CS & IT ENGINEERING

Data Structure & Programming

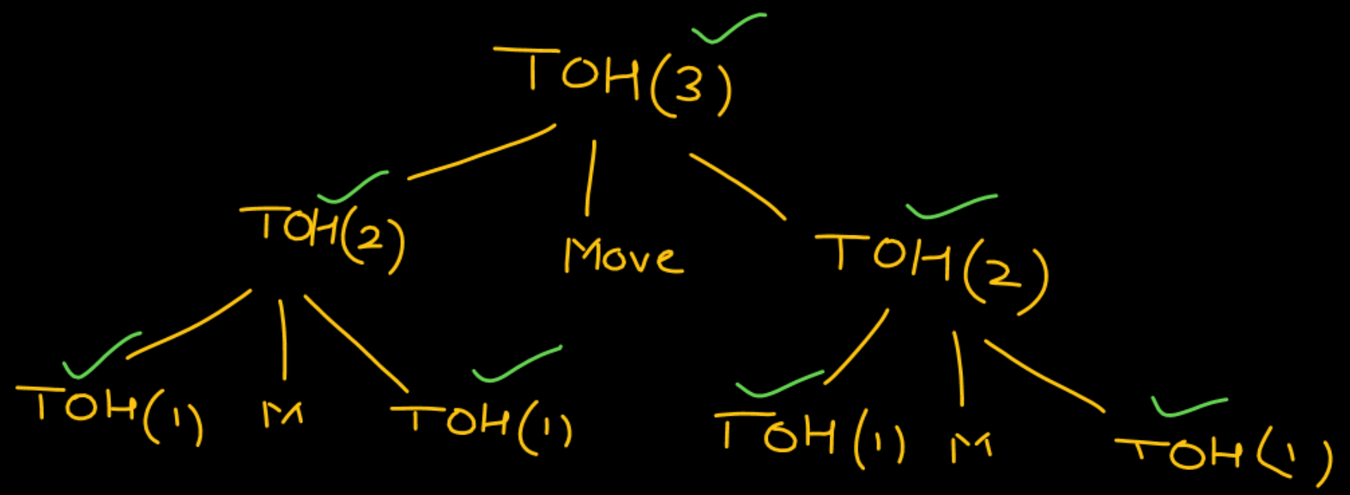
Stack and Queues

Lec- 04

By- Pankaj Sharma sir

Stack-4

1. No. of functions calls in TOH(n)

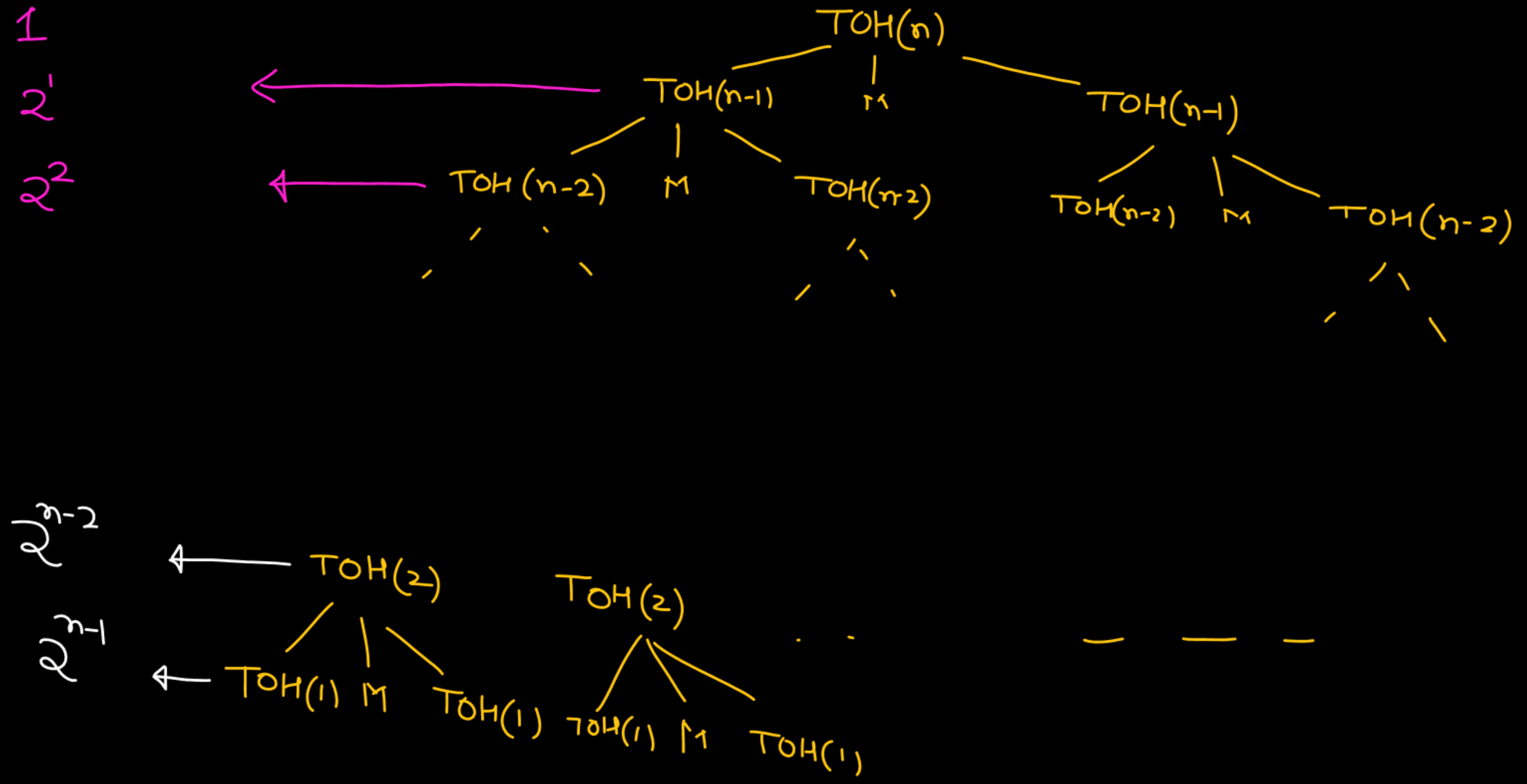


n = 3

No. of function calls

7

No. of calls


1
2^1
2^2
TOH(n)
TOH(n-1)
M
TOH(n-1)
TOH(n-2)
M
TOH(n-2)
TOH(n-2)
M
TOH(n-2)
2^(n-2)
2^(n-1)
TOH(2)
TOH(2)
TOH(1)
M
TOH(1)
TOH(1)
M
TOH(1)

function calls = $1 + 2^1 + 2^2 + \cdots + 2^{n-1}$ (G.P n terms)

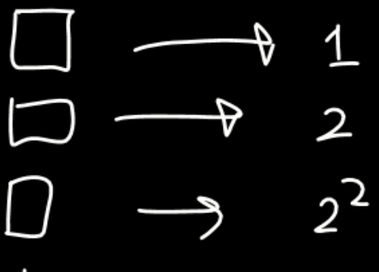
$$= \frac{(2^n - 1)}{2 - 1} = 2^n - 1$$

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

$$= O(2^n)$$

$2^n \rightarrow$ नरक

2^{450}



1kg → 1 Lac

1,00,000 ⇒ 10^5

$2^{10} = 1024 = 10^3$

$\frac{8 \times 10^{18}}{10^5}$ kg

8×10^{13} kg

8000000000000000 kg

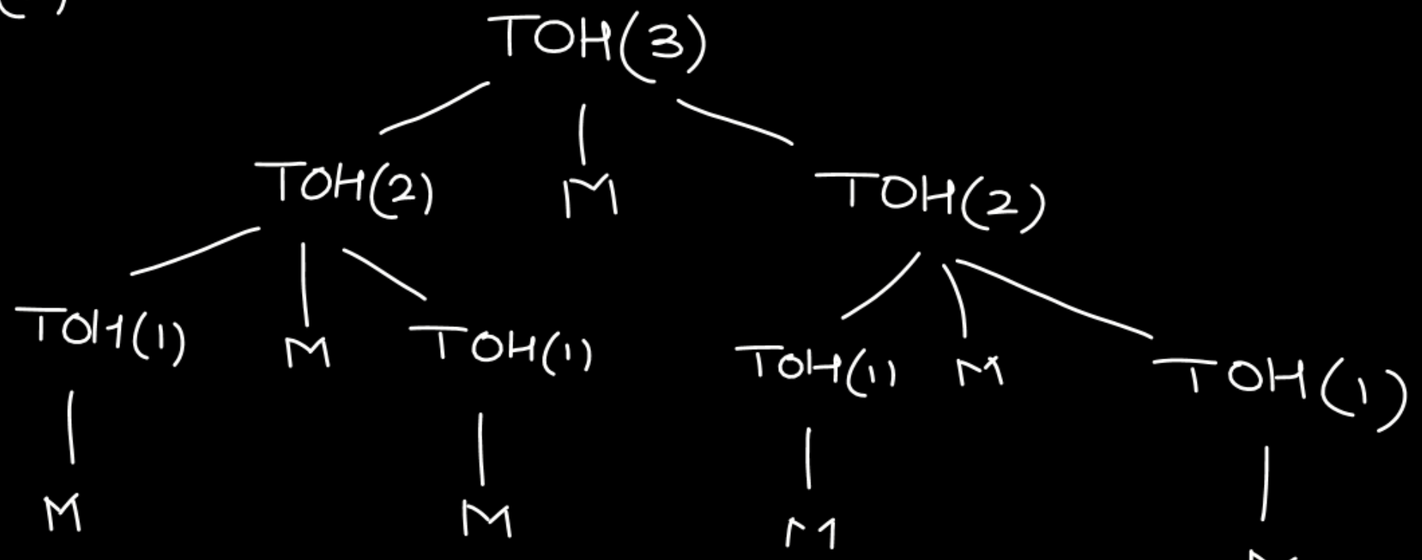
2^{63}

$\Rightarrow (2^{10})^6 \times 2^3$

$= (10^3)^6 \times 2^3$

$= 8 \times 10^{18}$

2. # moves in TOH(n)



$= 2^n - 1$

Fibbonacci Series

Fix

0, 1, 1, 2, 3, 5, 8

$$Fib(n) = \begin{cases} Fib(n-1) + Fib(n-2) & n \geq 2 \\ 0 & n = 0 \\ 1 & n = 1 \end{cases}$$

$Fib(0) = 0$

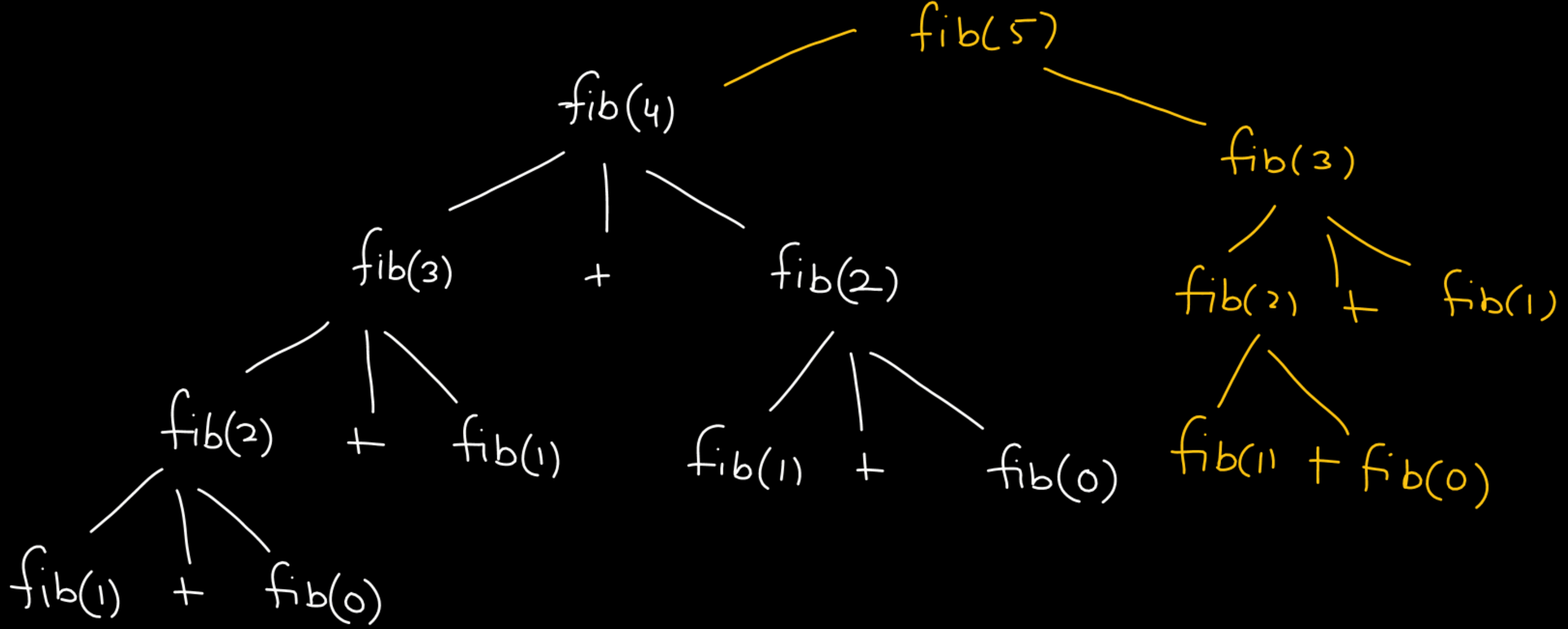
$Fib(1) = 1$

$Fib(2) = Fib(1) + Fib(0) = 1 + 0 = 1$

$Fib(3) = Fib(2) + Fib(1)$

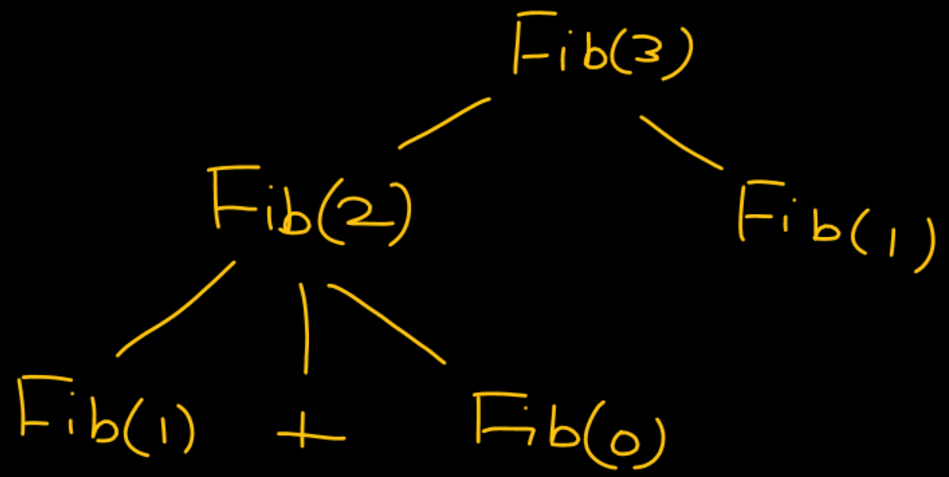
$= 1 + 1 = 2$

```
int fib(int n) {
    if (n == 0 || n == 1)
        return n;

    return fib(n-1) + fib(n-2);
}
```

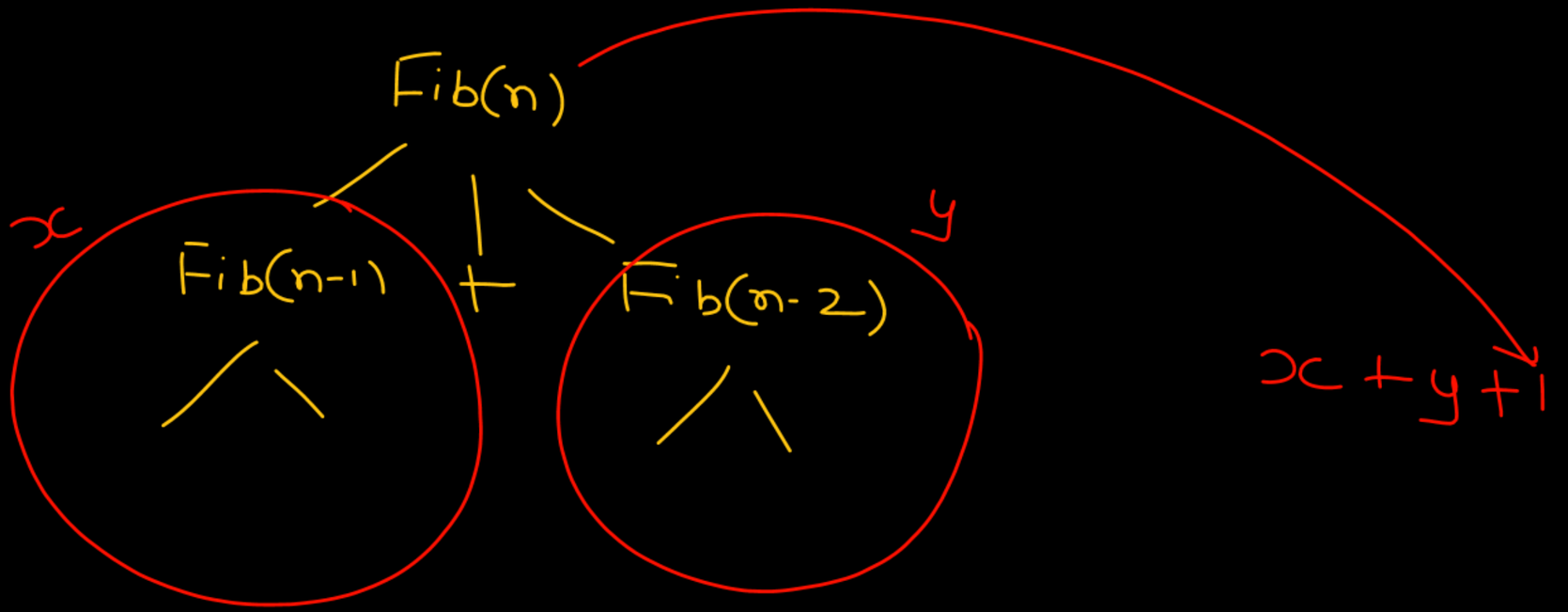

fib(5)
fib(4)
fib(3)
fib(3)
+
fib(2)
fib(2)
+
fib(1)
fib(2)
+
fib(1)
fib(1)
+
fib(0)
fib(1)
+
fib(0)
fib(1)
+
fib(0)

1/ No. of function calls in fib(n)

	calls
Fib(0)	1
Fib(1)	1
Fib(2)	3
Fib(3)	5



Let $H(n)$ be the no. of function calls in Fib(n)



$$H(n) = H(n-1) + H(n-2) + 1$$

$$H(0) = 1$$
$$H(1) = 1$$

$$H(n) = H(n-1) + H(n-2) + 1$$

$\Rightarrow$ Fibonacci

$H(0) = 1$

$H(1) = 1$

$H(2) = H(1) + H(0) + 1$

$= 1 + 1 + 1 = 3$

$H(3) = H(2) + H(1) + 1$

$= 3 + 1 + 1 = 5$

n	0	1	2	3	4	5	6
$H(n)$	1,	1,	3	5	9	15	25

2. Total no. of additions in calculating Fib(n)

Fib(0) = 0

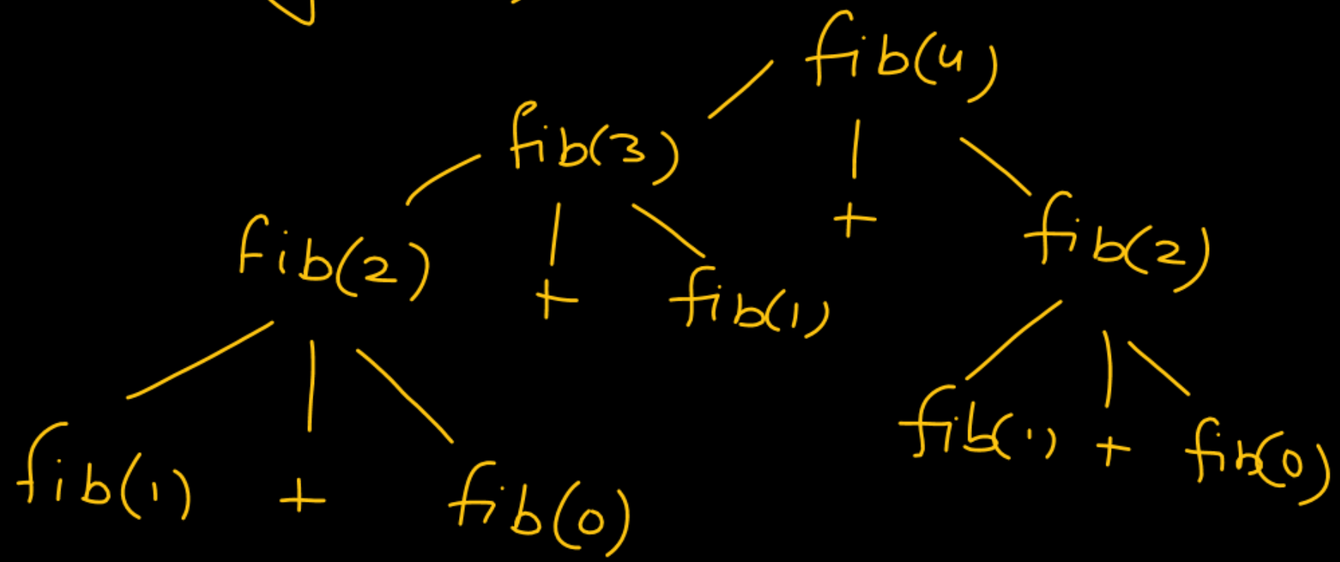
Fib(1) = 0

Fib(2) = 1

Fib(3) = 2

fib(4) = 4

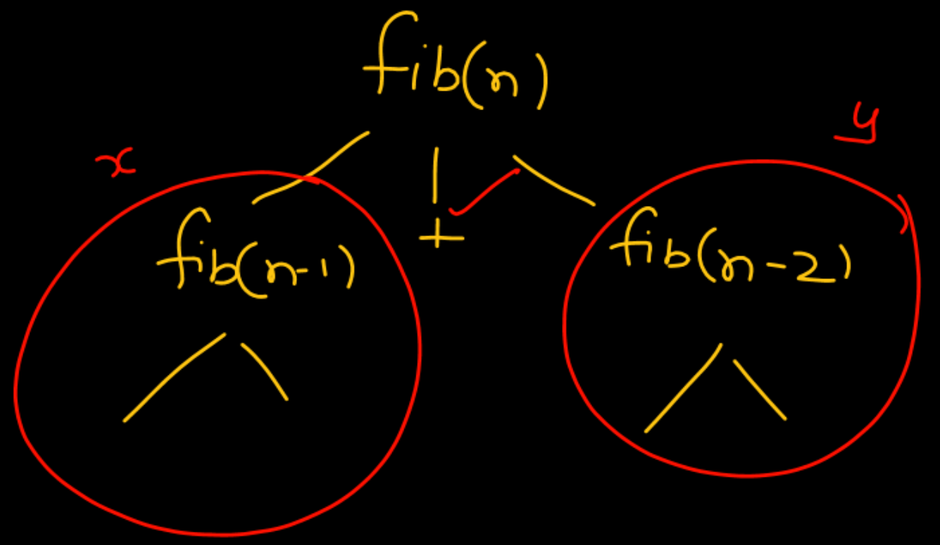
fib(5) = 7



Let $G(n)$ be the no. of addition in $fib(n)$

$G(0) = 0$

$G(1) = 0$



$G(n) = G(n-1) + G(n-2) + 1$

$G(0) = 0$

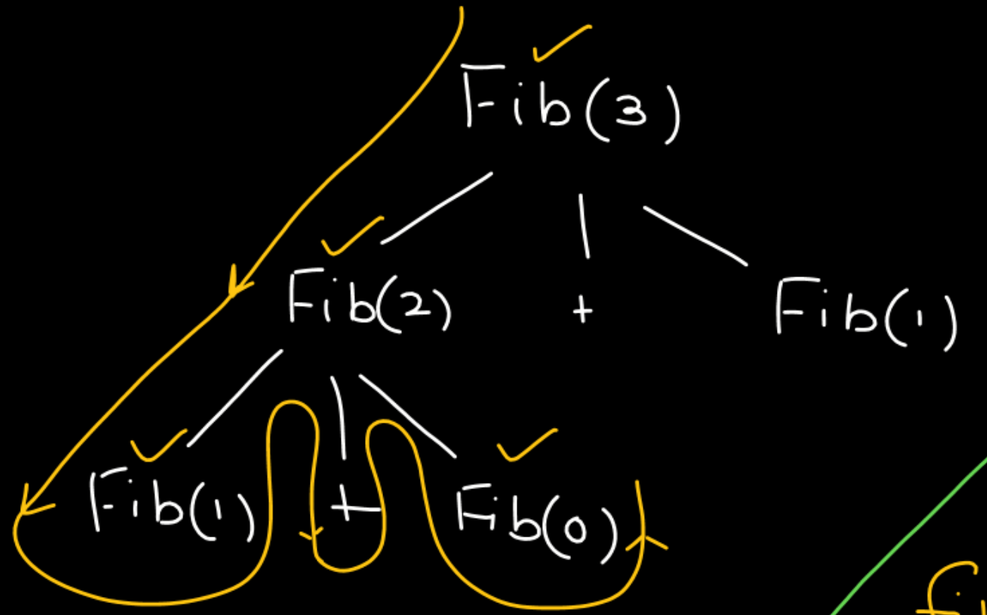
$G(1) = 0$

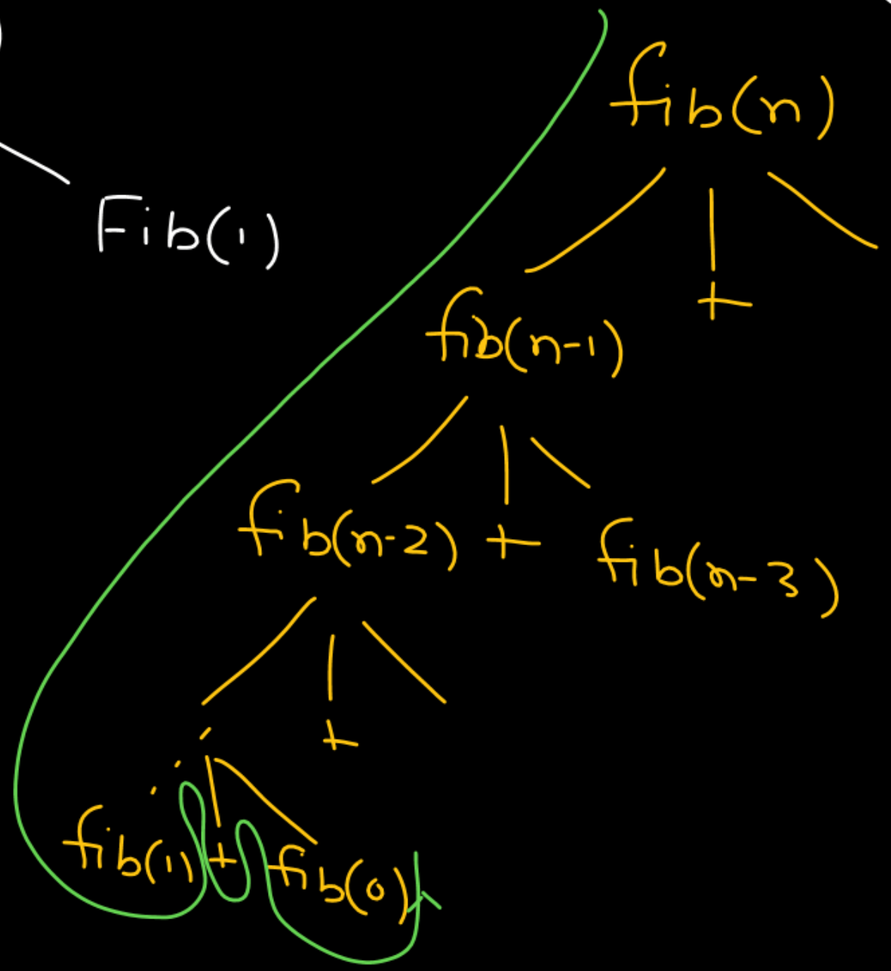
$x + y + 1$

n	0	1	2	3	4	5	6
$G(n)$	0	0	1	2	4	7	12

...

3. No. of function calls invoked before 1st addition take place in fib(n) {Recursion}





4. Code → $O(2^n)$ → नरक

Prog → How to write code

Algo → Efficient code

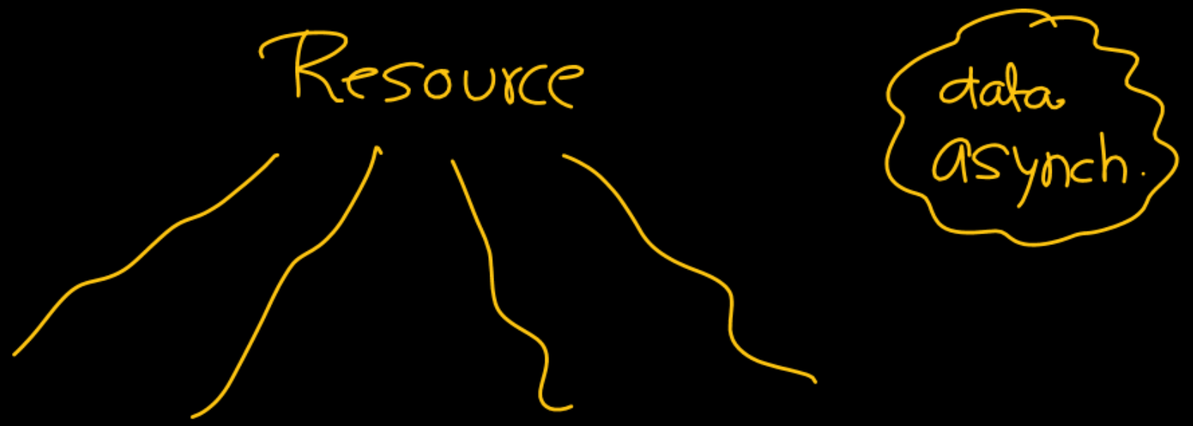
Queue

* Linear data structure

* First - In First Out

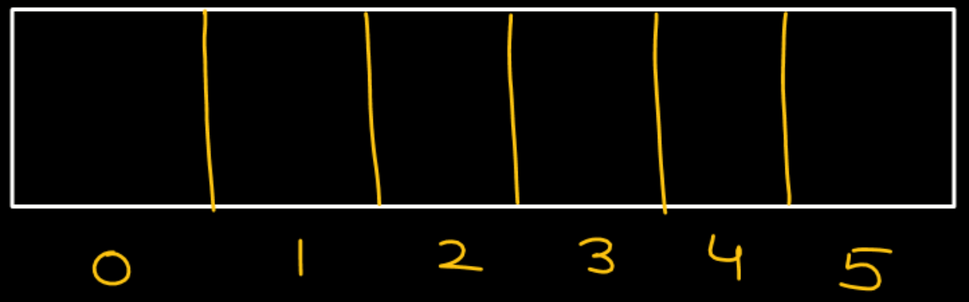
* Order of deletion is same as order of insertion

Insertion : Rear
Deletion : Front


Resource
data
asynch.

Simple Queue

Queue




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

Initially, when Queue is Empty

Front : Index from which an element can be deleted.

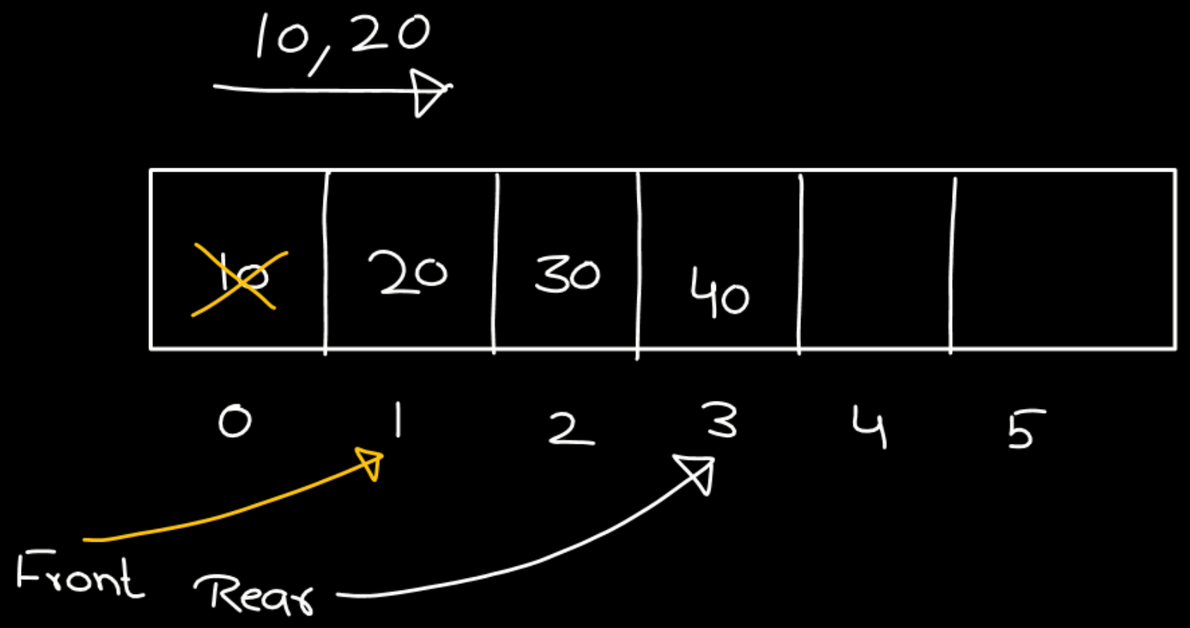
Rear : Index of most recently added element.

$$\begin{cases} Front = -1 \\ Rear = -1 \end{cases}$$

Case1 : when we insert 1st element

⇒ F = R = 0

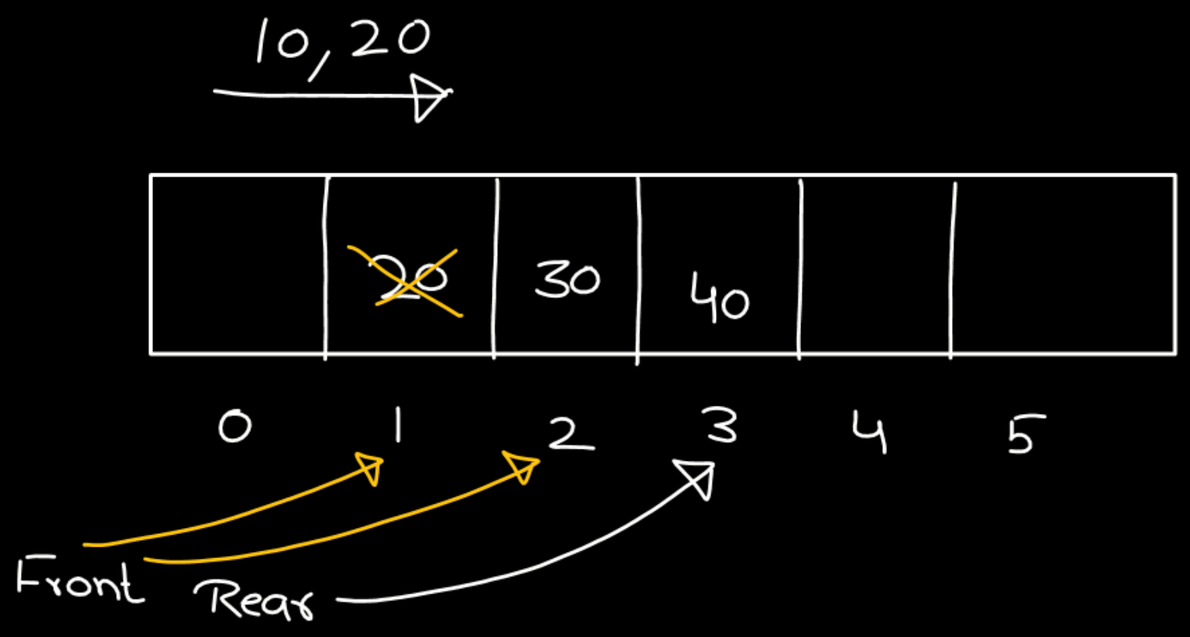
	F	R	
Initially	-1	-1	
Insert(10)	0	0	
Insert(20)	0	1	
Insert(30)	0	2	
Insert(40)	0	3	
delete			



Case1 : when we insert 1st element

⇒ F = R = 0

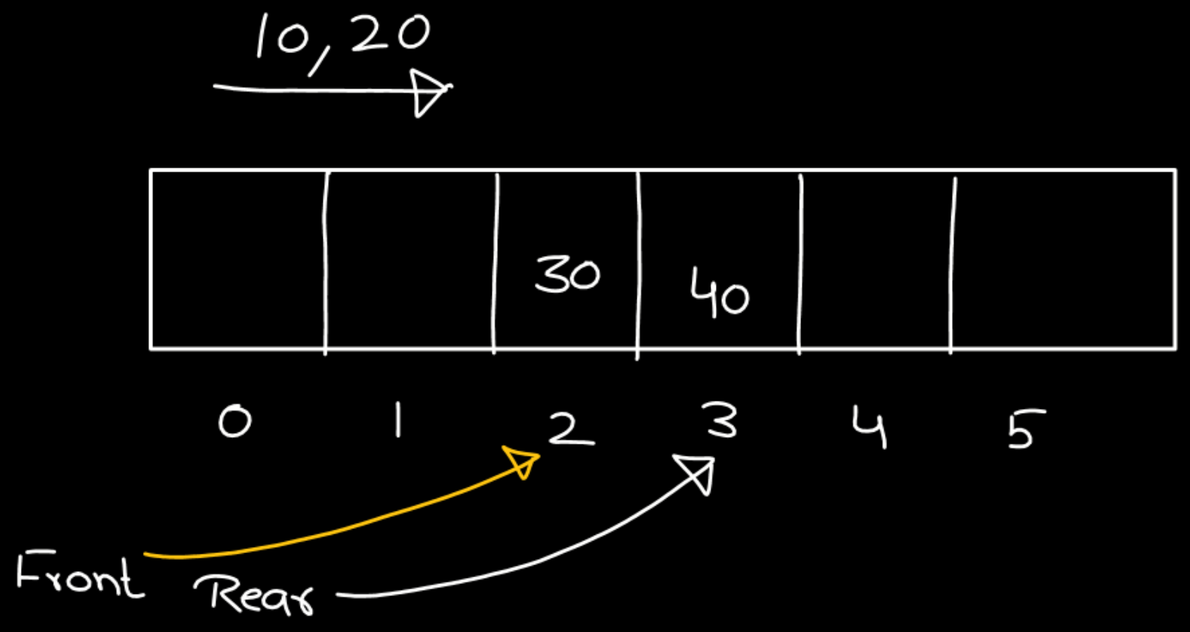
	F	R	
Initially	-1	-1	
Insert(10)	0	0	
Insert(20)	0	1	
Insert(30)	0	2	
Insert(40)	0	3	
delete	1	3	
delete			



Case1 : when we insert 1st element

⇒ F = R = 0

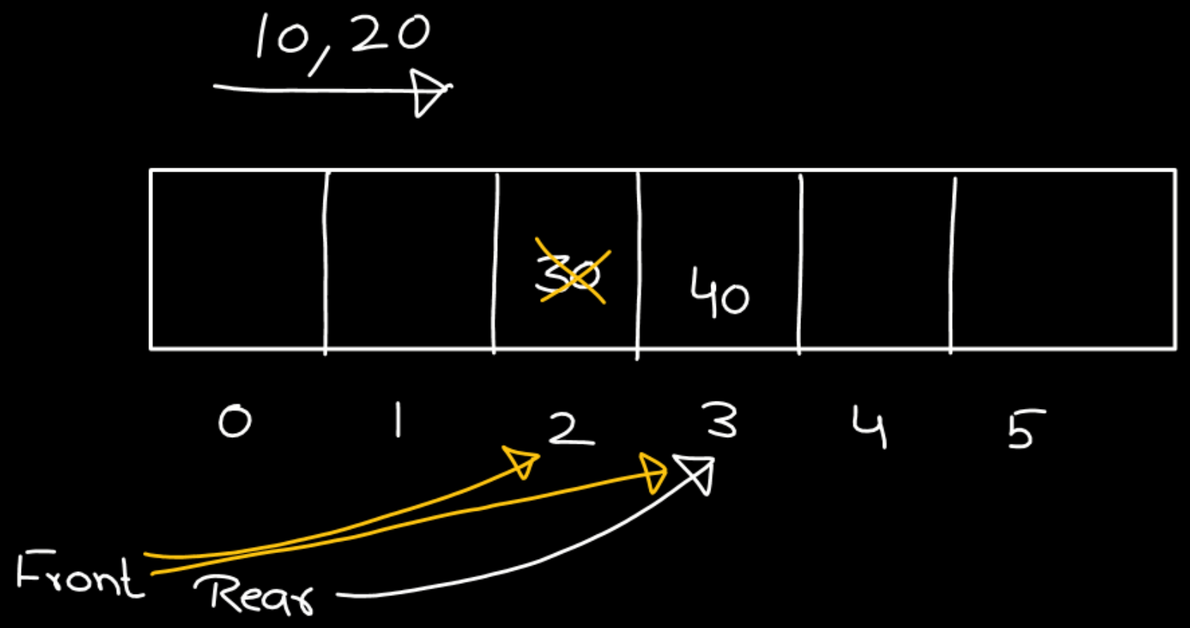
	F	R	
Initially	-1	-1	
Insert(10)	0	0	
Insert(20)	0	1	
Insert(30)	0	2	
Insert(40)	0	3	
delete	1	3	
delete	2	3	



Case1 : when we insert 1st element

⇒ F = R = 0

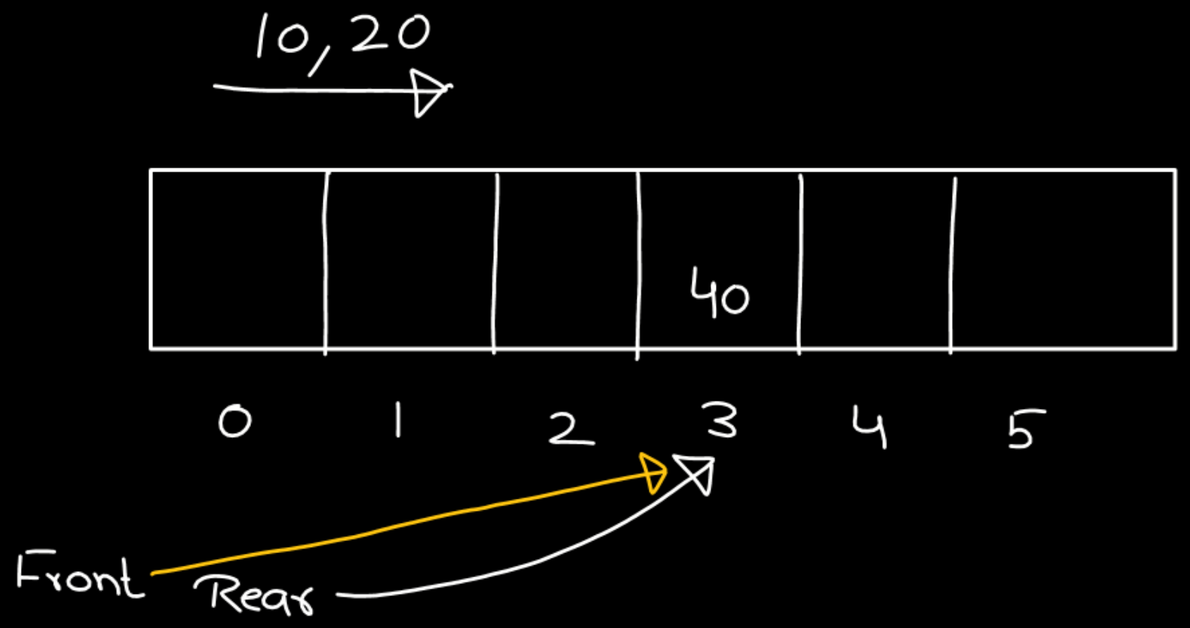
	F	R	
Initially	-1	-1	
Insert(10)	0	0	
Insert(20)	0	1	
Insert(30)	0	2	
Insert(40)	0	3	
delete	1	3	
delete	2	3	
delete			



Case1 : when we insert 1st element

⇒ F = R = 0

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



Front == Rear

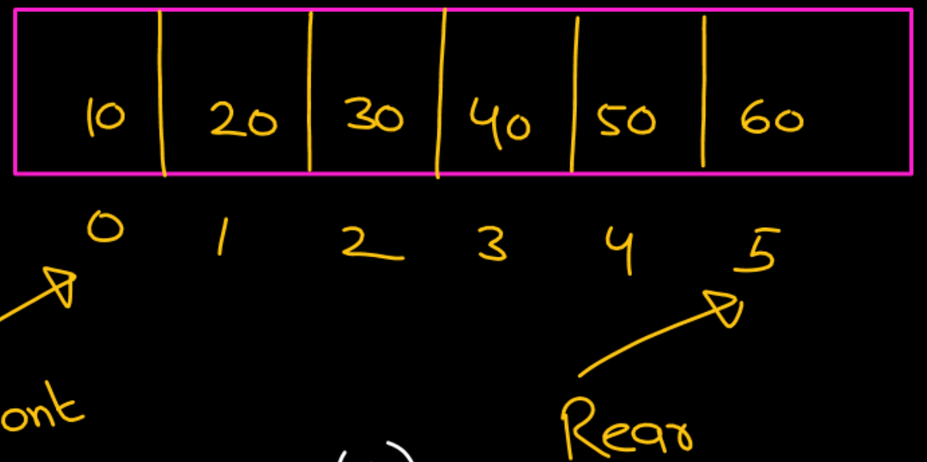
Case 1 : When Queue is Empty

Front == Rear = -1

Case 2 : When there is only 1 element in Queue

Front == Rear ≠ -1

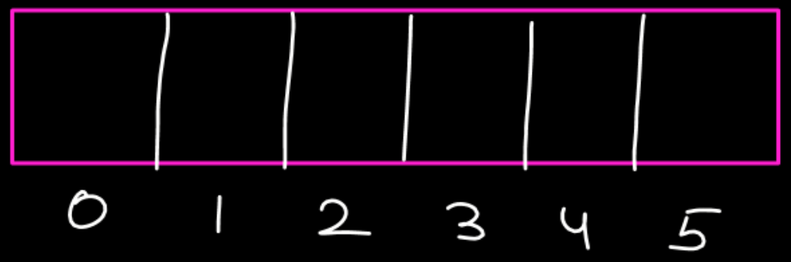
```
void EnQueue(int x){
    if(Rear == SIZE-1)
            return;

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



EnQueue ⇒ O(1)

DeQueue

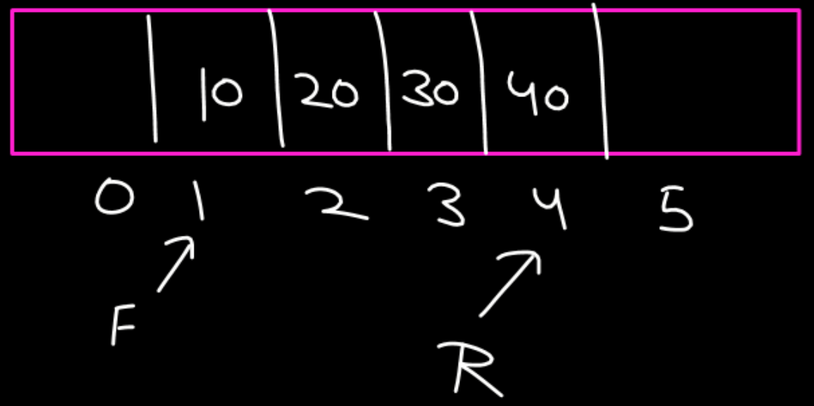
1.



Front = -1
Rear = -1

$\Rightarrow$ Can not delete

2.



Save element
Front = Front + 1
ele $\Rightarrow$ return

```
int DeQueue( ) {
    int temp;
    if( Front == -1)
        return INT_MIN;
```

Empty

sure atleast 1 element

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

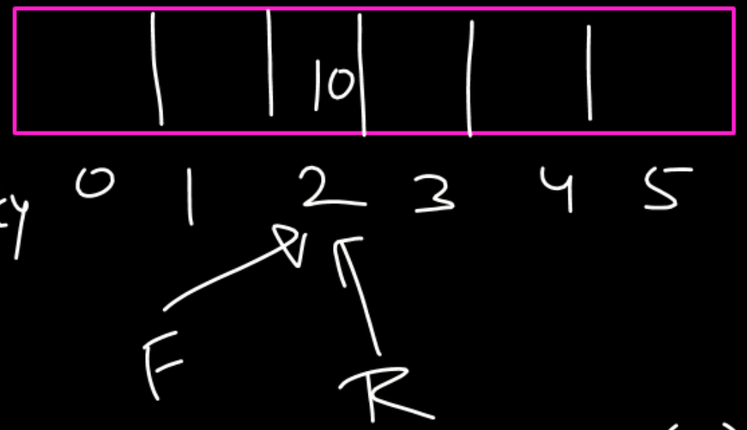
1 element

Single element

deletion $\Rightarrow O(1)$

```
else {
        temp = Queue[Front];
        Front ++;
        return temp;
    }
}
```

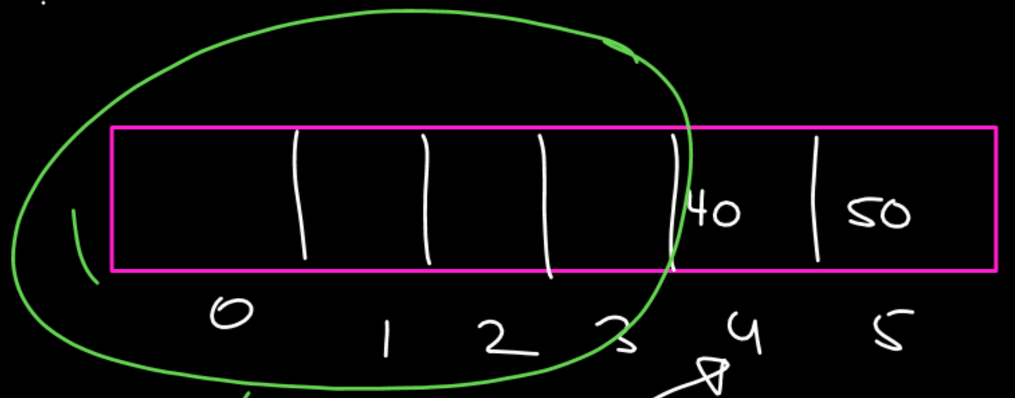
2 or more element



Sunday - 3 classes

Problem ?

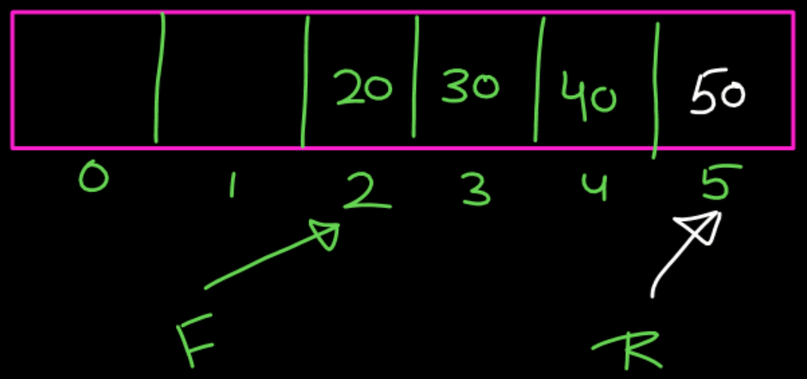
Overflow



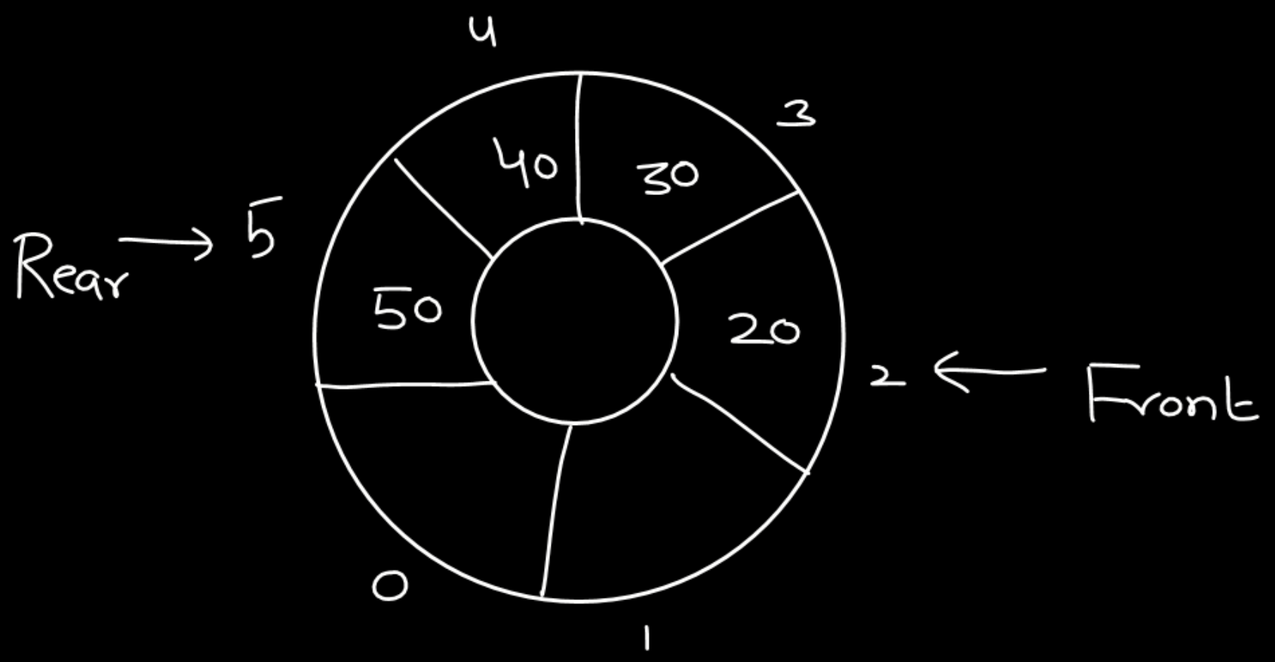
Space is available

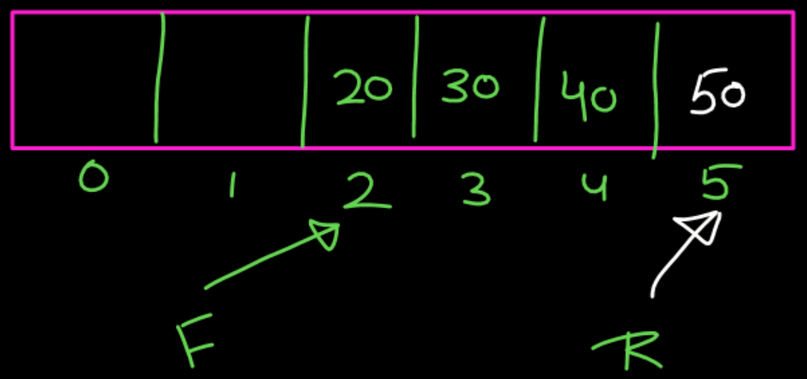
Circular Queue

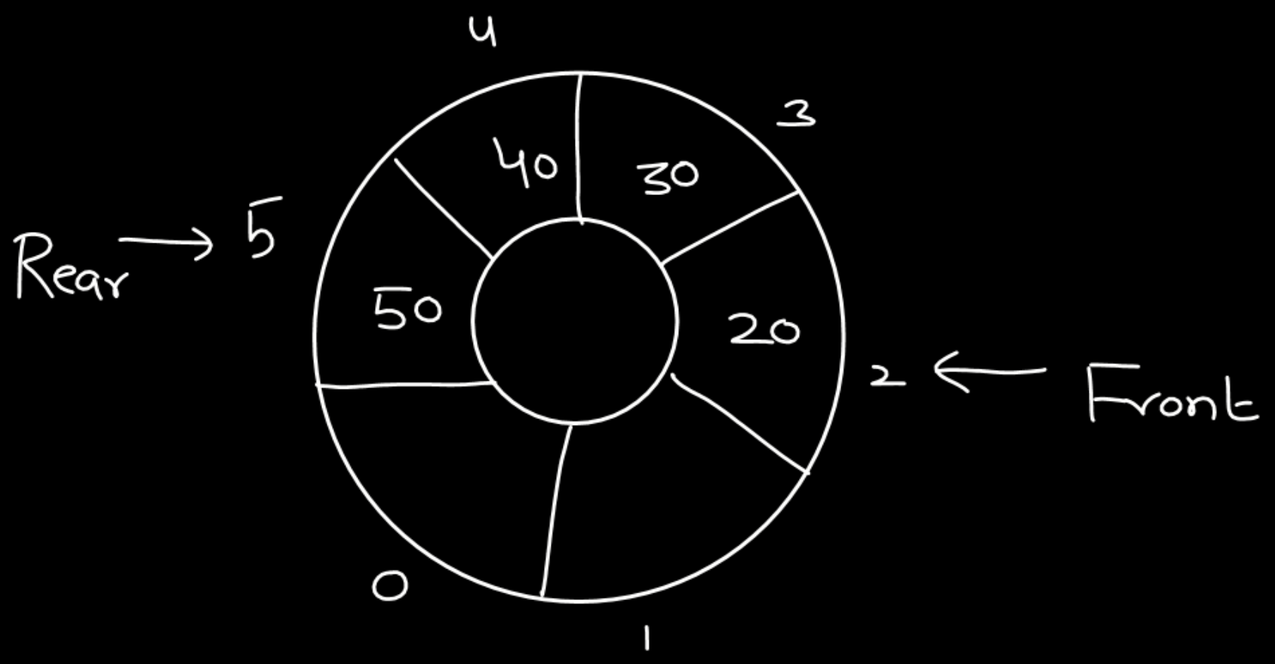
Overflow



Insert(50) ⇒ EnQueue(50)



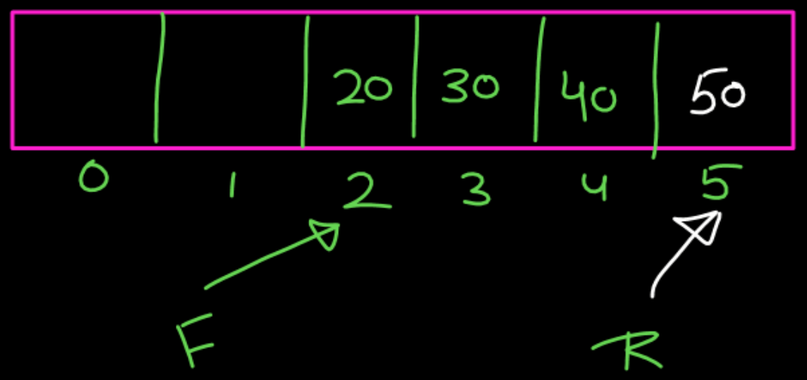


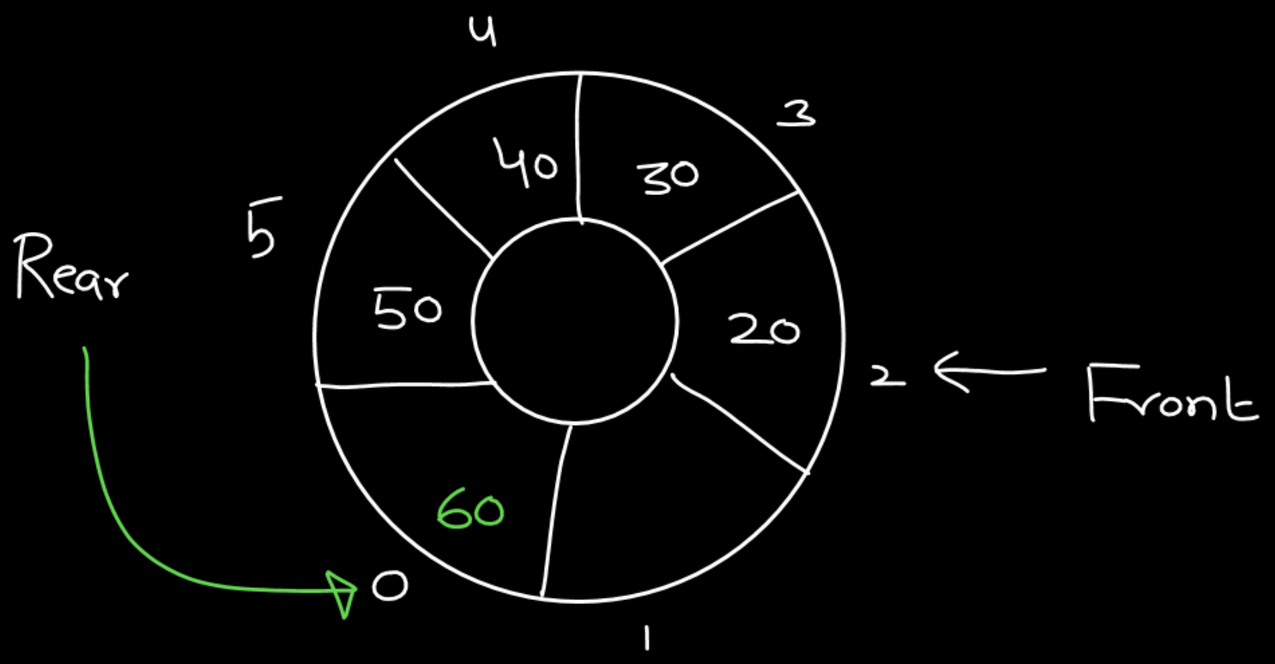


Insert(50) ⇒ EnQueue(50)

EnQueue(60) ⟶ Rear = 0

Queue[Rear] = x ;





Insert(50) ⇒ EnQueue(50)

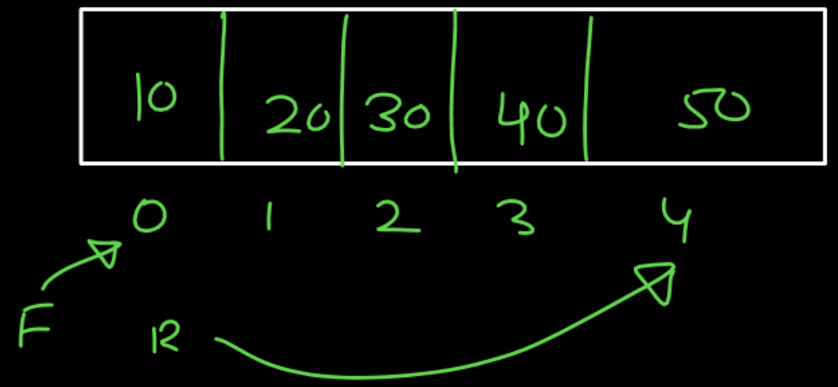
EnQueue(60) ⟶ Rear = 0

Queue[Rear] = x ;

Case 1 :

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

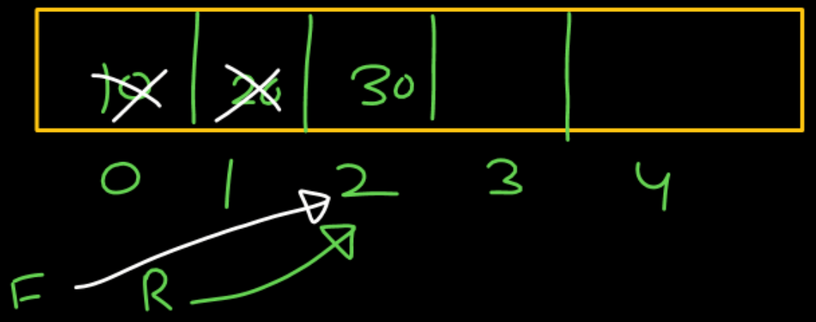
EnQueue ⇒ Overflow



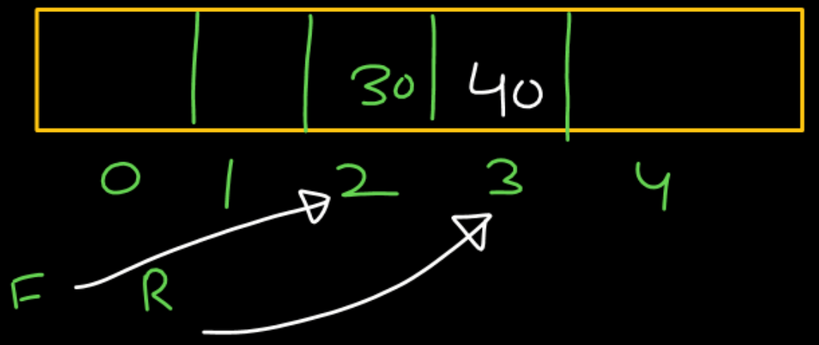
Queue is Full

Front == 0 && Rear == SIZE - 1

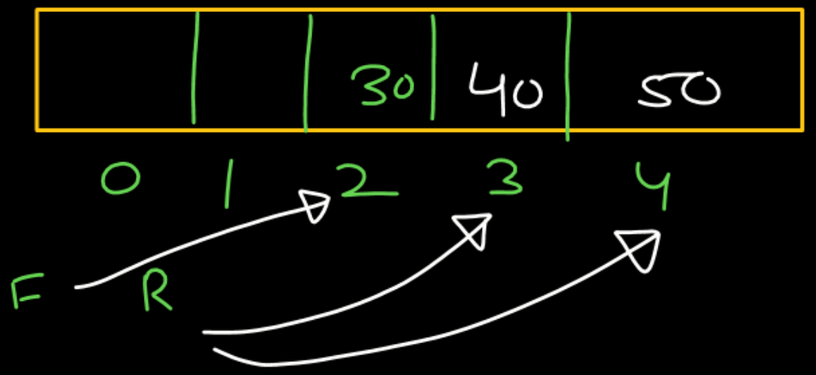
	F	R
Initially	-1	-1
EnQueue 10	0	0
EnQueue 20	0	1
EnQueue 30	0	2
DeQueue	1	2
DeQueue	2	2



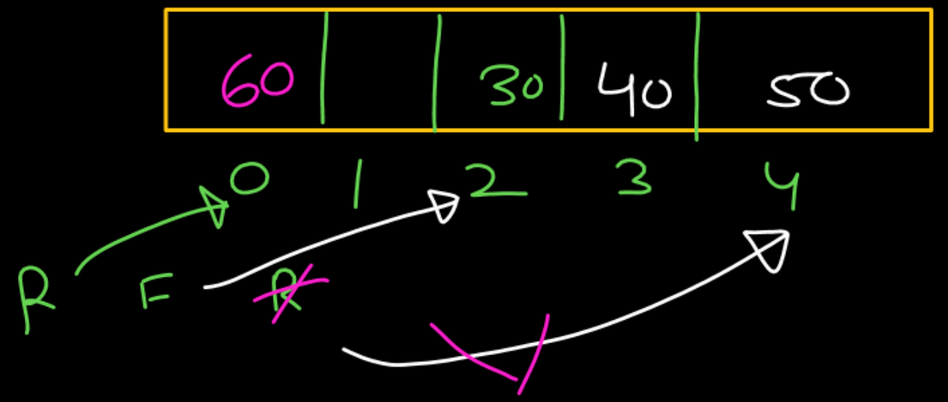
	F	R
Initially	-1	-1
EnQueue 10	0	0
EnQueue 20	0	1
EnQueue 30	0	2
DeQueue	1	2
DeQueue	2	2
EnQueue 40	2	3

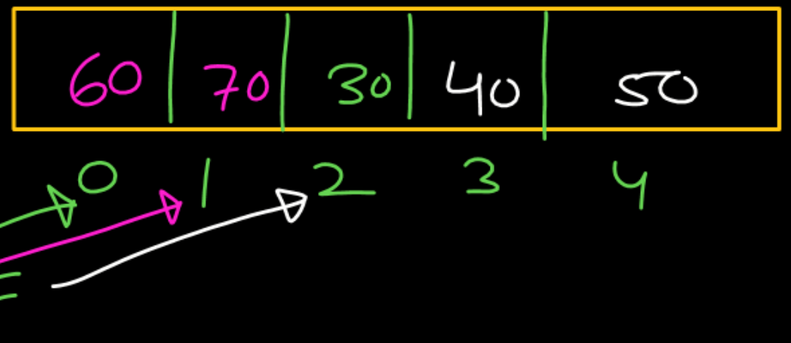


	F	R
Initially	-1	-1
EnQueue 10	0	0
EnQueue 20	0	1
EnQueue 30	0	2
DeQueue	1	2
DeQueue	2	2
EnQueue 40	2	3
EnQueue 50	2	4



	F	R
Initially	-1	-1
EnQueue 10	0	0
EnQueue 20	0	1
EnQueue 30	0	2
DeQueue	1	2
DeQueue	2	2
EnQueue 40	2	3
EnQueue 50	2	4
EnQueue 60	2	0

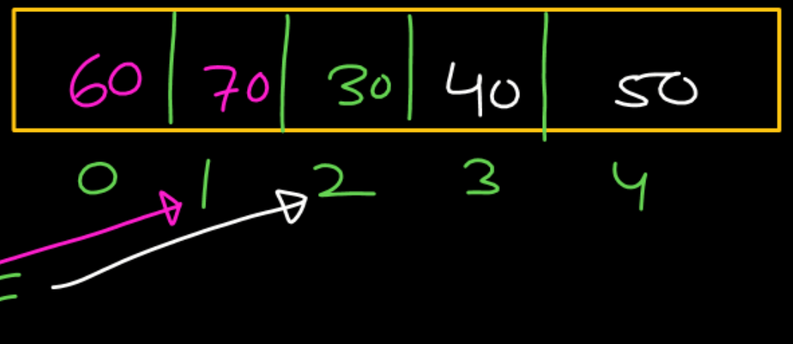




	F	R
Initially	-1	-1
EnQueue 10	0	0
EnQueue 20	0	1
EnQueue 30	0	2
DeQueue	1	2
DeQueue	2	2
EnQueue 40	2	3
EnQueue 50	2	4
EnQueue 60	2	0

		F	R
EnQueue	70	2	1

	F	R
Initially	-1	-1
EnQueue 10	0	0
EnQueue 20	0	1
EnQueue 30	0	2
DeQueue	1	2
DeQueue	2	2
EnQueue 40	2	3
EnQueue 50	2	4
EnQueue 60	2	0



Full

F == R+1

	F	R
EnQueue 70	2	1

(i) F = 0 && R = SIZE-1

(ii) F := R+1

F == (R+1) % SIZE

- Circular Queue
- Priority Queue
- PYQs


THANK YOU GW
SOLDIERS !