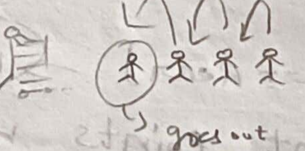
Queue ADT

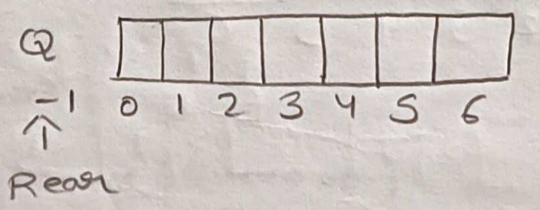
- -> FIFO
- -> tollgate FIFO
- -> Queue will have Two ends
  1. front end
  2. Rear end
- -> insertion done at Rear end
- -> deletion done at front end



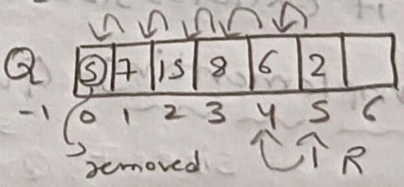
TC o(1) insert
TC o(n) deln

Queue using single Pointer

Size=7



-> To insert an element move rear to next location & insert an element.



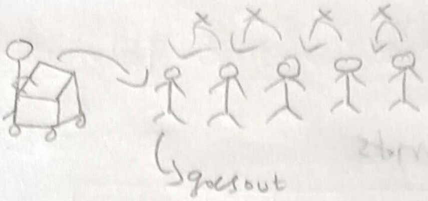
insert Time complexity -> o(1)

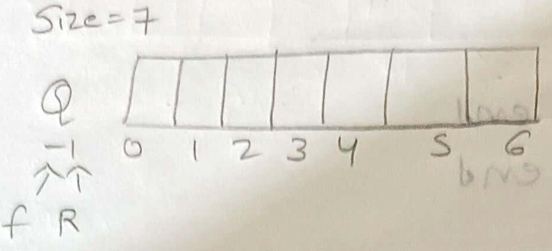
-> we should not have blank spaces in an array if we have, Then we have to check every Time whether There is an ele or it is blank. so we have to do one, some extra work. so we avoid.

-> so to occupy That Blank space all These eles Should be shifted. Then it Takes o(n) TC

Queue using Two Pointers

(idea)





-> To insert an element more Rear to next location and insert an element,

-> To delete an element more front to next location & delete an element.

-> then Here front point to before first element.

-> Both insert & deln O(1) TC

Empty

if (front== Rear) -> better

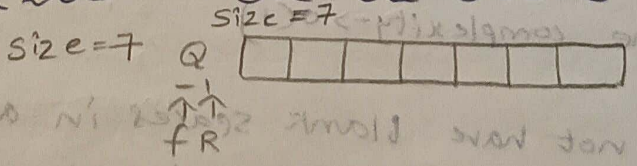
if front points not before first element points to first element Then The empty condn is

if (Rear<front)

Full

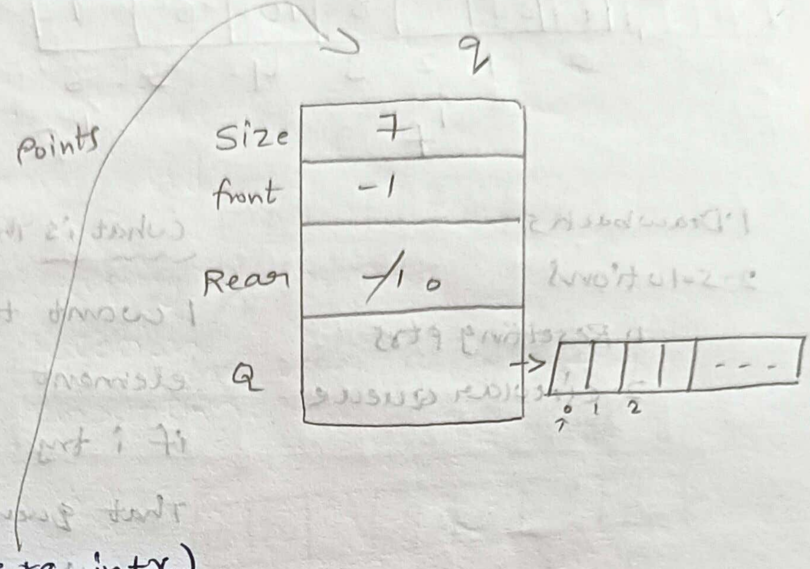
if (Rear==size-1)

Implementing Queue using array



```
struct Queue
{
 int size;
 int front;
 int Rear;
 int *Q;
};
```

```
int main()
{
  struct Queue q;
  printf("Enter size");
  Scanf("%d", &q.size);
  q.Q = (int *) malloc (q.size * sizeof(int));
  q.front=-1;
  q.Rear=-1;
```



enqueue

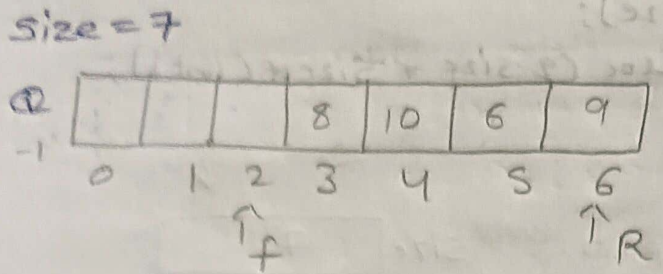
```
void enqueue(Queue *q, int x)
{
    if(q->Rear==q->size-1)
        printf("Queue is Full");
    else
    {
        q->Rear++;
        q->Q[q->Rear]=x;
    }
}
```

dequeue

```
void dequeue (Queue *q )
{
    int x=-1;
    if(q->front==q->Rear)
        printf(" Queue is Empty");
    else
    {
        q->front++;
        x=q->Q[q->front];
    }
    return x;
}
```

Code Queue using array pdf ✓

Drawback of Queue using Array :

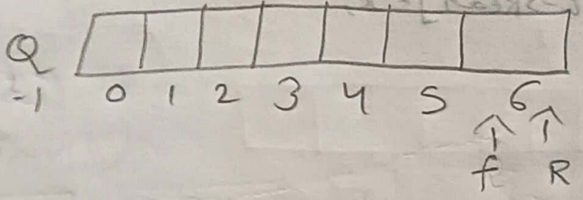


1. Drawbacks
2. Solutions
    1. Resetting ptrs
    2. circular queue

What is The drawback here?

I want to insert a new element in The queue. if i try to insert i get a msg That queue is full, but at starting some space is There but i can't use it bcz insertion is done from rear end.

1. we cannot reuse The The spaces of deleted element.

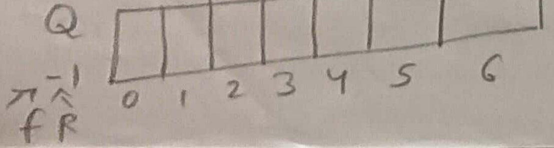


⇒ queue is empty after remove all

Resetting pointers

At any time if a queue is becoming empty. bring front & rear pointer at The beginning, That is re-initialize Them to minus one, so That They can again start from The beginning. So in This way we can reuse Those places.

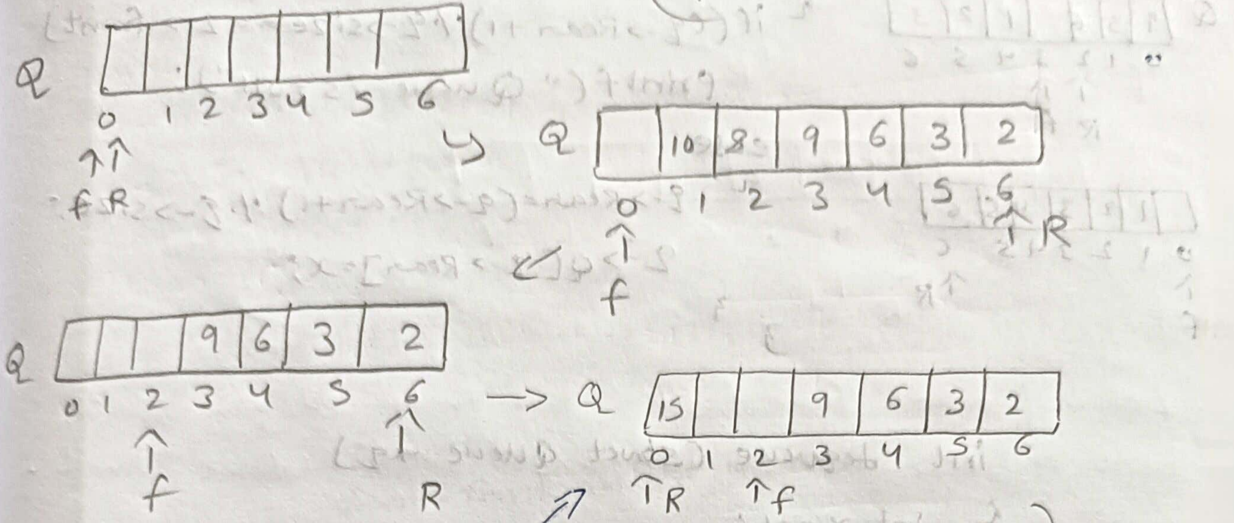
(Not at end only whenever front & rear are becoming equal).



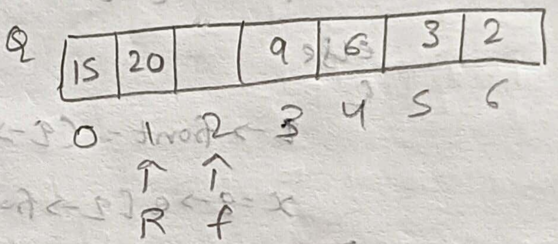
Circular Queue

Size=7

(Now from first onwards f & R at 0)



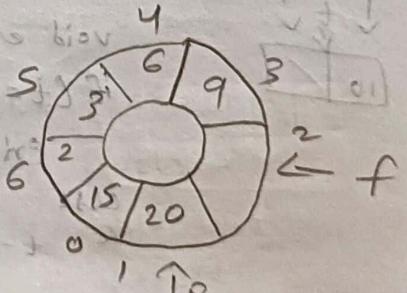
-> I want to insert. then bring rear at zero

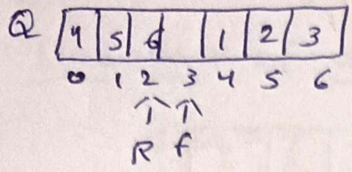


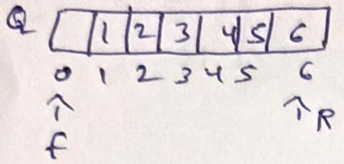
-> & The Question is can I insert one more element? no There's free space, can't i use? Don't use That space. wherever front is pointing, That space must be left emety.

what happens if I want to use That space? So if U bring rear There. Then The rear & front, both are equal. when equal queue is empty.

Rear	(Rear+1)%size
0	(0+1)%7 = 1
1	(1+1)%7 = 2
2	3%7 = 3
3	4%7 = 4
4	5%7 = 5
5	6%7 = 6
6	7%7 = 0
0	





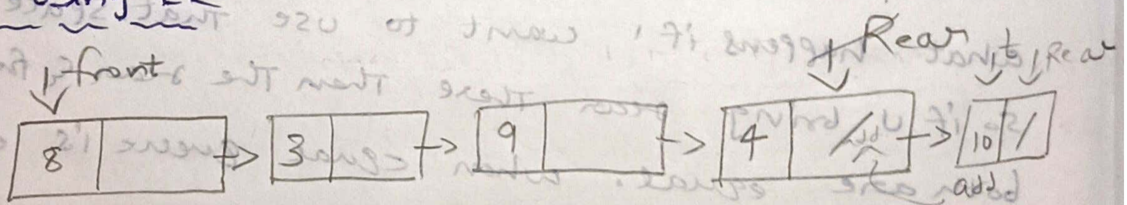


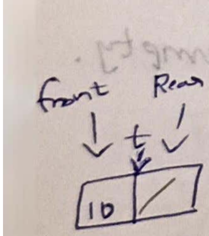
```
void enque (struct Queue *q, int x)
{ if ((q->Rear +1)%q->size == q-> front)
      printf(" Queue is Full");
  else
  {
      q->Rear=(q->Rear+1) %q->size;
      q->Q[q-> Rear]=x;
  }
}
```

```
int dequeue (struct Queue *q)
{  int x=-1;
   if (q-> front==q->Rear)
       printf("Queue is Empty");
   else
   {
       q-> front= (q-> front+1) % q->size;
       x=q-> Q [q-> front];
   }
   return x;
}
```

Code circular queue pdf ✓

Queue using LL





```
void enqueue (int x)
if (t==NULL)
    printf("Queue is Full");
else
{
    t->data=x;
    t->next=NULL;
    if (front==NULL) front=Rear=t;
    else
    {
      Rear-> next=t;
      Rear=t;
    }
}
```

definitely it becomes last node so..

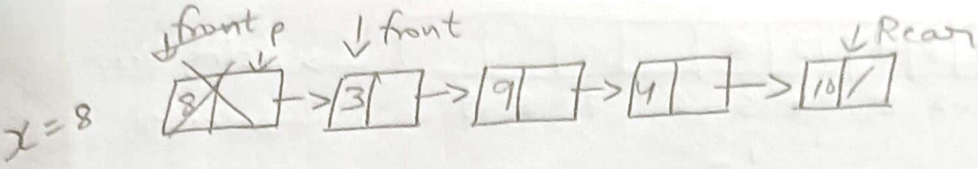
Empty

if (front==NULL)

FULL

Node *t=new Node;

if (t== NULL).



```
int dequeue()
{ int x=-1;
  Node *p;
  if(front==NULL)
   printf("Queue is Empty");
  else
  {
     p=front;
     front=front->next;
     x=p->data;
     free(p);
  }
  return x;
}
```

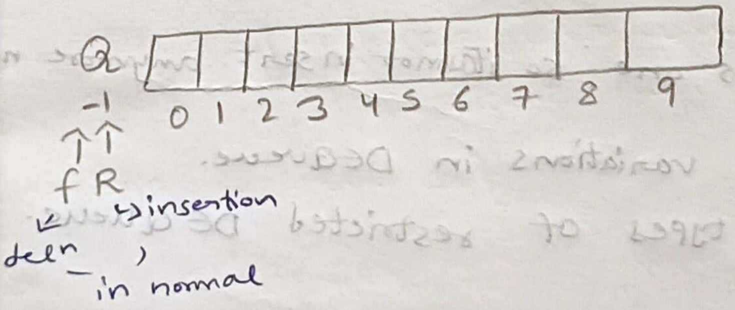
code Queue using LL pdf✓

Double Ended Queue (DEQueue)

-> it doesn't strictly follow FIFO

-> if u want u can use it as a FIFO.

-> it can be implemented using array & LLs.



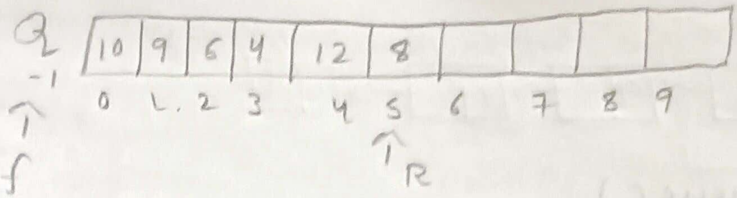
-> But in DEQueue u can use both the ptrs for both operations. insertn as well as deln: with rear ptr we can inert as well as delete. And with front ptr we can insert as well as delete.

Queue

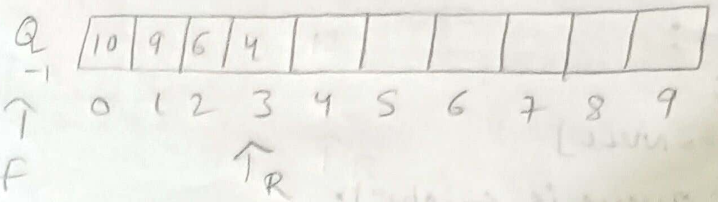
	Insert	delete
front	X	✓
Rear	✓	X

DEqueue

	insert	dele
fro	✓	✓
re	✓	✓

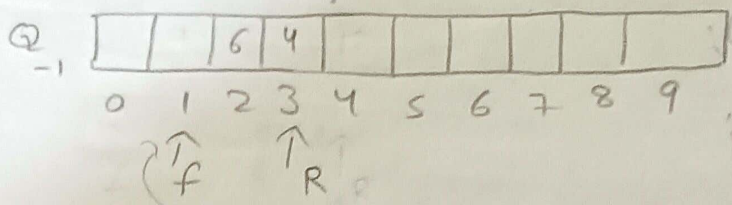


-> using rear inserted

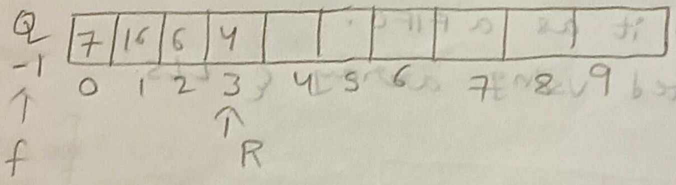


-> using rear deleted

Let us insert from front, but There is no space here to insert. so i cannot insert from front. Can I delete? Yes.



Now, want to insert using front. insert The value here at front for this pointer, & decrement front.



front is at minus one so itcannot insert anymore now.

-> There are Some variations in DEQueue.

There are Two types of restricted DE Queue.

insert

i/p Restricted DE Queue -> if U want U can use This type of data structure also.

	Inse	Dele
f	×	✓
R	✓	✓

delete

o/p Restricted DE Queue

	In	De
f	✓	✓
R	✓	×

Priority Queues

There are Two methods of Implementing priority queues depending on The situations.

1. limited set of Priorities
2. Element priority

1st Method:- This method is mostly useful in operating system, like in Some O.S allows priority based scheduling java jvm supports multithreading so its allow priorities upon threads so u can set the priorities for Threads & java supports priorities from 1 to 10, so higher priority threads will execute first. (10↑, 1↓)

priorities = 3

Element ->	A	B	C	D	E	F	G	H	I	J	K	L
priority ->	1	1	2	3	2	1	2	3	2	2	1	1

(in this ex we are assuming as 1↑ 3↓)

Priority Queues

Q1	~~A~~ ~~L~~	~~B~~	~~F~~	~~K~~		
Q2	C	E	G	I	J	
Q3	D	H				

-> when ever we are deleting we must delete Highest priority queue. & strictly FIFO

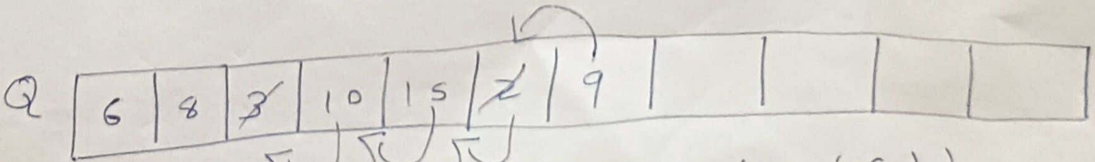
2nd Method:-

Elements -> 6, 8, 3, 10, 15, 2, 9, 17, 5 (where, The ele itself is a priority)

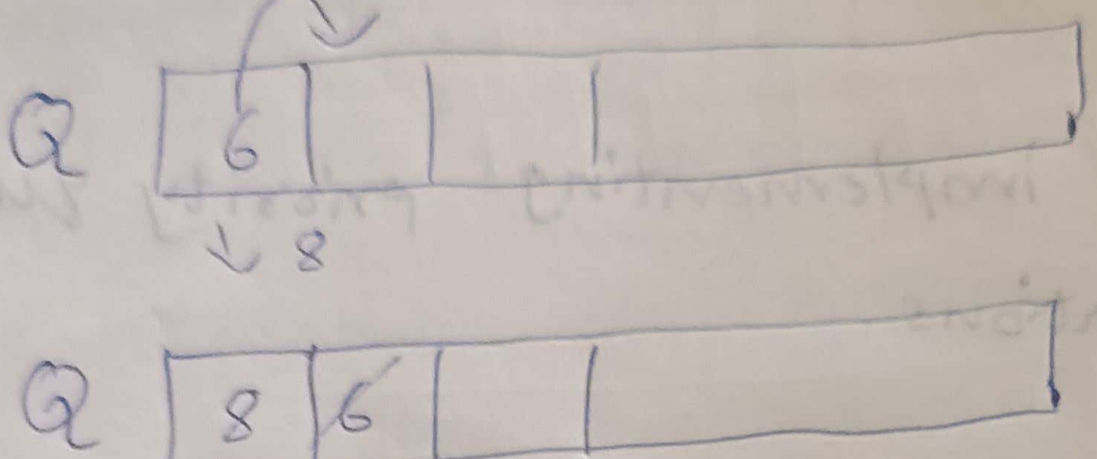
Smaller number Higher priority (we can also change this like= ↑Num ↑prio)

Q									

1. insert in same order
   Delete Max prio by searching it
2. insert in ↑ing order of priority
   Delete last ele of Array.



ins (O(1))
Del (O(n))



ins o(n)

del o(1)