CS & IT ENGINEERING

Data Structures

Stack and Queues

Lecture No.- 05

By- Pankaj Sharma Sir

Recap of Previous Lecture

Topic Stack and Queues Part - 04

Practice Questions, PYQs on stack

Topics to be Covered

Topic Stack and Queues Part - 05

Queue data structure

Topic : Stack and Queues

Queue

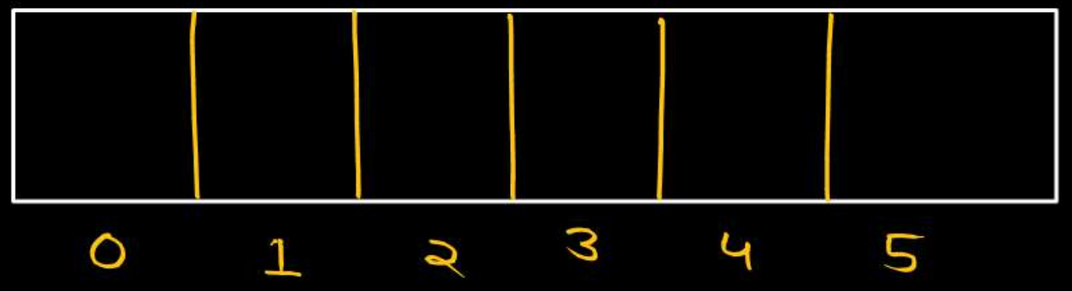
* Linear data structure
* First in First out
* Last – in Last out

Insertion : Rear

Deletion : Front

Array implementation

```
#define SIZE 6
int Queue[SIZE];
int Front = -1;
int Rear = -1
```



Rear : The index of most recently added element.

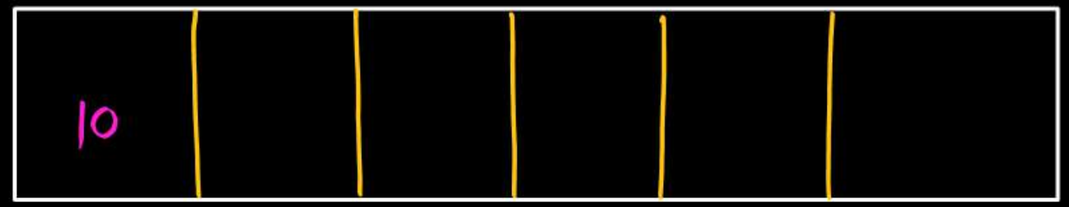
Front : The index of element that can be deleted.

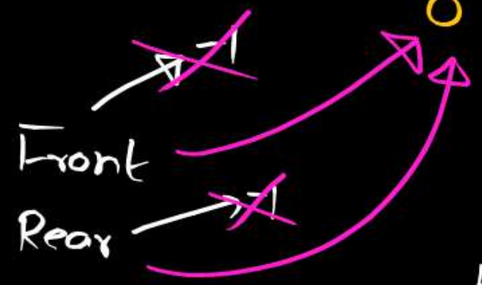
① Initially, when Queue is Empty.

Front = Rear = -1

Front == Rear == -1

Empty





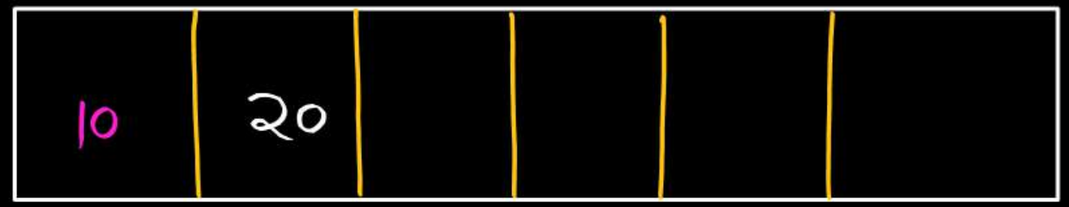
	Front	Rear
Initially	-1	-1
Insert(10)	0	0

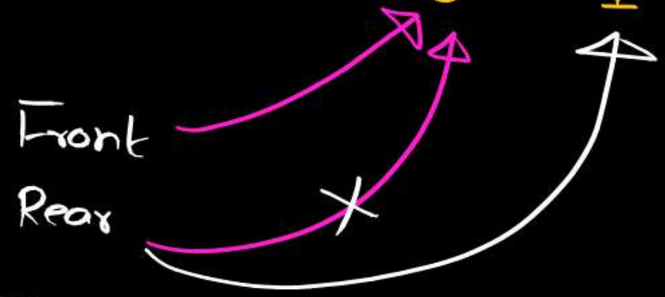
Front = 0 ⇒ index 0 element Can be deleted from Queue.

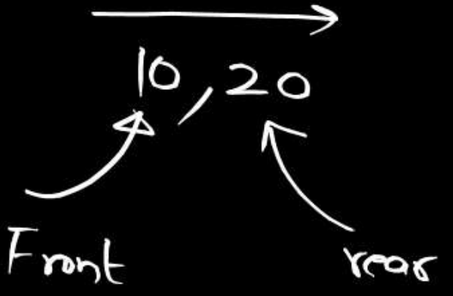
Rear = 0 ⇒ index 0 element is the most recently added element.

Front == Rear == -1

Empty



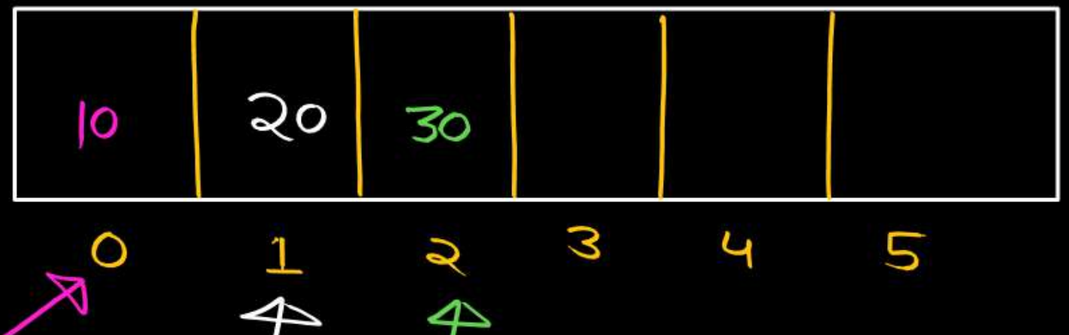




	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)		

Front == Rear == -1

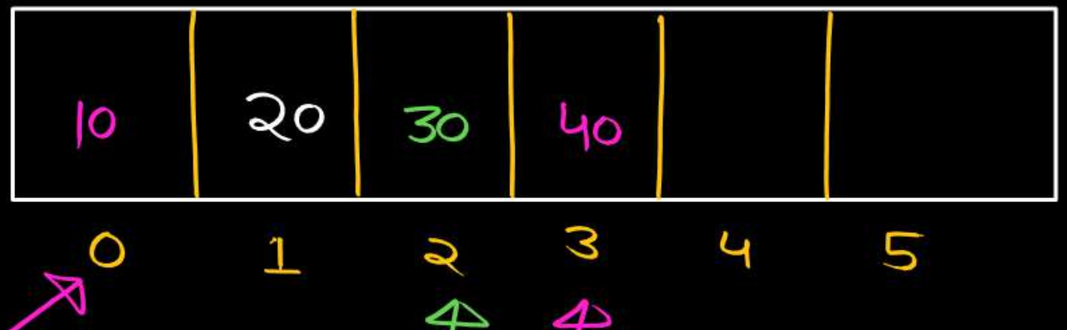
Empty



	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2

Front == Rear == -1

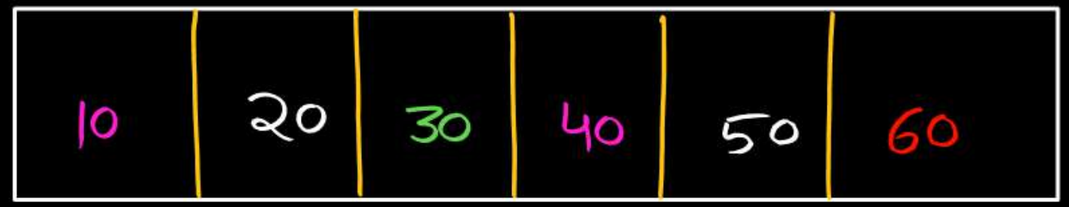
Empty

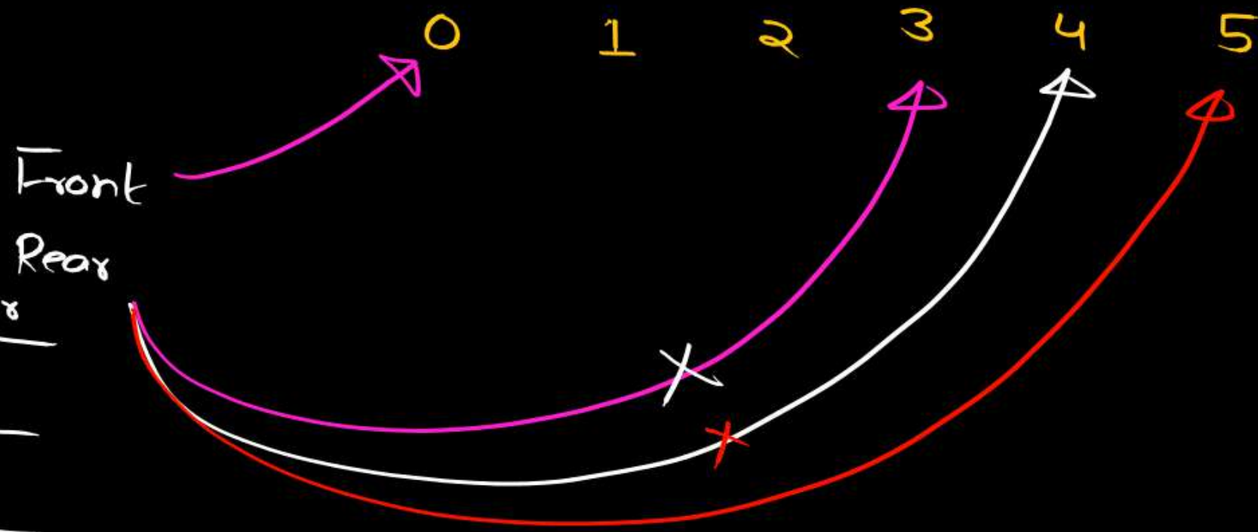


	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3

Front == Rear == -1

Empty



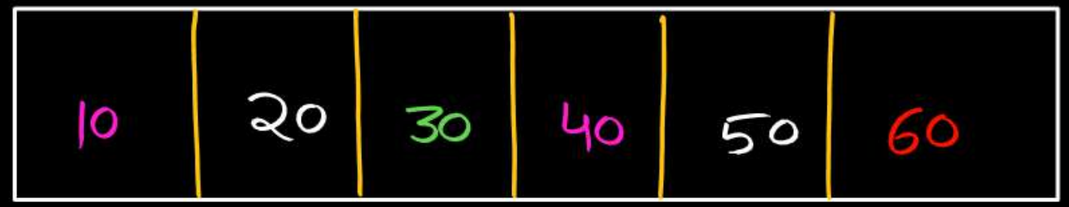


	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

Front == Rear == -1

Empty

Simple Queue



Front

Rear

	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

	Front	Rear
Insert(60)	0	5
Delete		

Insert(70)

↓

Overflow

if (Rear == SIZE-1)

Overflow

Front == Rear == -1
Empty

Simple Queue

~~10~~, 20, 30, 40, 50, 60

~~10~~	20	30	40	50	60
0	1	2	3	4	5

Front

Rear

	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

	Front	Rear
Insert(60)	0	5
Delete	1	5

Insert(70)

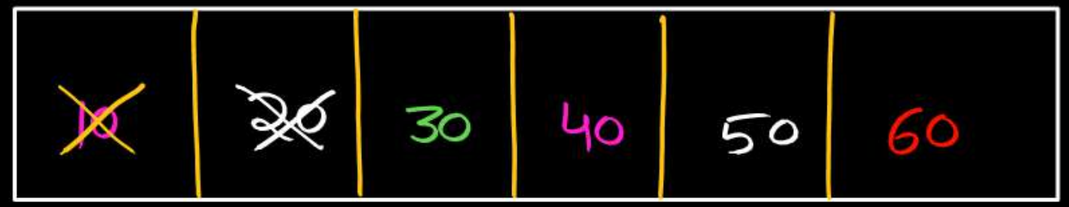
↓

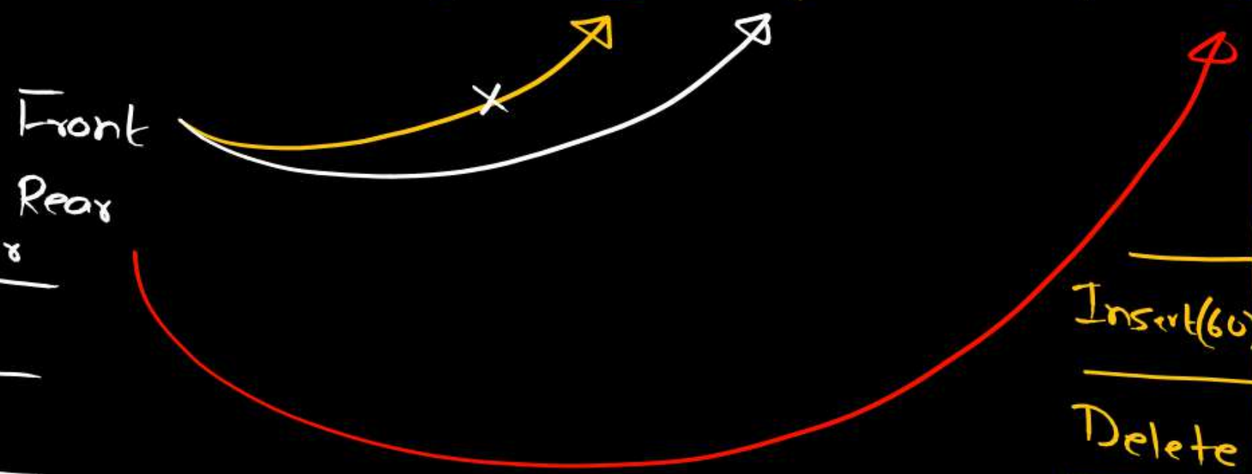
Overflow

if (Rear == SIZE-1)
Overflow

Front == Rear == -1
Empty

Simple Queue 10, 20, 30, 40, 50, 60





	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

Insert(70)

↓

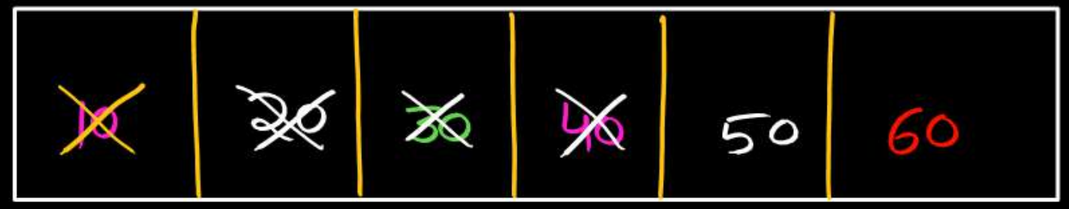
Overflow

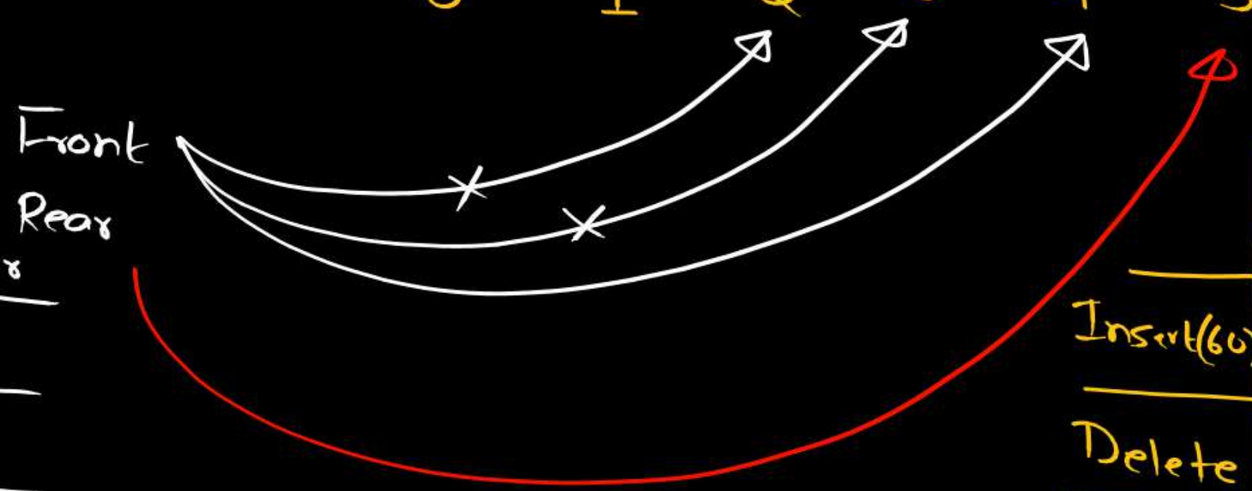
if (Rear == SIZE-1)
Overflow

	Front	Rear
Insert(60)	0	5
Delete	1	5
Delete	2	5

Front == Rear == -1
Empty

Simple Queue 10, 20, 30, 40, 50, 60





	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

Insert(70)

↓

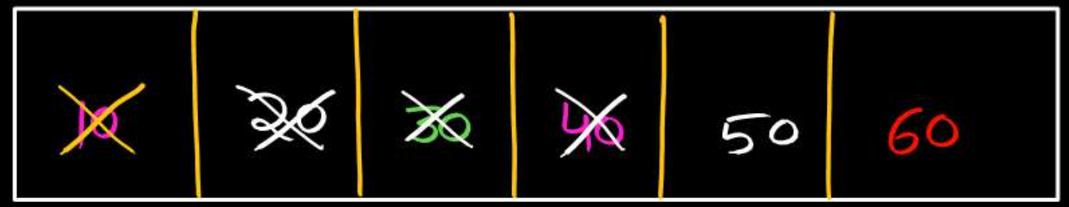
Overflow

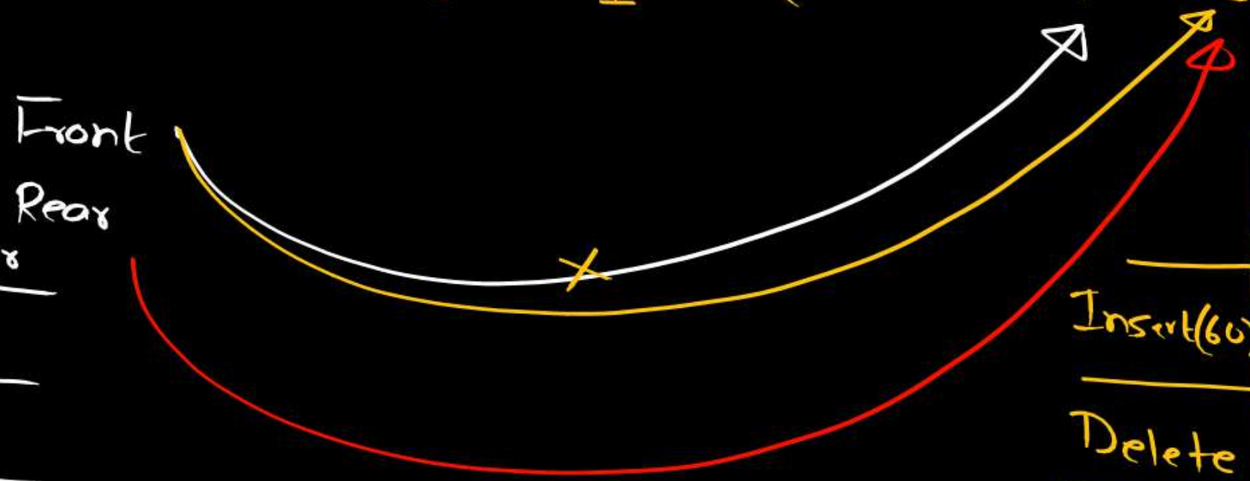
if (Rear == SIZE-1)
Overflow

	Front	Rear
Insert(60)	0	5
Delete	1	5
Delete	2	5
Delete	3	5
Delete	4	5
Delete		

Front == Rear == -1
Empty

Simple Queue 10, 20, 30, 40, 50, 60





	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

Insert(70)

↓

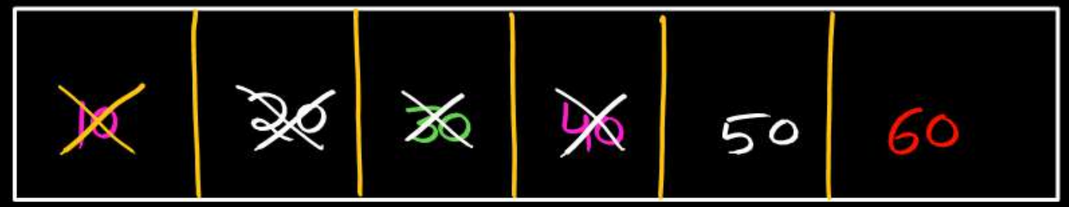
Overflow

if (Rear == SIZE-1)
Overflow

	Front	Rear
Insert(60)	0	5
Delete	1	5
Delete	2	5
Delete	3	5
Delete	4	5
Delete	5	5

Front == Rear == -1
Empty

Simple Queue 10, 20, 30, 40, 50, 60



Front
Rear

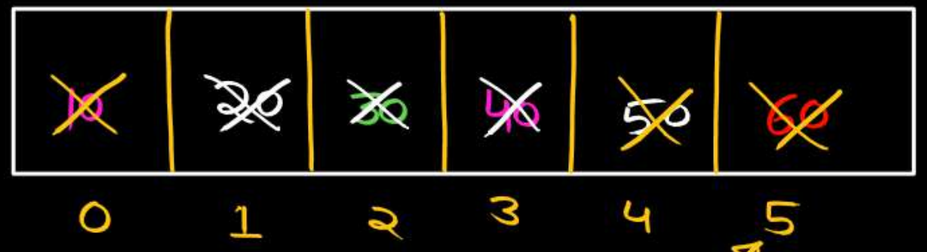
	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

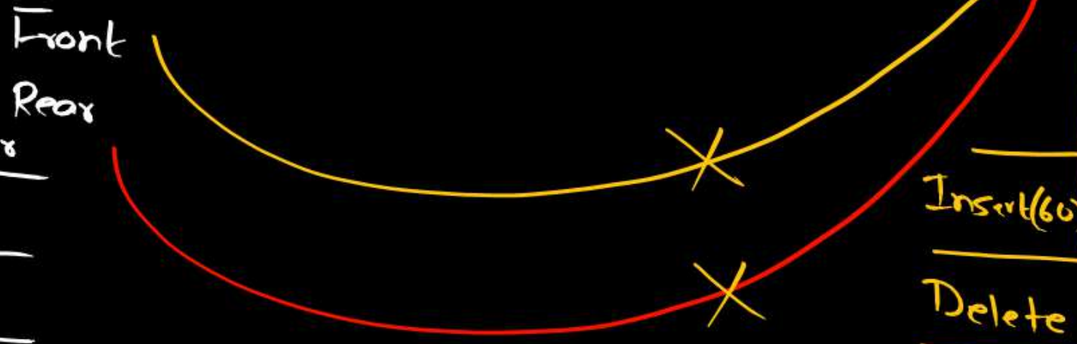
bcz there is only 1 element

	Front	Rear
Insert(60)	0	5
Delete	1	5
Delete	2	5
Delete	3	5
Delete	4	5
Delete	5	5

Front == Rear == -1
Empty

Simple Queue 10, 20, 30, 40, 50, 60





	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

1 element in Queue

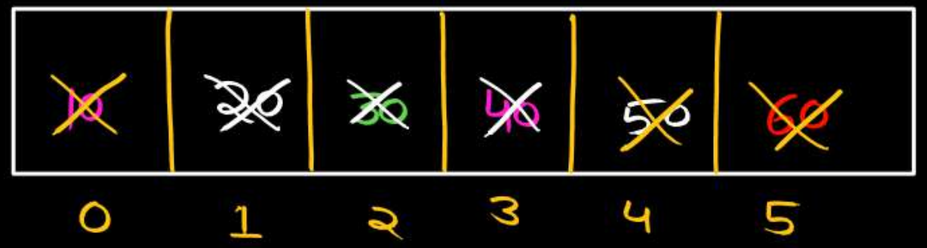
↓ Delete

Queue become Empty
(Rear, Front both to -1)

	Front	Rear
Insert(60)	0	5
Delete	1	5
Delete	2	5
Delete	3	5
Delete	4	5
Delete	5	5
Delete		

Front == Rear == -1
Empty

Simple Queue 10, 20, 30, 40, 50, 60



Avijit

Front = -1
Rear = -1

	Front	Rear
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4
Insert(60)	0	5

1 element in Queue

↓ Delete

Queue become Empty
(Rear, Front both to -1)

	Front	Rear
Insert(60)	0	5
Delete	1	5
Delete	2	5
Delete	3	5
Delete	4	5
Delete	5	5
Delete	-1	-1

```
void Enqueue( int key){

    if(Rear == SIZE -1)

            return;

    else if( Front == -1 )
        {
        Rear = Front = 0;
        Queue[Rear] = key;
        }
    else {
            Rear ++ ;
        Queue[Rear] = key;
        }

            }
```

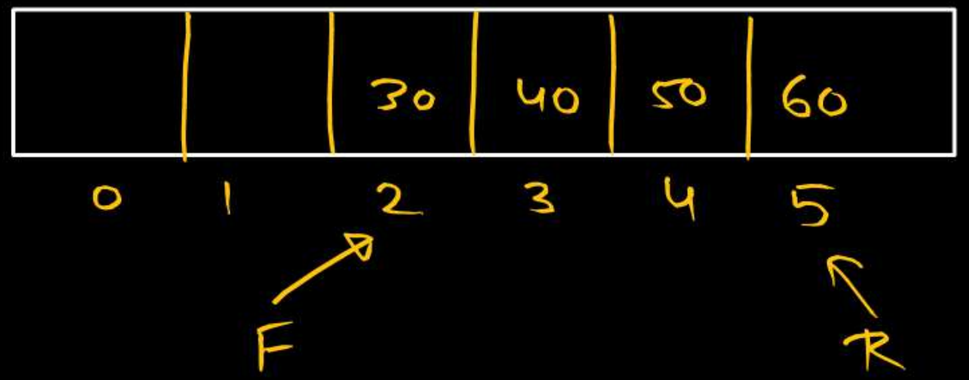
Constant time
→ O(1)

```
int Dequeue ( ){  int temp;
    if(Front == -1)
        return INT_MIN;
    else if( Front == Rear )
        {
            temp = Queue[Front];
            Front = Rear = -1;
        }
    else {
            temp = Queue[Front];
            Front ++;
        }
    return temp;
    }
```

constant time

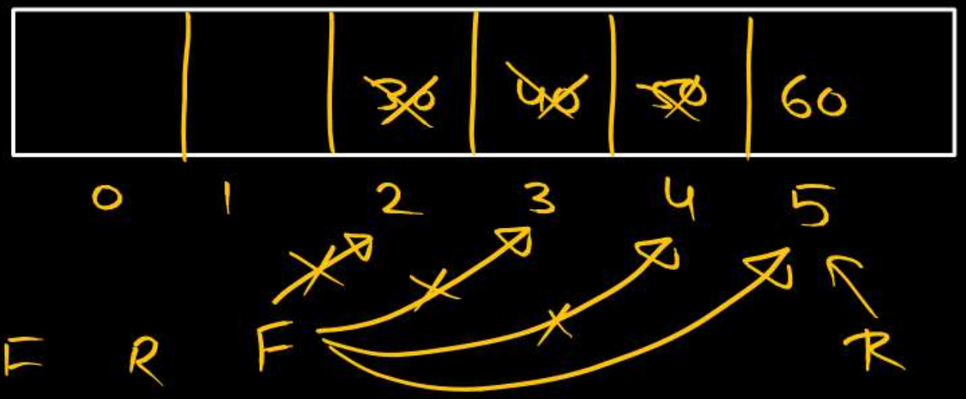
Simple Queue

Simple Queue



Insert (70) → overflow

Simple Queue



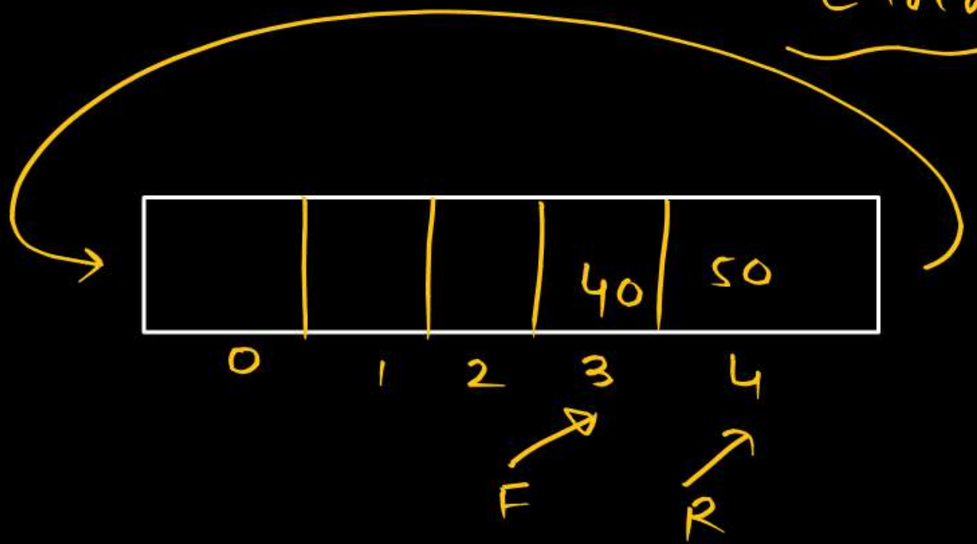
	F	R
Delete	3	5
Delete	4	5
Delete	5	5

Simple Queue

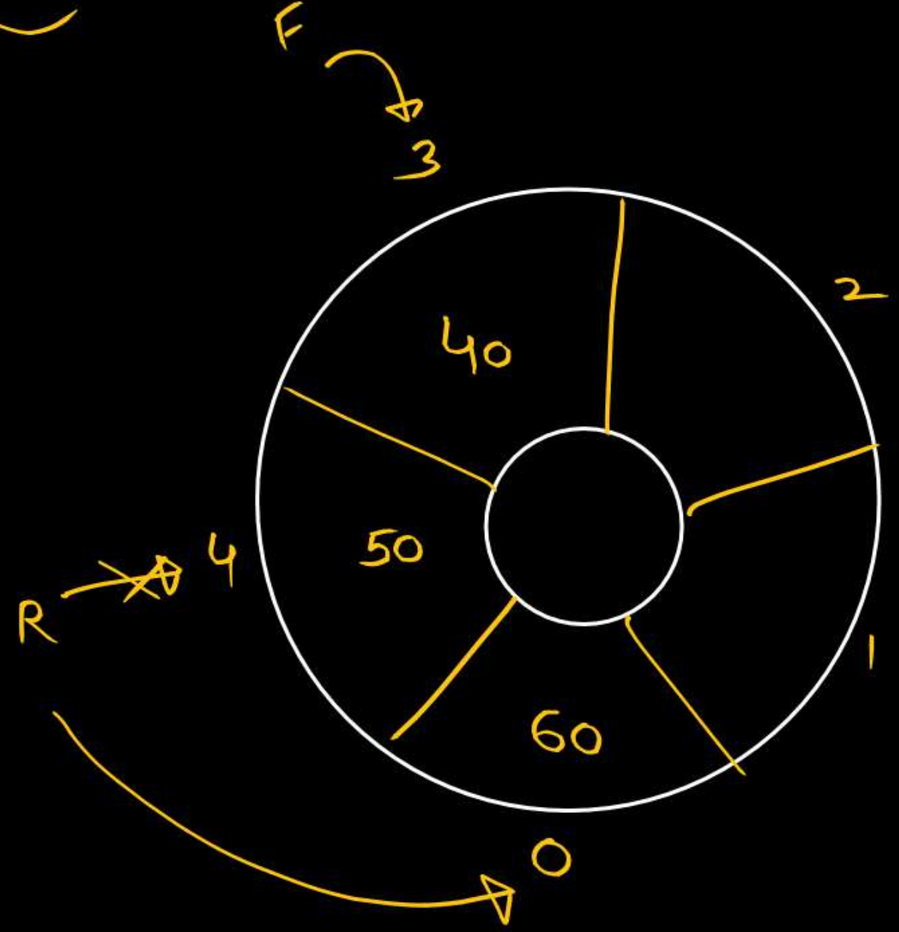
	F	R
Delete	3	5
Delete	4	5
Delete	5	5
Insert(70)	→ overflow	

Disadvantage of Simple Queue

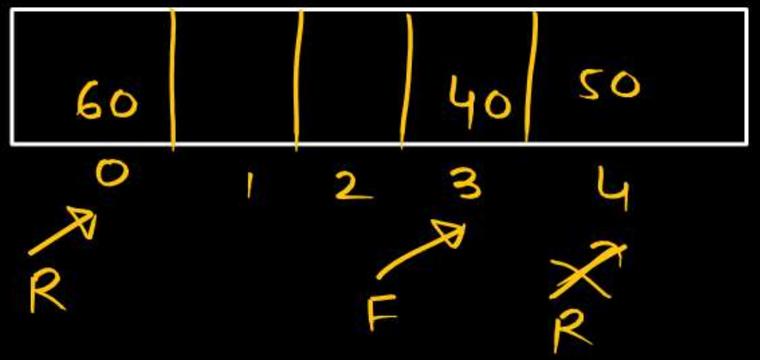
Circular Queue



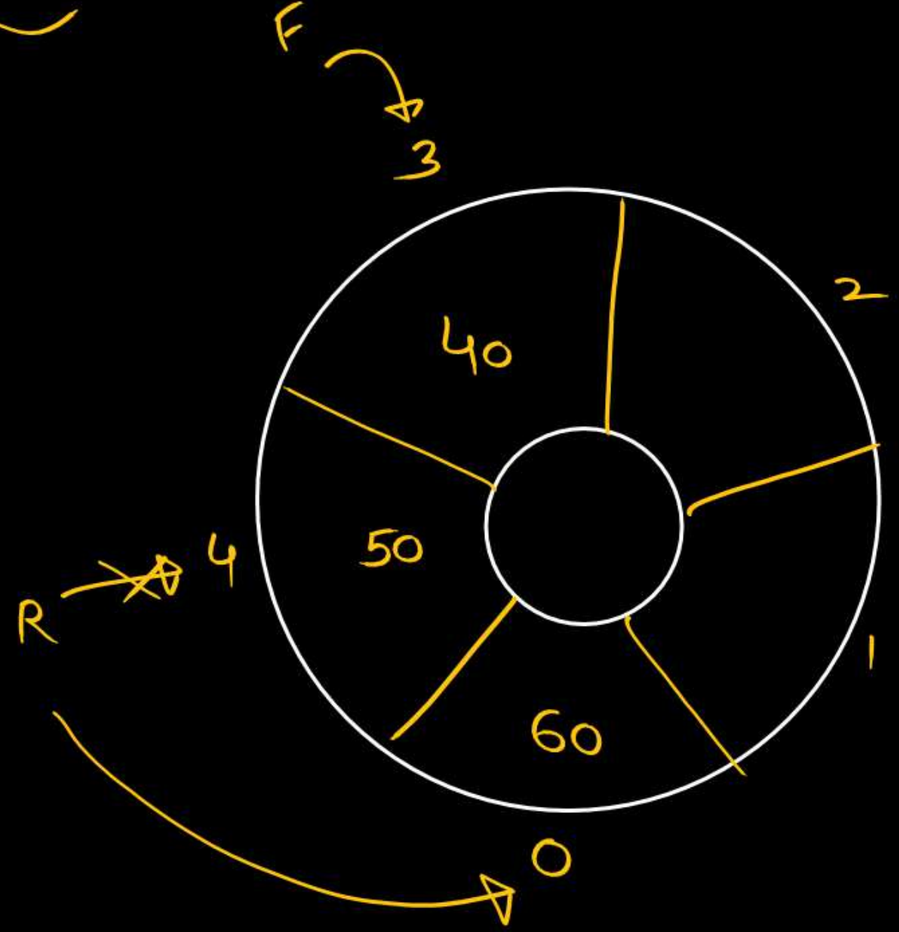
	F	R
Insert 60	3	0



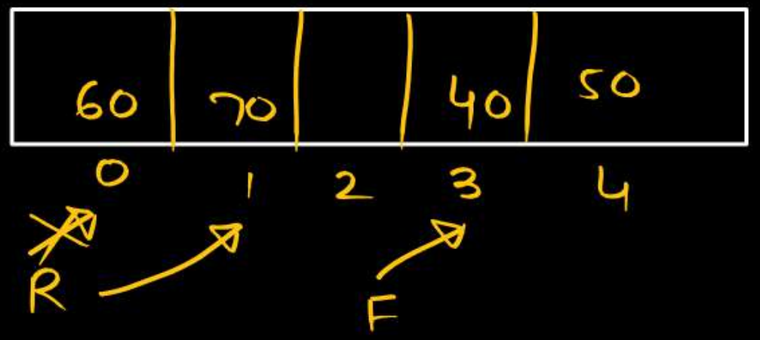
Circular Queue



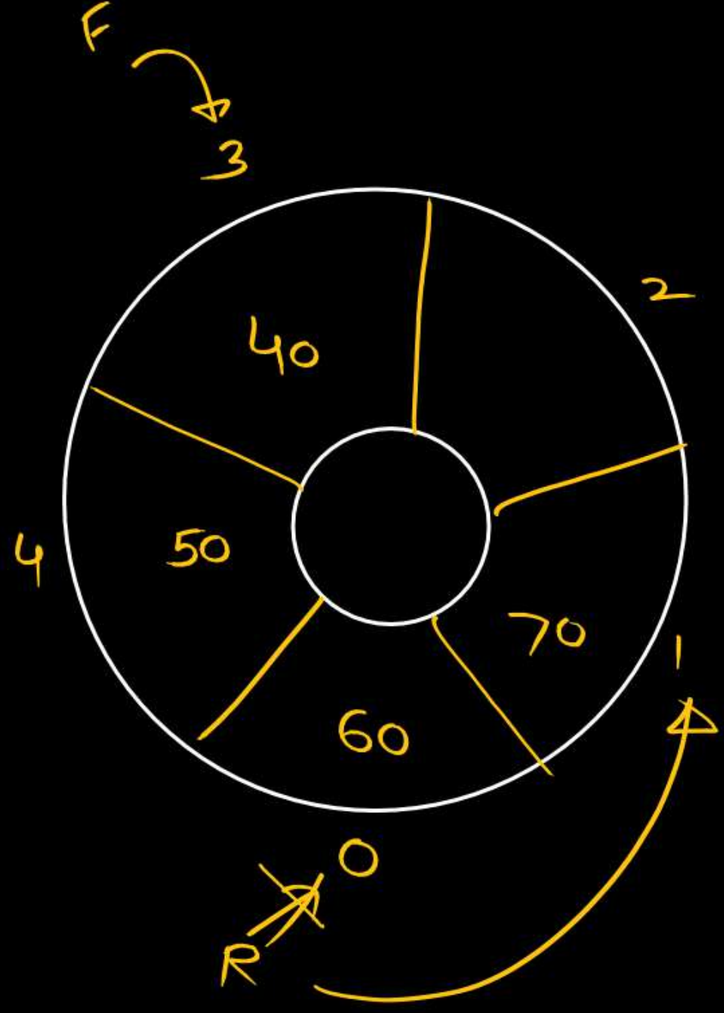
Insert 60	F	R
	3	0



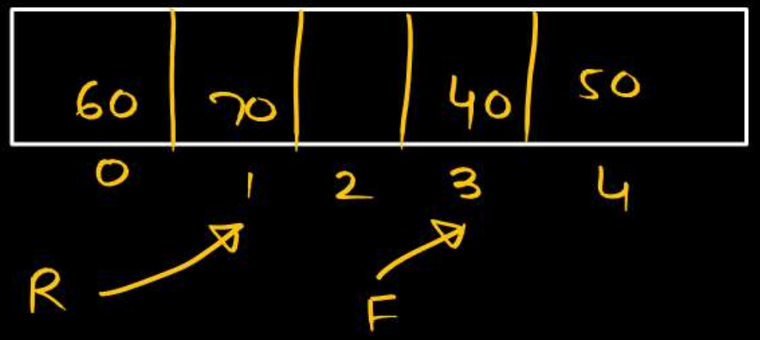
Circular Queue

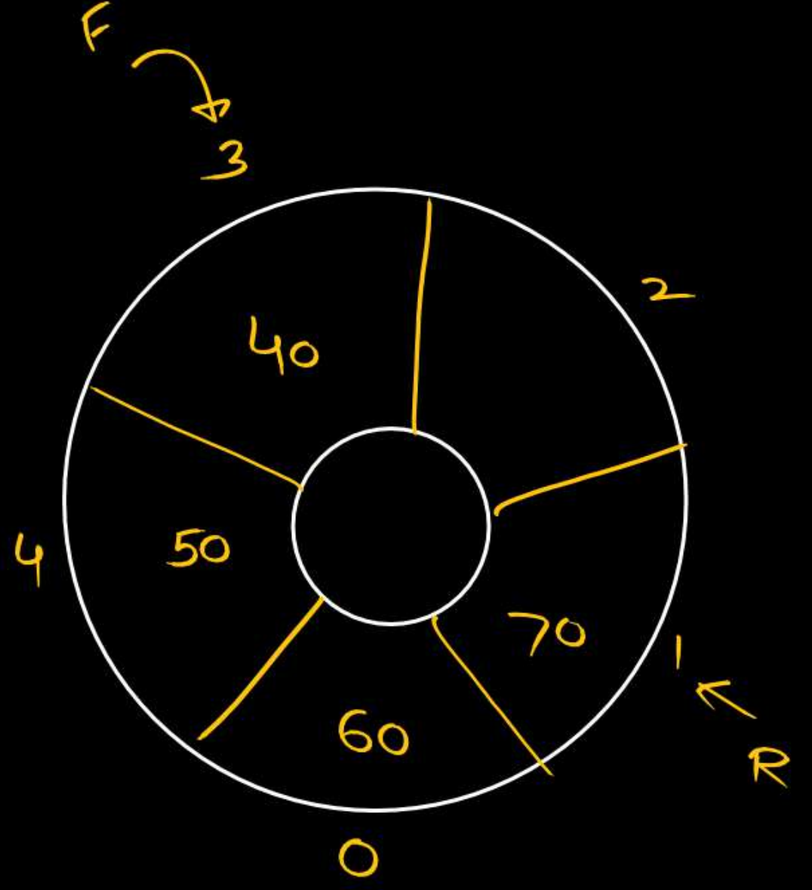


	F	R
Insert 60	3	0
Insert 70	3	1



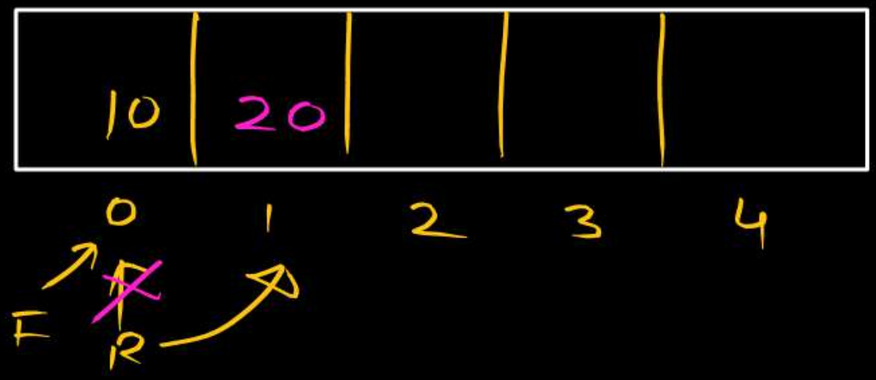
Circular Queue



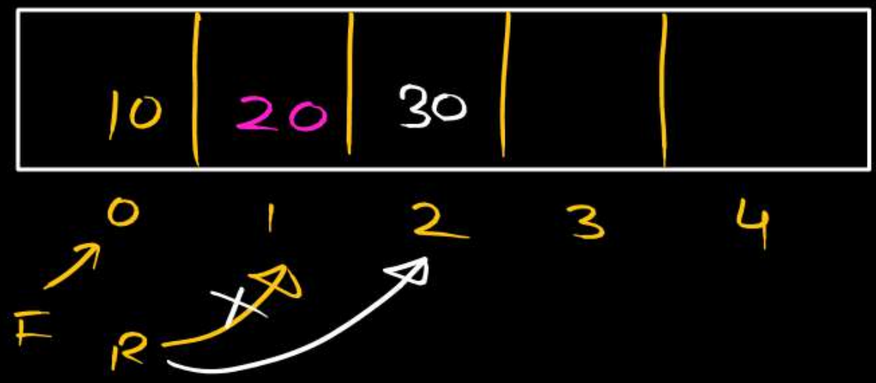


	F	R
Insert 60	3	0
Insert 70	3	1

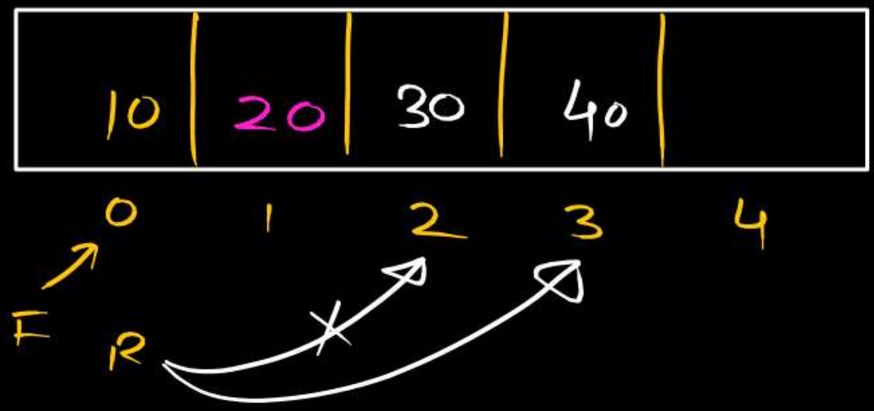
	F	R
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1



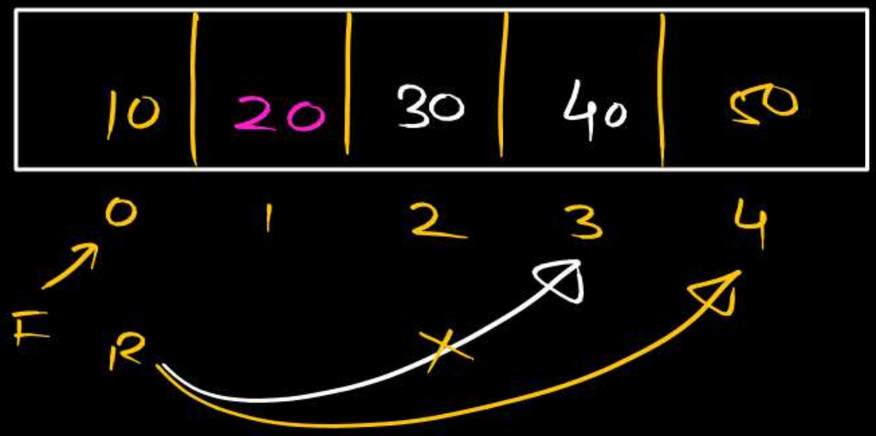
	F	R
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2



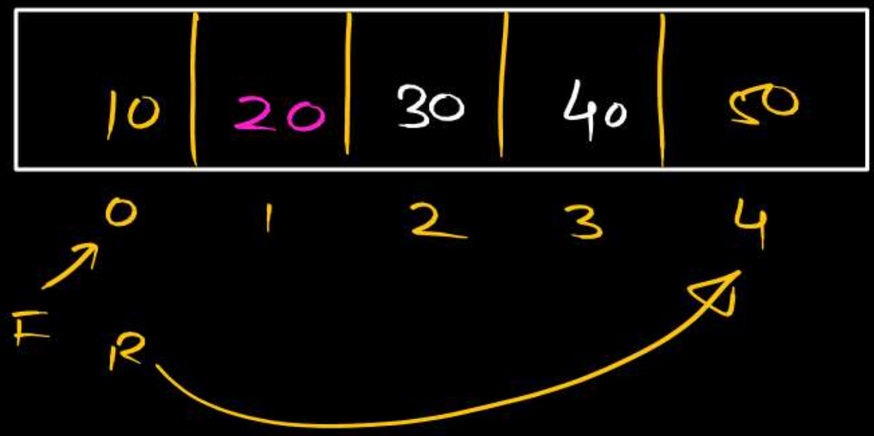
	F	R
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3



	F	R
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4



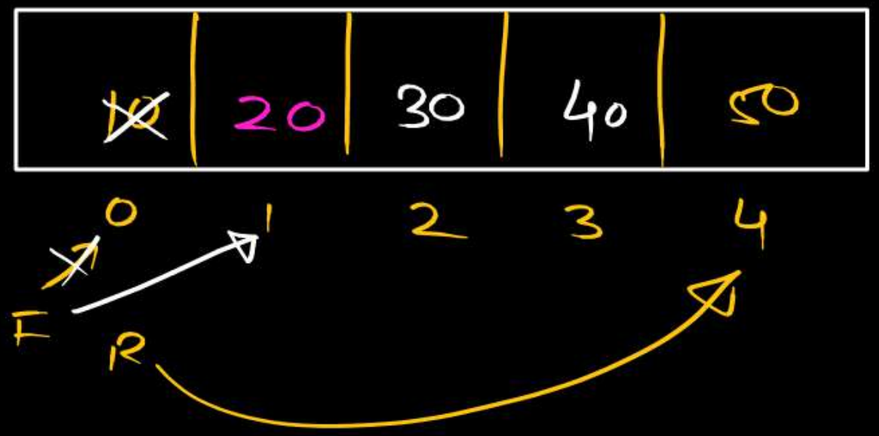
	F	R
Initially	-1	-1
Insert(10)	0	0
Insert(20)	0	1
Insert(30)	0	2
Insert(40)	0	3
Insert(50)	0	4



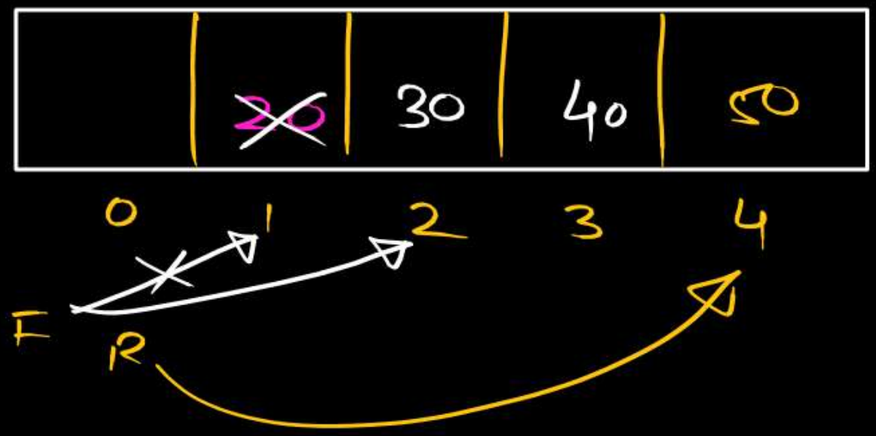
Full

$$\begin{cases} F \rightarrow 0 \\ R \rightarrow SIZE-1\ (4) \end{cases}$$

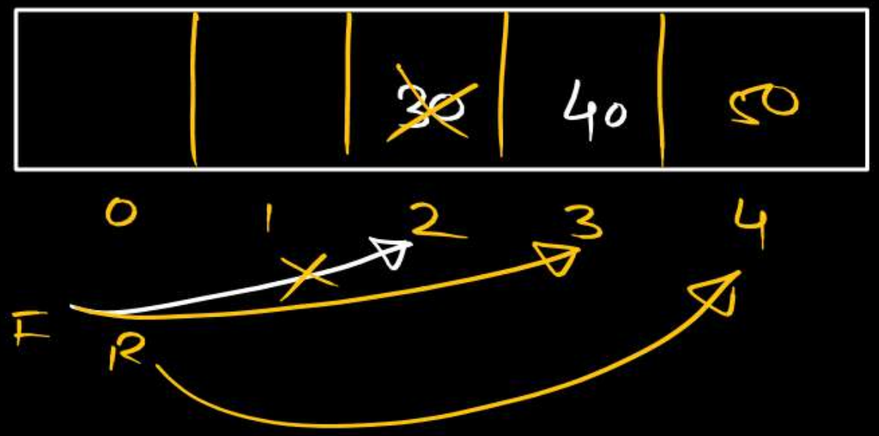
	F	R
	0	4
Delete()	1	4



	F	R
	0	4
Delete()	1	4
Delete()	2	4



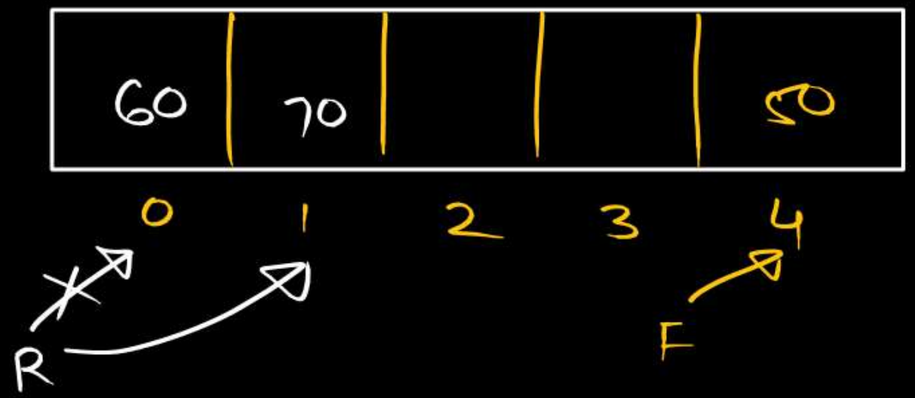
	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4



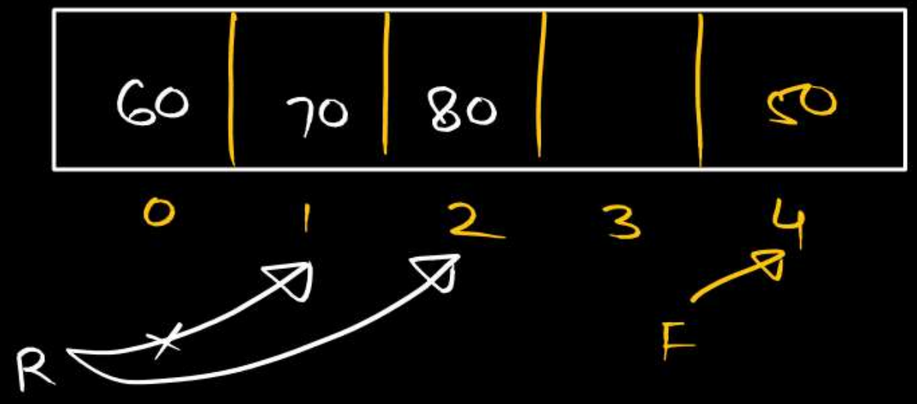
	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4
Delete	4	4

	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4
Delete	4	4
Insert 60		

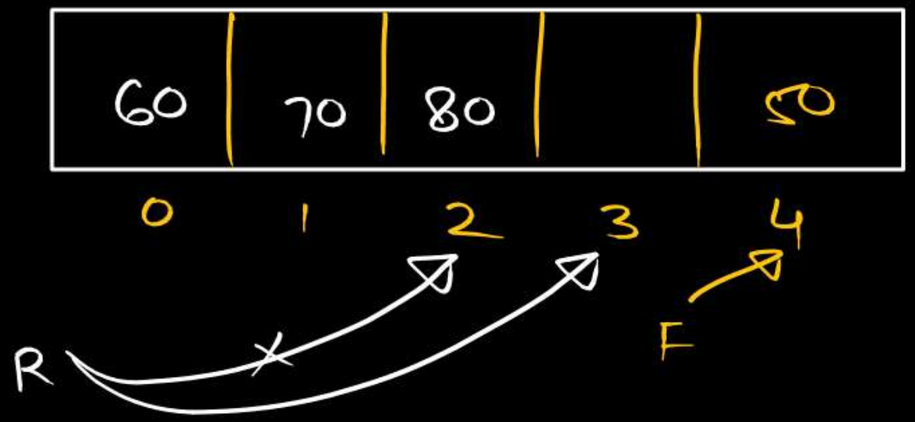
	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4
Delete	4	4
Insert 60	4	0
Insert 70		



	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4
Delete	4	4
Insert 60	4	0
Insert 70	4	1
Insert 80	4	

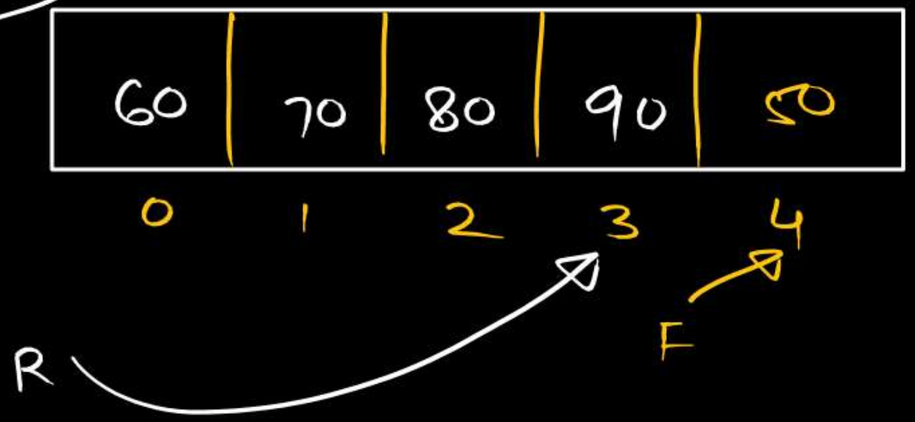


	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4
Delete	4	4
Insert 60	4	0
Insert 70	4	1
Insert 80	4	2
Insert 90	4	

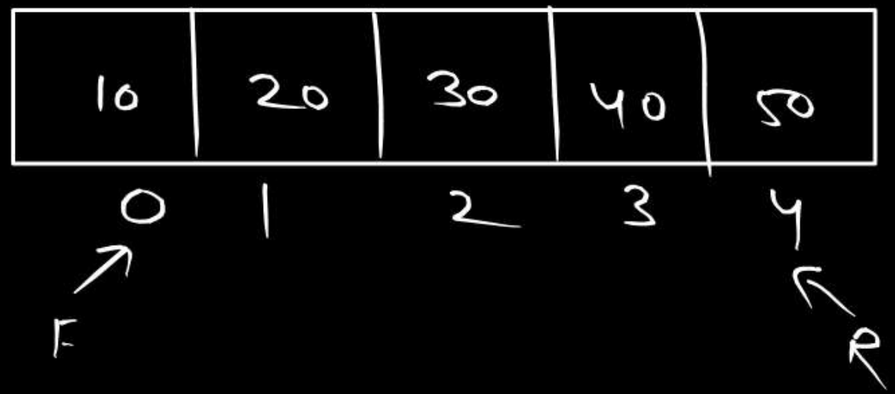


Front = (Rear+1) % SIZE

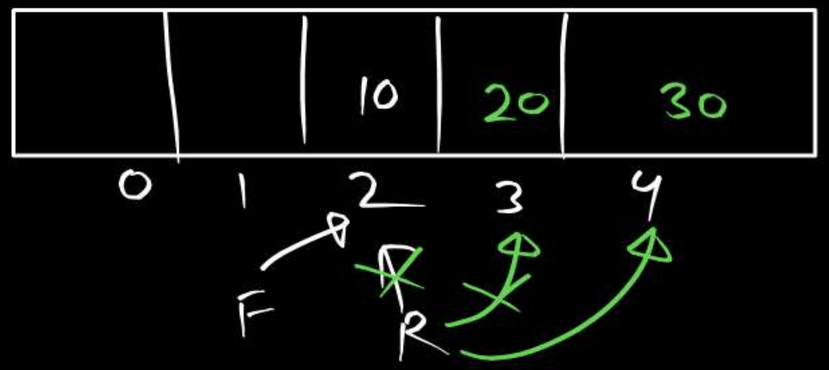
	F	R
	0	4
Delete()	1	4
Delete()	2	4
Delete()	3	4
Delete	4	4
Insert 60	4	0
Insert 70	4	1
Insert 80	4	2
Insert 90	4	3



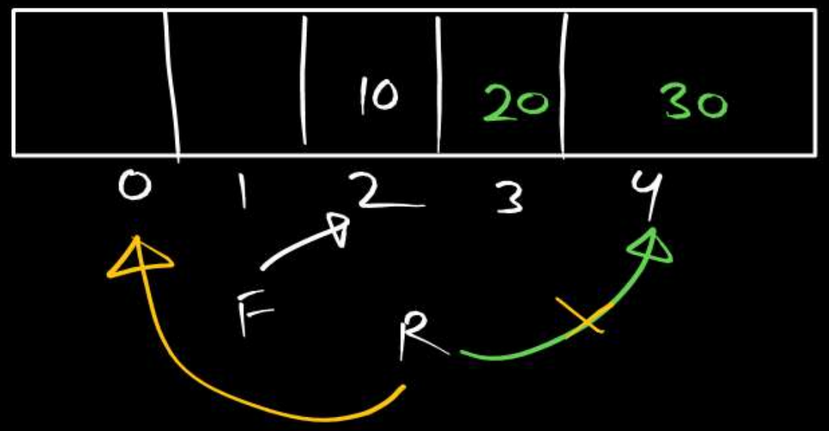
Queue is full : Front == Rear + 1



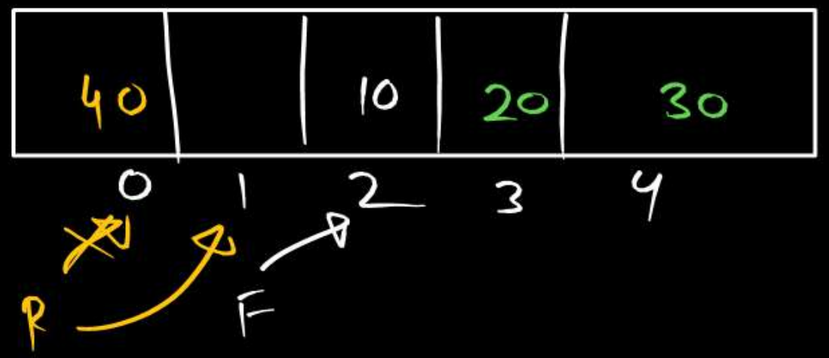
	F	R	
	2	2	
Insert 20	2	3	
Insert 30	2	4	



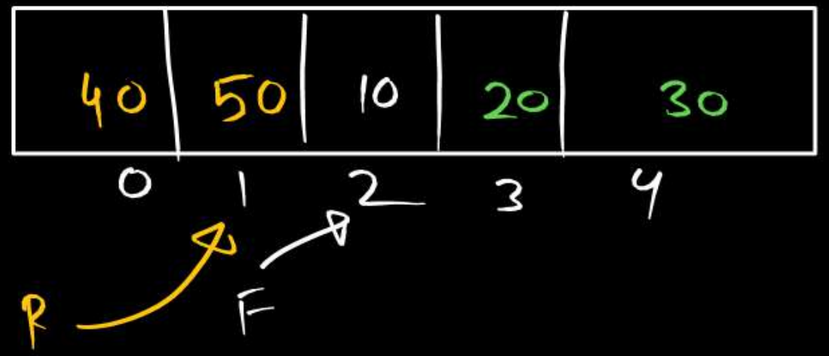
	F	R	
	2	2	
Insert 20	2	3	
Insert 30	2	4	
Insert 40	2	0	



	F	R	
	2	2	
Insert 20	2	3	
Insert 30	2	4	
Insert 40	2	0	
Insert 50	2	1	



	F	R
	2	2
Insert 20	2	3
Insert 30	2	4
Insert 40	2	0
Insert 50	2	1

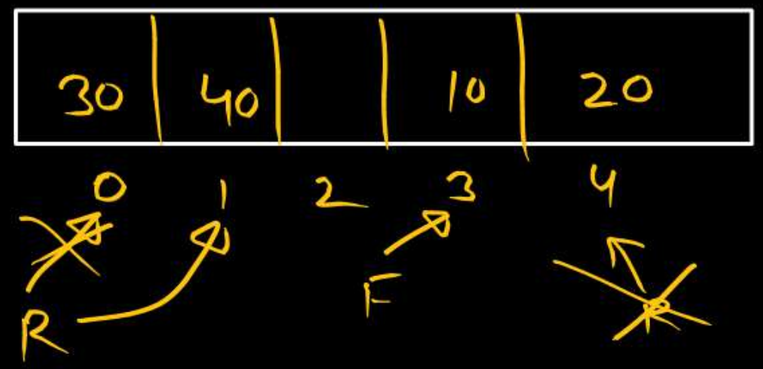


```
void CQ_insert(int key){

    if( Front == (Rear+1)% SIZE)
            return;

    if( Front == -1)
     {
        Front = Rear = 0;
        Queue[Rear] = key;
     }
    else if( Rear == SIZE-1)
        {
            Rear = 0;
            Queue[Rear] = key;
        }
    else{
        Rear++;
        Queue[Rear] = key;
        }
}
```

```
void CQ_insert(int key){
    if( Front == (Rear+1)%SIZE )
        return;
    if( Front == -1)
        Front = Rear = 0;
    else if( Rear == SIZE-1)
        Rear = 0;
    else
        Rear++;
    Queue[Rear] = key;
}
```

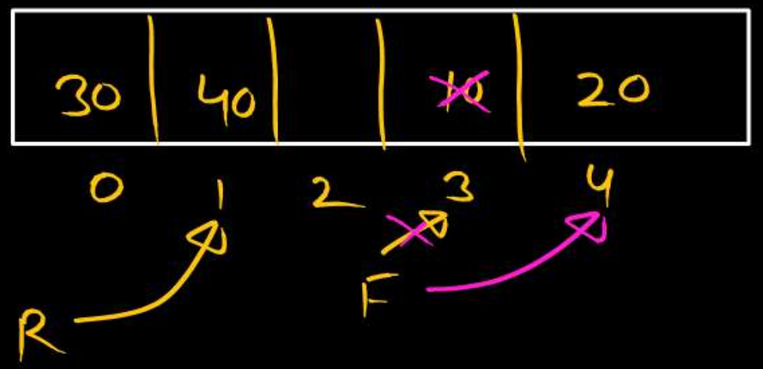
10, 20, 30

	F 3	R 4
Insert 30	3	0
Insert 40	3	1



10, 20, 30, 40

	F	R
	3	4
Insert 30	3	0
Insert 40	3	1
Delete	4	1

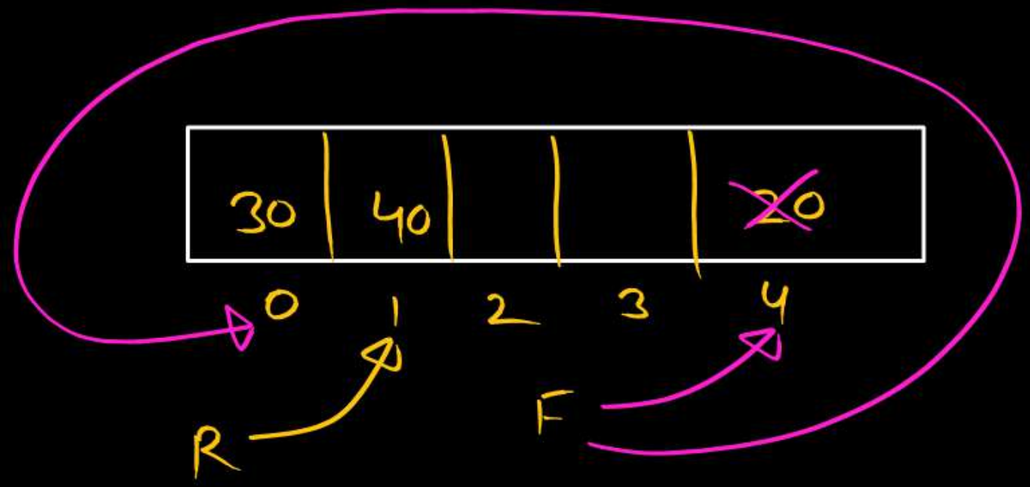


Delete

10, 20, 30, 40

F

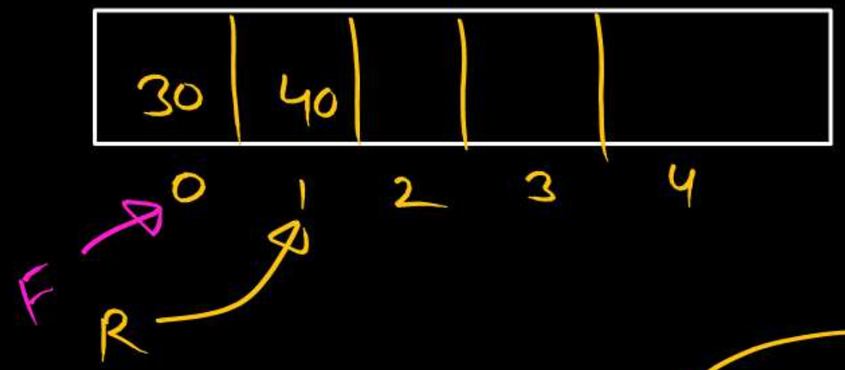
	F	R
	3	4
Insert 30	3	0
Insert 40	3	1
Delete	4	1
Delete	0	1



~~10~~, ~~20~~, 30, 40

F

	F 3	R 4
Insert 30	3	0
Insert 40	3	1
Delete	4	1
Delete	0	1



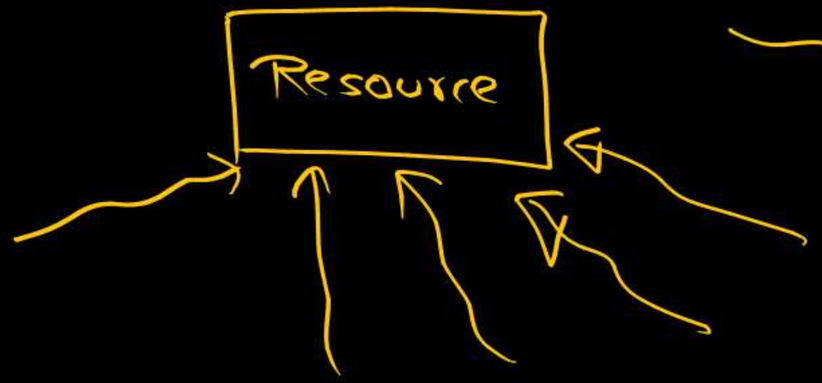
2 min

```
int CQ_delete() { int temp;
    if ( Front == -1)
        return INT_MIN;
    else if( Front == Rear)
        {
          temp = Queue[Front];
          Front = Rear = -1;
        }
    else if( Front == SIZE-1)
        {
          temp = Queue[Front];
          Front = 0;
        }
    else {
          temp = Queue[Front];
          Front ++;
          }
    return temp;
}
```

constant time

- Stack → Push, Pop
- Queue → simple
  - CQ

	Insert	delete
	O(1)	O(1)
	O(1)	O(1)



Applications

1. CPU scheduling
2. Spooling
3. Slow & Fast device ⇒ Synch.

THANK - YOU