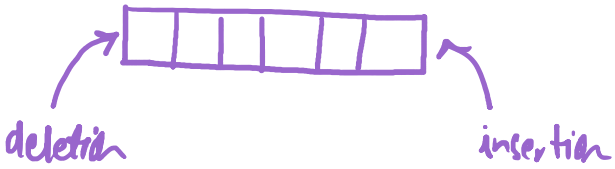
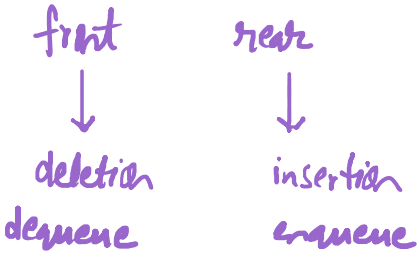
Queue

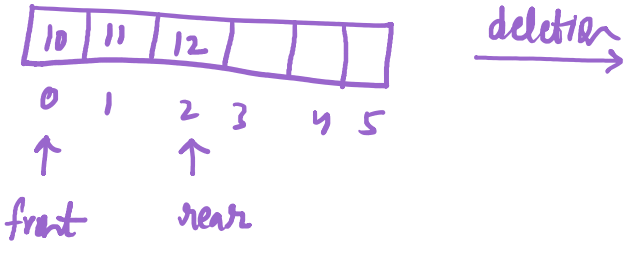
Thursday, May 6, 2021 10:00 AM

Queue :- FIFO (first in first out)

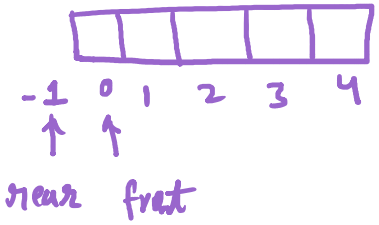
Linear Data Structure in which insertion & deletion are performed at two different ends.

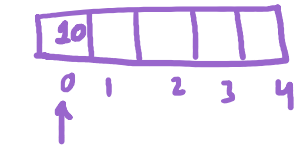




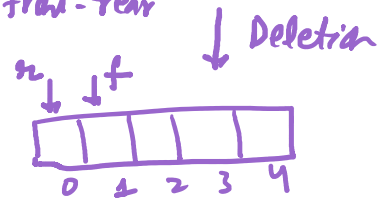


Initialization :-



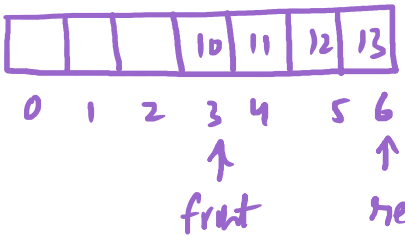


elements in queue = rear − front + 1

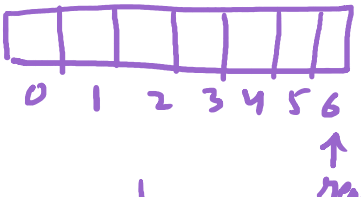



```
Insertion / Enqueue (x)

if (rear == n-1)
    Print "Overflow. Queue is full";
    return ;
q[++rear] = x ;
```

```
Deletion / Dequeue ()

if(front > rear)
    Print "Underflow. Queue is empty";
return q[front++];
```



Deletion 4 times

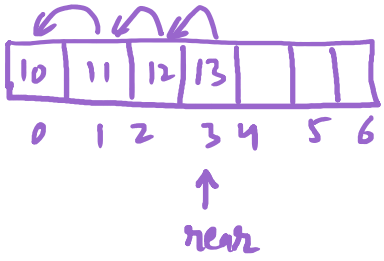


Insert(10)

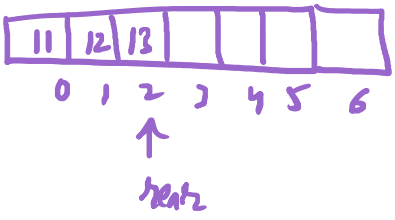
Overflow. Why ??

How to resolve this issue?

1) For deletion don't use front

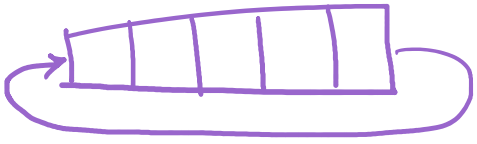


Deletion

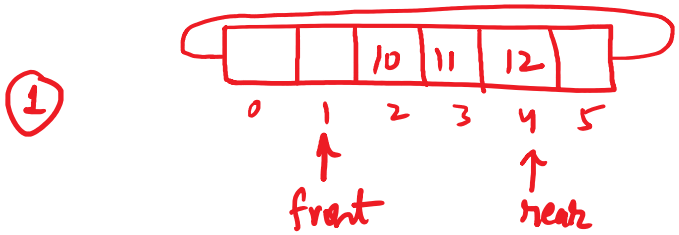


Problem: Time Complexity for deletion is O(n).

2) Moving circularly



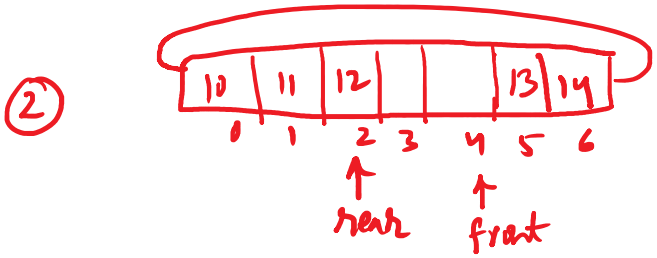
Circular Queue :-



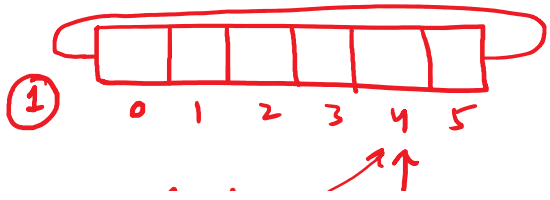
Initialization :-

front = rear = n-1

front = 0 , rear = n-1

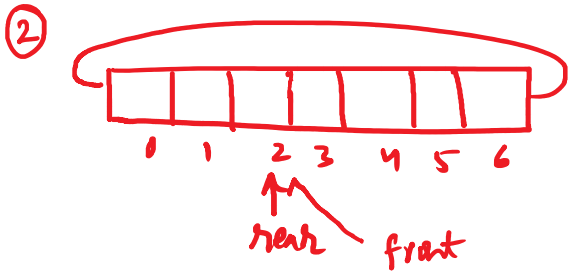


Empty Queue: if (front == rear)



Deletion :-

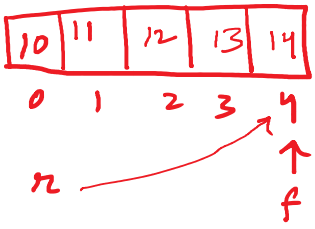
```
if (front == rear)
    print " Underflow. Queue is empty"
      return ;
return   q[ ++front % n ] ;
```



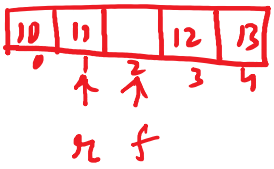
Insertion :-

```
if ( front == rear )
    print " Overflow. Queue is full"
    return;
q[ ++rear %n ] = x;
```

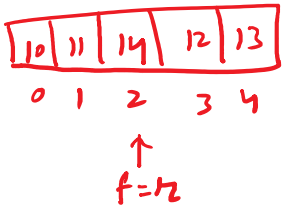
Full Queue : if (front == rear)



Problem :- Same condition for both Overflow & Underflow.



Insert (14)

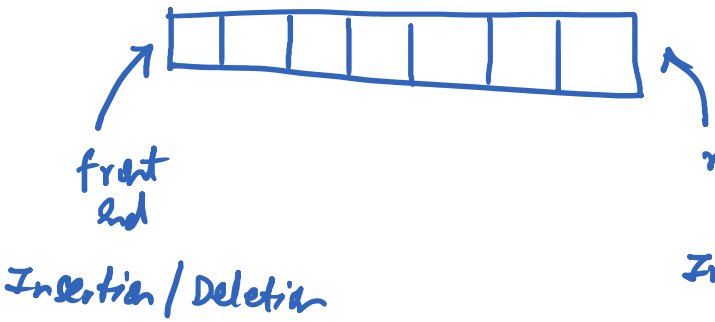


Solutions :-

1) Check the value at front or rear. If it exists then its overflow else underflow.
2) Take flag variable. Whenever we call deletion then flag = 0 else flag = 1.
   - flag = 0 ⇓ Underflow
   - flag = 1 ⇓ Overflow
3) Take a counter. Whenever we call deletion then counter -- else counter ++.
   If counter == 0 then underflow. If counter == n then overflow.

Doubly Ended Queue (DEQUE) :- Read it

Combines both stacks & Queues together.



Insert Front(x)
Insert Rear(x) ✓
Delete Front() ✓
Delete Rear()