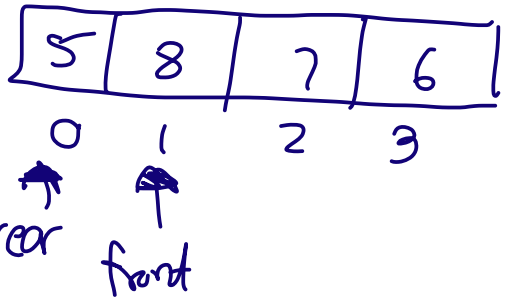
Circular Queues

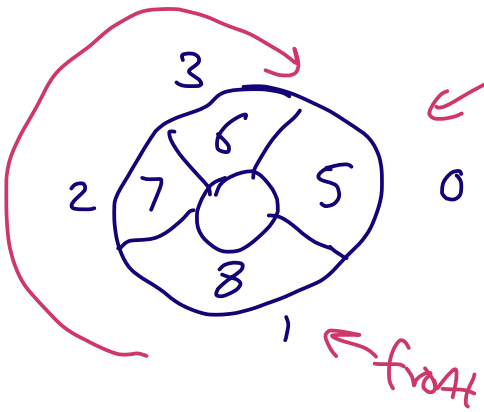
exists a space





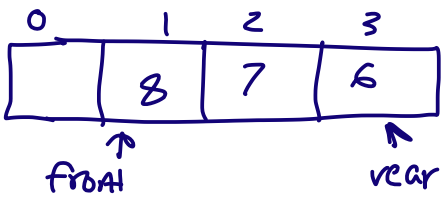
enqueue(5);





rear (still ahead of front)

↳ value of rear is < value of front, but position is ahead



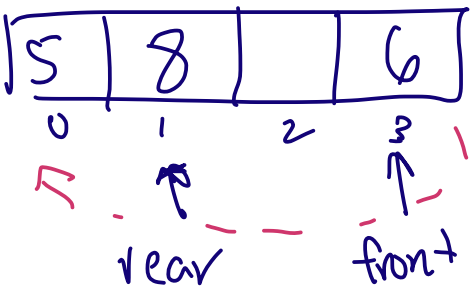
Add(5)

rear++ = 4

← rear = rear++ % length

rear = 4 % 4

rear = 0



Delete from front()

front++ = 4

front = front++ % length

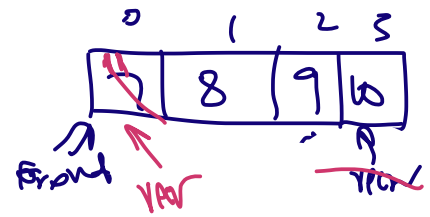
Code :

```
void addrear(int element) {
  if(rear - front == arr.length -1)
   || (front - rear == 1) {
      //Queue is full
      return;
  }

  if(front == -1) {
     front = 0;
     rear = 0;
     arr[0] = element;
     return;
  }

  rear = (rear+1) % arr.length;
  arr[rear] = element;
```

checks if full

if empty

(rear + 1)



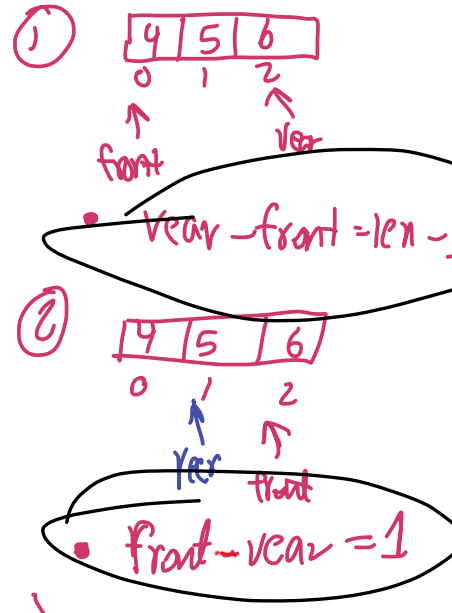
add (11)

so

rear = 4 % 4 = 0

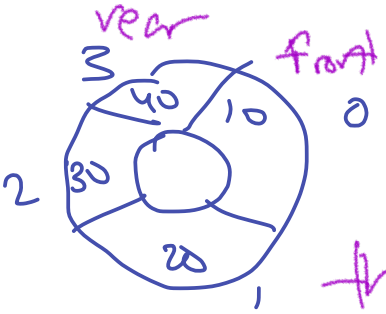
Note

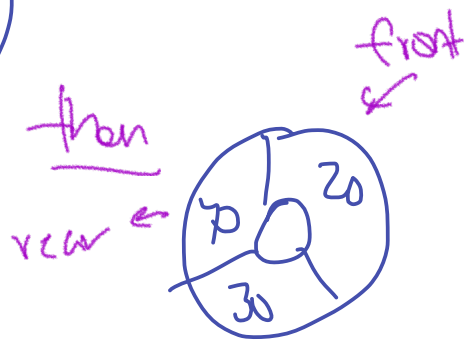
→ 2 instance when queue full:

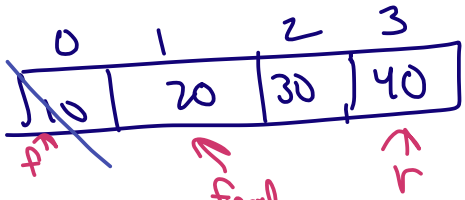


```
int deletefront() {
  if (front == -1) {
    return -1;
  }
  if (rear == front) {
    //lastelement
    int temp1 = arr[rear];
    front = -1;
    rear = -1;
    return temp;
  }
  int temp = arr[front];
  front = (front + 1) % arr.length;
  return temp;
}
```

One element left





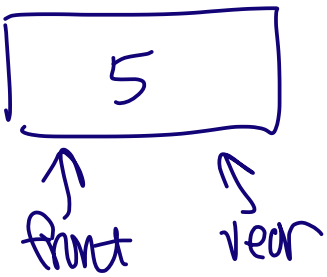


front = 0

rear = 3

size = 4

front = (0 + 1) % 5 = 1



temp = 5

front = -1

rear = -1

Convention to indicate queue is empty