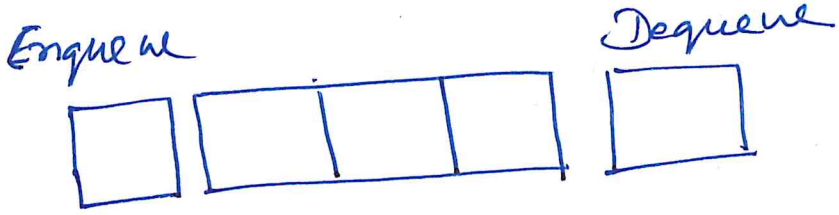
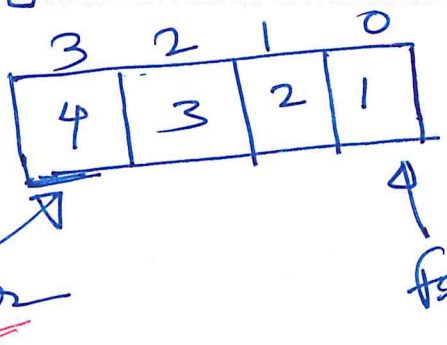
Revisiting Queue.

"Underlying concept of Queue"



(1) Array based or (2) LinkedList

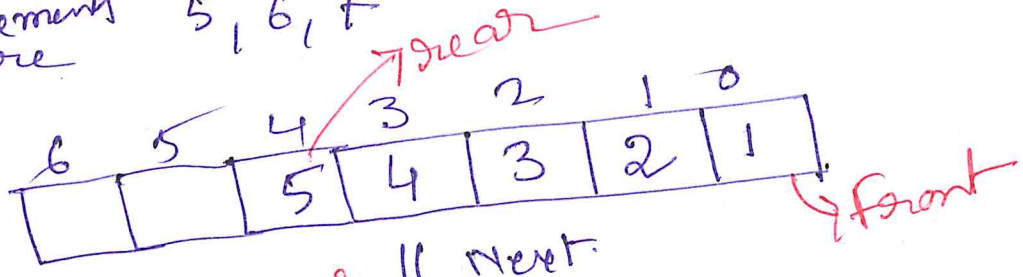


* random snap of a queue with 4 elements *

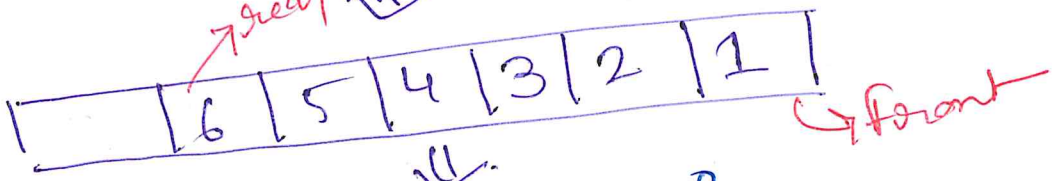
Let say queue size is (7)

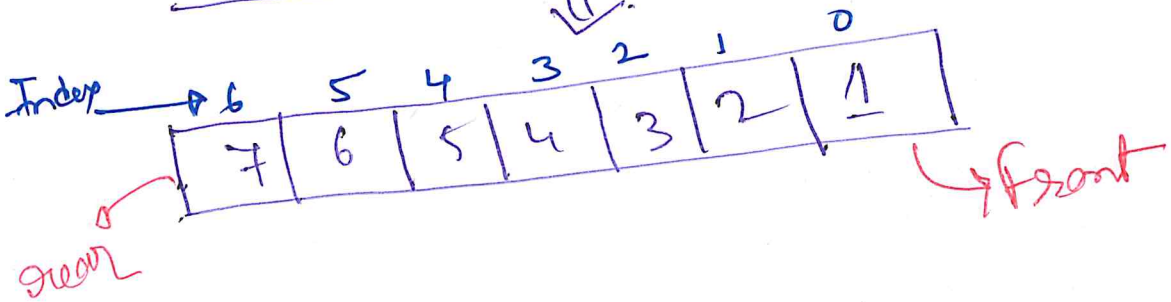
So let say, we are going to insert few more elements to the queue.

Elements are 5, 6, 7



Next



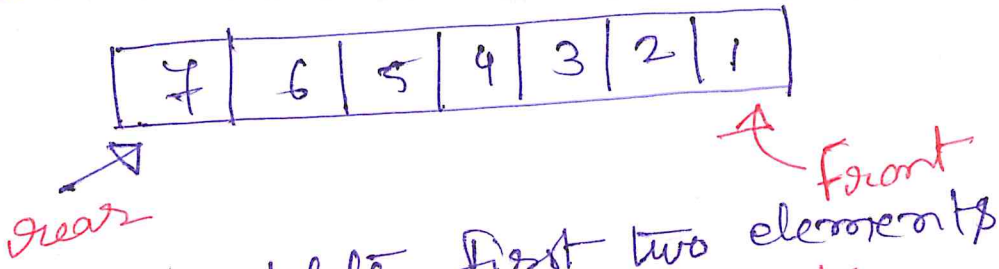


* In this queue we have consider 7 elements.

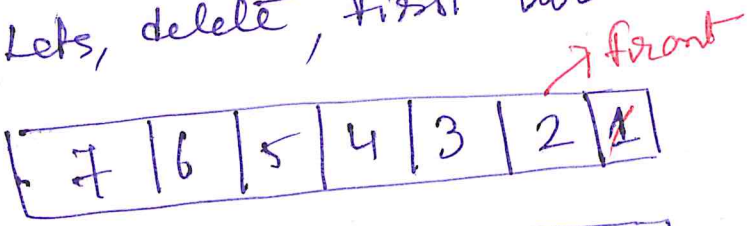
Now how to implement this queue using array, lets take array of size 7

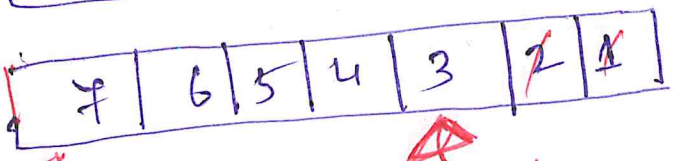
```
int queue[7];
```

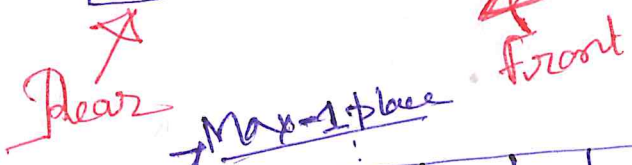
But the interesting thing is one can delete element of any time randomly

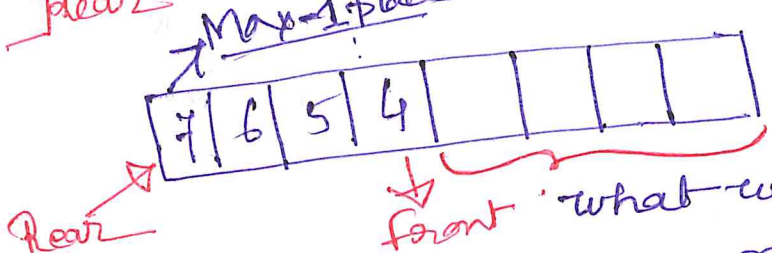


Lets, delete, first two elements









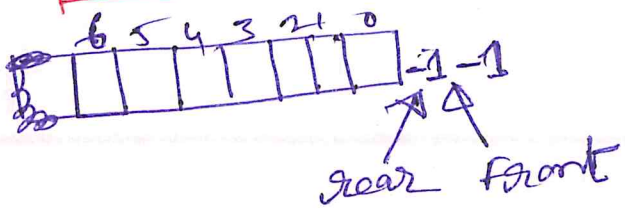
what will happen to these many cells.

So, how to insert further elements, although 4 vacant places are available

Pseudo Code

Initialization

```
int queue[7]
rear ← -1
front ← -1
```

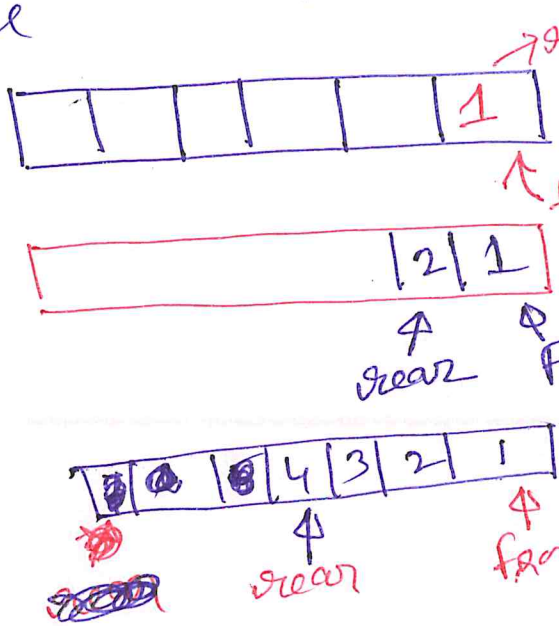



```
isempty ()
{   if (front == -1 && rear == -1)
    return true
    else
    return false
}
```

```
enqueue (element)
{   if (rear == sizeof(queue) - 1)
```

```
enqueue (element)
{ if isfull()
  return queue overflow
else if isempty()
  {  front ← rear ← 0
     queue[rear] ← element
  else { /* increment the rear by 1 i.e */
         rear ← rear + 1
       }
       queue[rear] ← element
}
```

Now, let us, perform the enque operations & dequeue operations to the queue of size



enqueue (1)

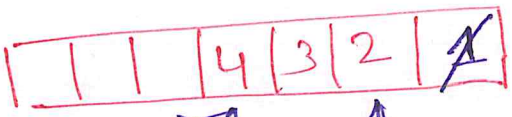
enqueue (2)

then simultaneously enque until 4

so, enqueue (3)
enqueue (4)

Now, perform 3 dequeue operations,

dequeue ()

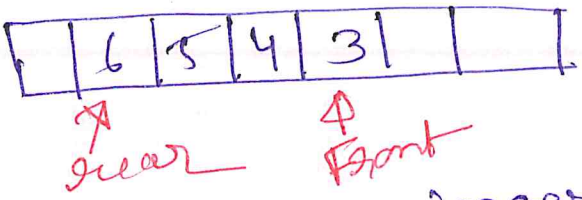


dequeue () → no parameter.

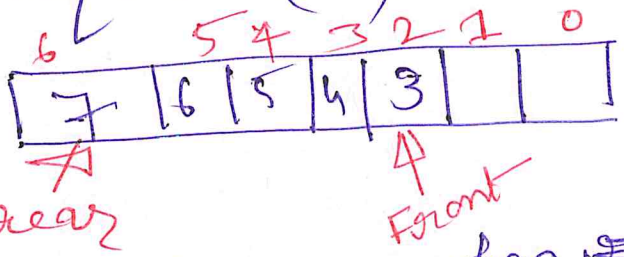
```
dequeue ()
{  if (empty ())
       print "nothing can be deleted"
   else if (front == rear)
       front = rear = -1   [goes to the square one]
   else
       front = front + 1
}
```

further insert 5, 6,
so, enqueue(5)
& enqueue(6)

then ~~qu~~ perform dequeue()



further ~~user~~ insert element, i.e
enqueue(7)

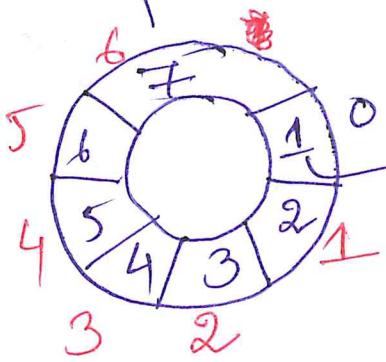


Here, rear pointer has ~~reached~~ reached maximum index i.e size of the queue i.e 7 has been reached but surprisingly two more places are still available; can we insert (i.e enqueue) further to those two places with linear queue concept.

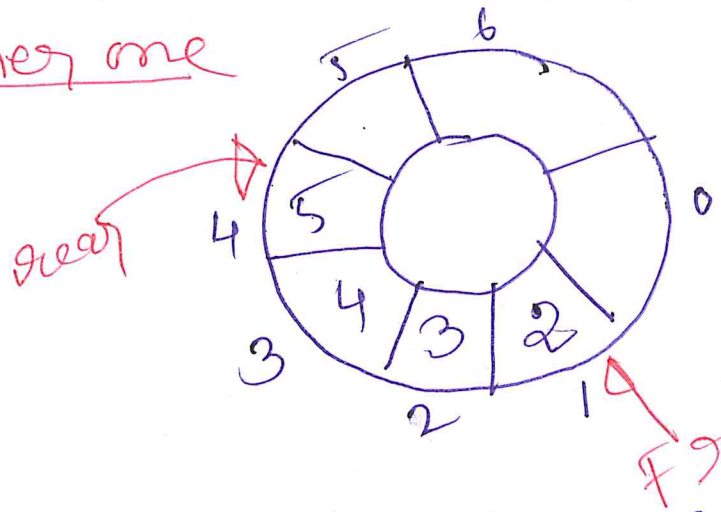
★ The answer is NO.

* We can ~~reuse~~ use those places (which were deleted already) only with the circular queue.

"Visualization of Circular Queue"



Another one



- We have to use modulus arithmatic,

if at i^{th} position already I have enqueded one element, then instead of $i+1$, the next index of the element to be enqueued will be, $(i+1)\%$ size of the queue

↳ modus.

Enqueue (element)

{ if (rear+1) % sizeof the queue == front

return

else if isempty()

{

front ← rear ← 0

}

else

{ rear ← (rear+1) % sizeof the queue

[i.e (5+1) % 7]

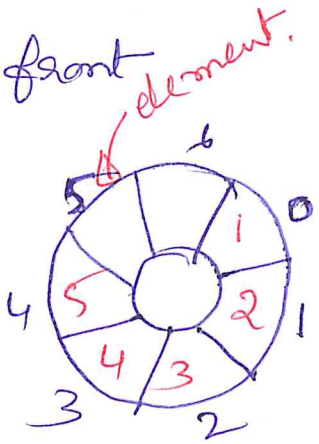
=

6

}

queue [rear] ← element

}



Let say rear ptr is at 5

Dequeue ()

{ if isempty()

return

else if front == rear

front ← rear ← -1

else front ← (front+1) % sizeof the que

}

peak () { return queue [front]

}

Queue MCQ question.

1 dequeue → takes 2 pops & 1 push.
Then we have n total enque oper's.
∴ " " deque "

∴ 1 dequeue takes 2 pops
n " " 2n pops.

You do as many ~~pushes~~ you enqueue you want ($n \leq m$) meaning as many pushes you want; but for minimum 1 dequeue or delete oper^n you must do, and that takes 2 pops; hence for n no of pops it executes 2n pops. That means minimum 2n pops you need to do under any circumstances; therefore 2n is the lower bound for pops; here y represent pops.

$$2n \leq y$$

Now, for n insert further you can add as it will add n pushes; alongside you also do 'n' delete oper^n; so n+n is the upper bound.

$$2n \leq y \leq n+n.$$