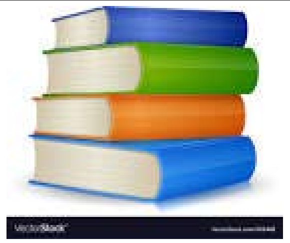
COLLECTIONS



STACKS & QUEUES

The central property of many data structures is that one element is always easiest to get at. Usually this depends on when they were put into the structure

E.G Queues at a bank, supermarket etc are FIFO. The oldest out first is a queue.
A stack of books is LIFO where the youngest is out first.

A queue supports insert(**enque**) and get oldest(**deque**).
A stack supports insert**(push)** and get youngest(**pop)**

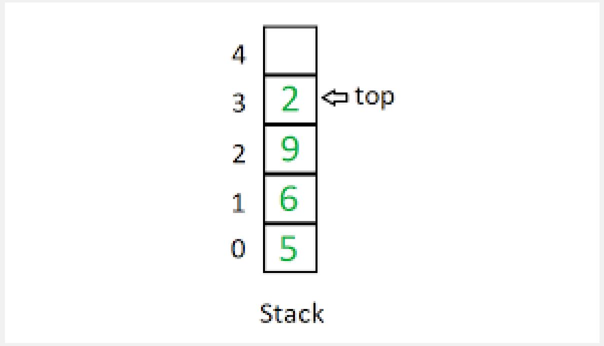
For both stacks and queues we need to know if they are full or empty. For both the exam boards like to populate via arrays but you need to know the disadvantages and that linked lists where you can efficiently insert and remove elements from both ends may be better for a queue.

Why not draw a stack or queue to help you remember the pseudocode?

STACKS –ONE TOP POINTER NEEDED PLUS ARRAY INDEX

```
stackTop =0

Push(element,stack)
       if FULL(stack)
          then
             Output "Stack full"
          else
              stackTop = stackTop +1
              stack[stackTop } = element
Pop(stack)
        if EMPTY(stack)
          then
             output "Stack is empty"
          else
              element =stack(stackTop)
              stackTop = stackTop-1
              return element
```

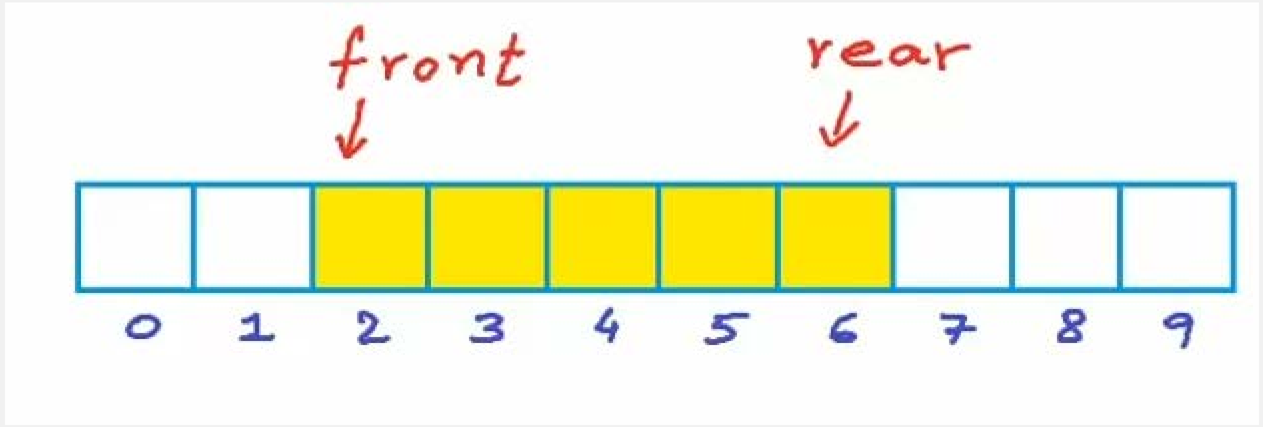
```
EMPTY(stack)
     if stackTop = 0
        then return true
        else return false
```

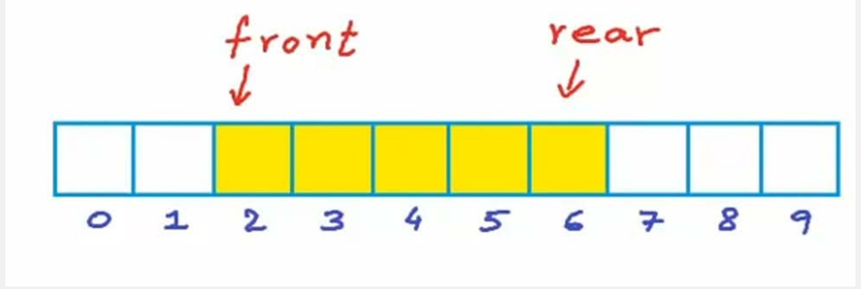


```
FULL(stack)
     if stackTop = stack,size
        then return true
        else return false
```

QUEUE -2 POINTERS NEEDED PLUS INDEX

If this is in paper 2 then there will be a scenario. So for Q for example put employees or whatever the array contains





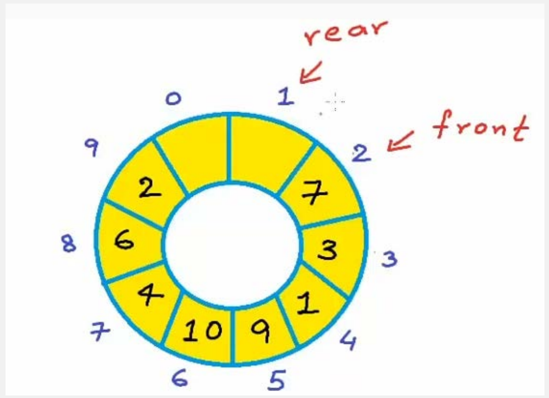
```
front, rear = -1
Enque(element,Q)
        if FULL(Q)
          then
            Output "Queue is full"
         else if EMPTY
            front = 0, rear = 0
         else
            rear = rear +1
         Q[rear] = element

Deque(Q)
         if EMPTY(q)
          then
            output "Queue is empty"
          else if front ==rear
            front = rear -1
          else  front= front+1
```

```
EMPTY(Q)
    if front & rear = -1
       then return true
       else return false
```

```
FULL(Q)
    if rear !=-1
       then return true
       else return false
```

CIRCULAR QUEUES

Using an array for a linear queue is easy to implement in code. However it does mean memory useage is inefficient as the memory location isn't reused when a deque takes place. In a normal theatre queue people shuffle up into the the space.



In a circular queue when the end is reached it tries to reuse memory by going into the first. We will need pointers for each element though and this is harder to implement, taking up space and we still have the limitations of fixed arrays.