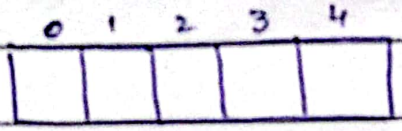
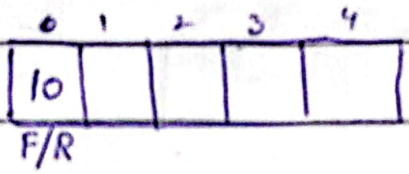
Day: _____ Date: _____

⇒ Insertion At Front



F = R = 0 - 1

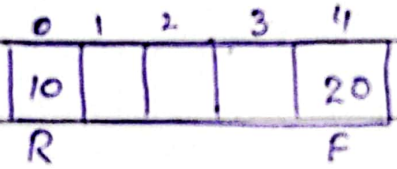
Insert_At_Front(10)



R = 0
F = 0

Insert At Front (20)

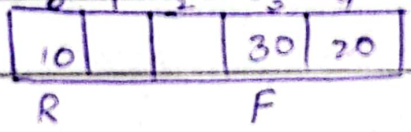
∴ F = 0
∴ F = n - 1



R = 0
F = n - 1 = 4

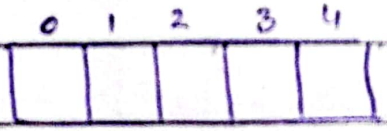
Insert_At_Front (30)

F--



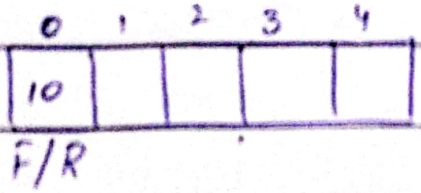
R = 0
F = 4 - 1 = 3

⇒ Insert At Rear



F = R = -1

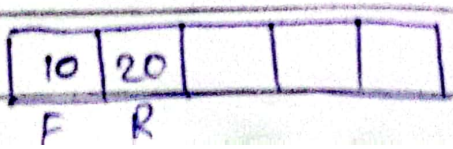
Insrt_At_Rear (10)



F = 0
R = 0

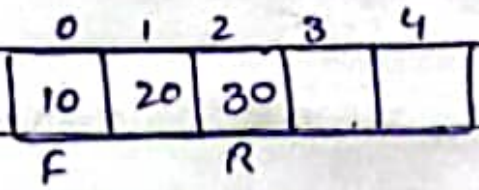
Insert_At_Rear (20)

R++



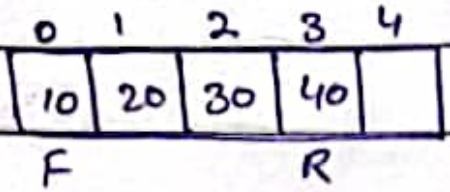
R = 0 + 1 = 1
F = 0

insert_At_Rear(30)



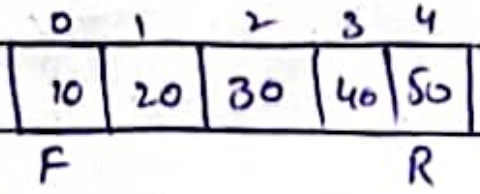
R++
R=2
F=0

insert_At_Rear(40)



F=0
R=3

insert_AT_Rear(50)

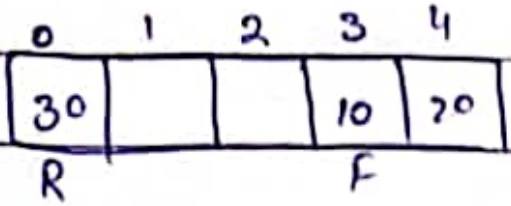


F=0
R=4

insert_At_Rear

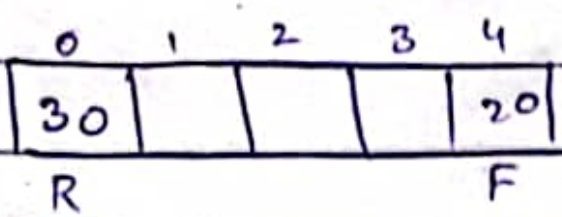
(F=0 && R=n-1) then return.

⇒ Deletion At Front



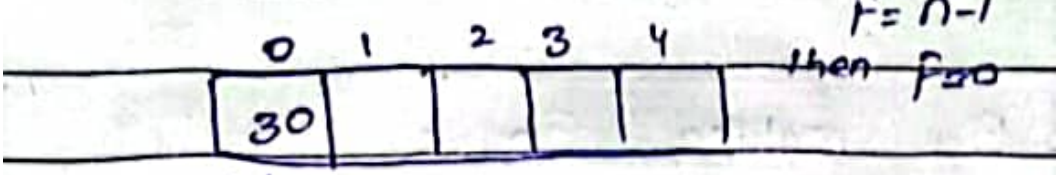
F=3
R=0

Delete_At_Front()



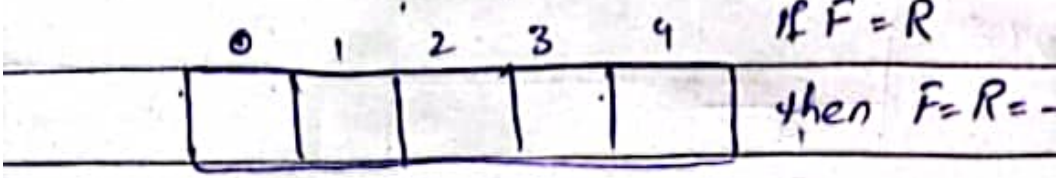
F++
F=4

delete_At_Front ()



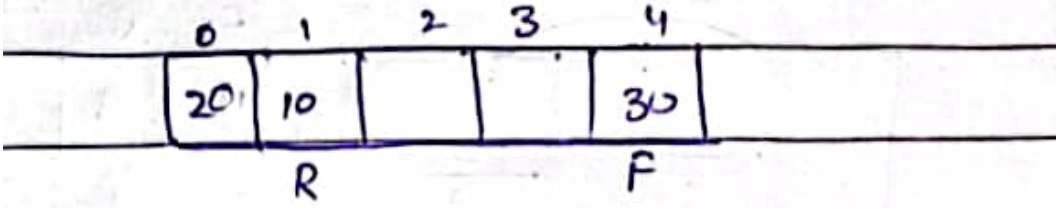
F = n-1
then f=0

delete_At_Front ()

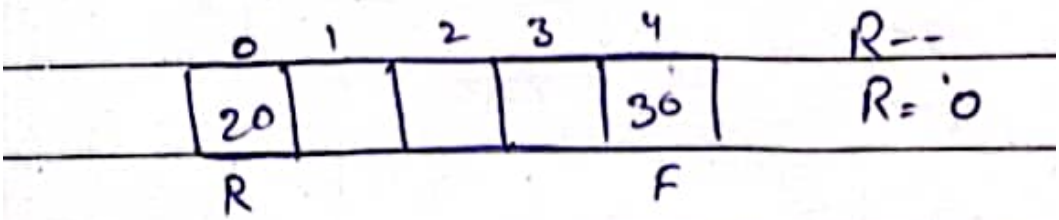


If F = R
then F = R = -

Delete At Rear

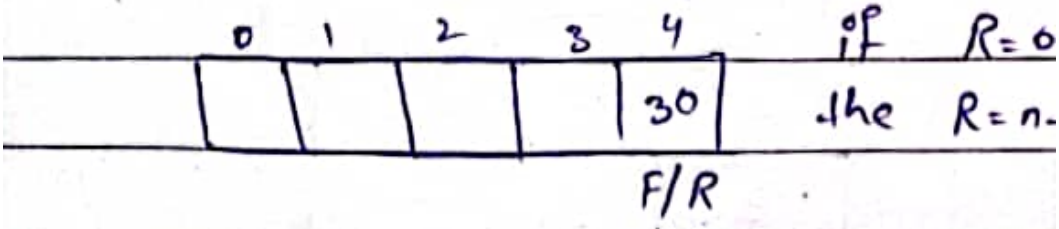


delete At Rear ()



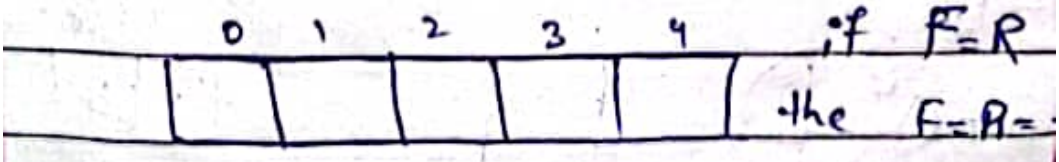
R--
R = 0

delete_At_Rear ()



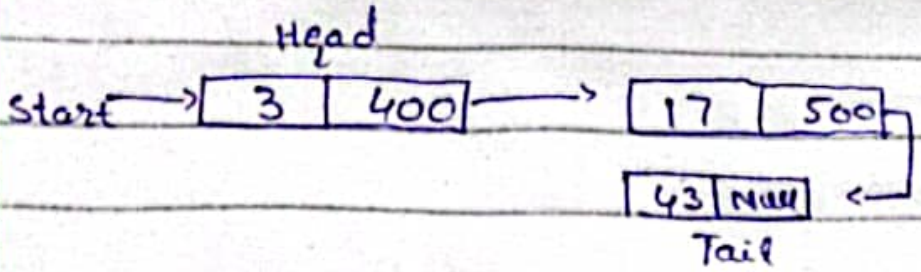
if R = 0
the R = n-

delet_At_Rear ()

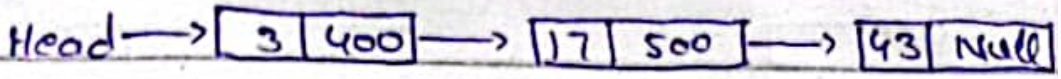


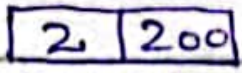
if F = R
the F = R =

Enqueue:

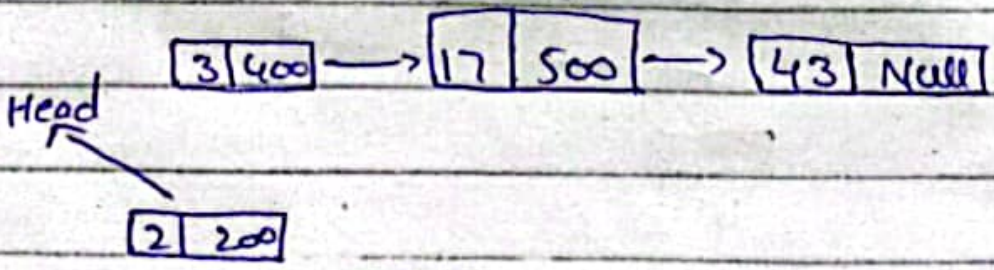


Add 2 in que.

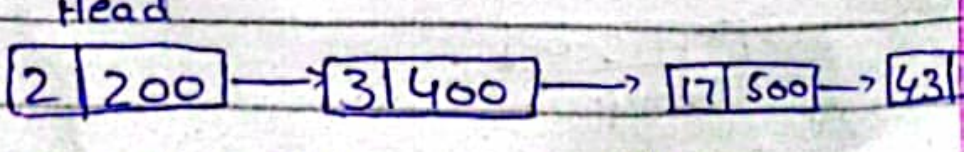




New Node = Current



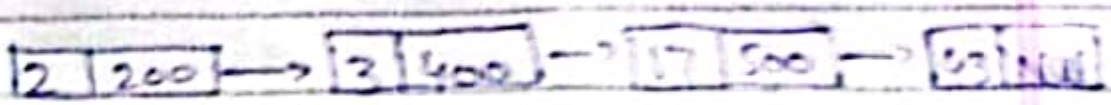
current => next

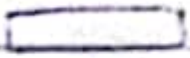


current -> next = new node.

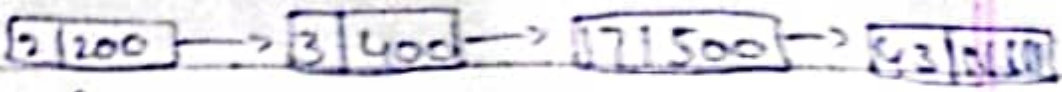
Dequeue:

Deque 17.



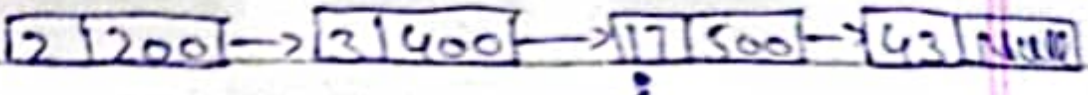


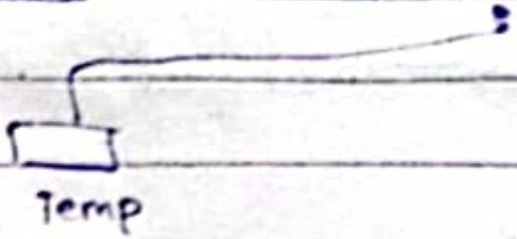
Node * Temp



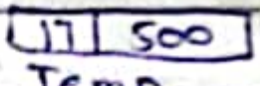
Node = Temp

Now traverse the qiue.





Temp



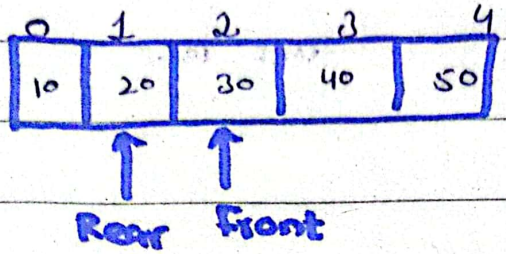
Temp

Temp → delete

Temp

Is Full Function:

```
If ((rear+1) %N == Front) {
    cout << " Queue is full " << endl; }
```



((rear +1) %N == Front)

(1+1) %N == 2

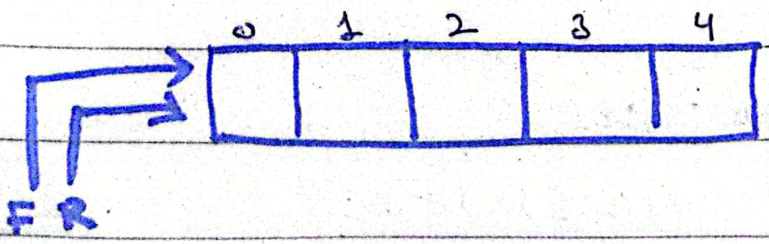
2 %5 == 2

2 == 2

Circular Queue is full.

Is empty:

(Front == -1 && rear == -1)



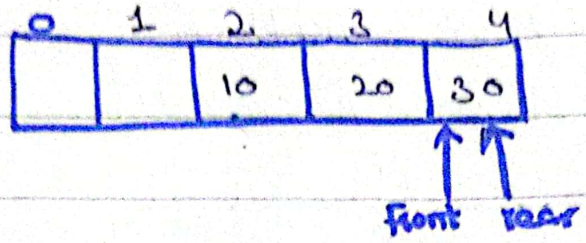
Front = -1

Rear = -1

Circular Queue is empty.

Enqueue Function

rear = (rear + 1) % N

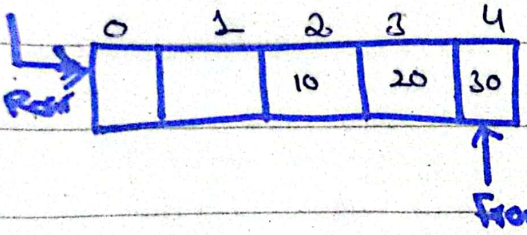


rear = (rear + 1) % N

= (4 + 1) % 5

= 5 % 5

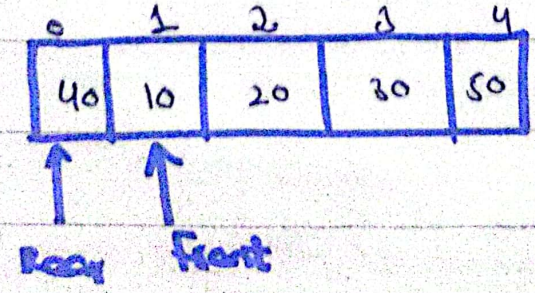
= 0



then we enqueue the value in the circular queue at zero index.

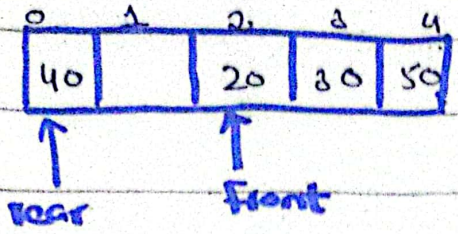
Dequeue Function

Front = (Front + 1) % N



(Front + 1) % N

(1 + 1) % 4

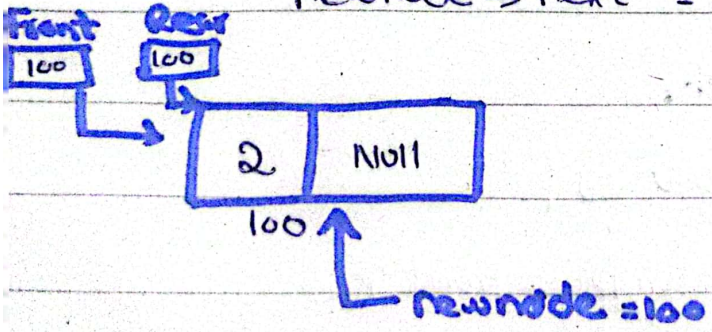


So, we dequeue the element 10 from one index and Front pointer move to the 2 index.

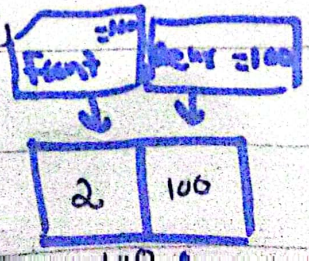
Circular Queue using linked list

Enqueue function

```
void enqueue(int x){
    Node * newnode = new node;
    newnode -> data = x;
    newnode -> next = NULL;
```

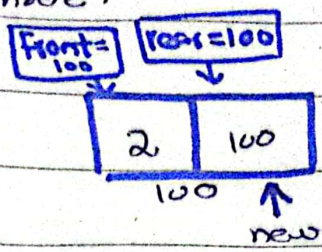



```
if (rear = = NULL)
    Front = rear = newnode;
    rear -> next = Front;
```

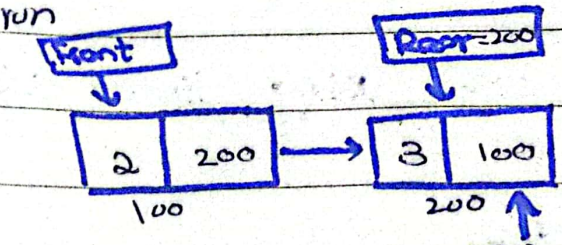


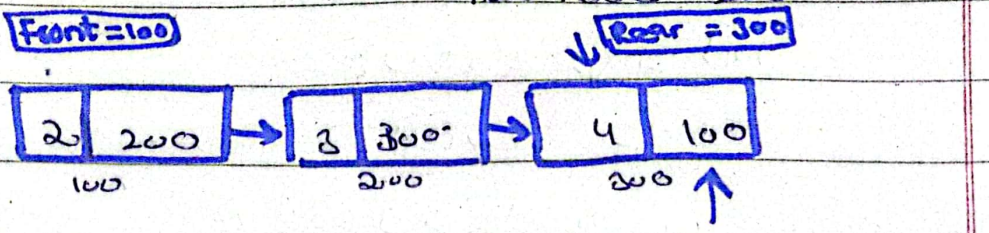
Date:

Thus we make a linklist which has only one node if we insert another node.



in this condition the rear is not equal to NULL. So else condition run

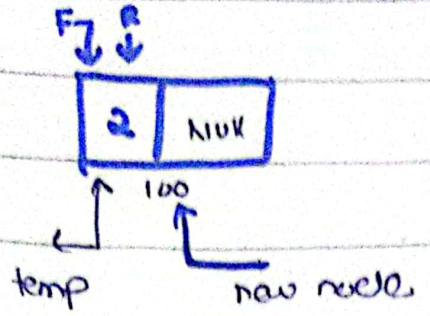




Dequeue function:

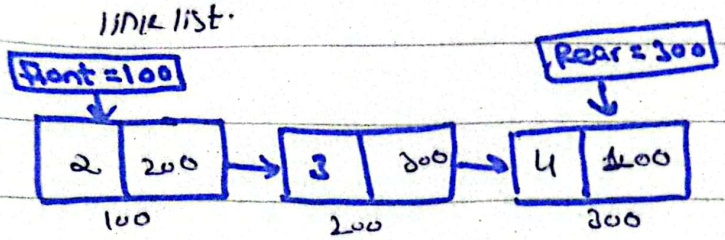
```
void dequeue (){
    Node * temp = front;
if (front == NULL && rear == NULL){
    cout<< "Queue is empty";
}
else if (front == rear){
    front = rear = NULL;
    delete temp;
```

when we have only one node in the link list



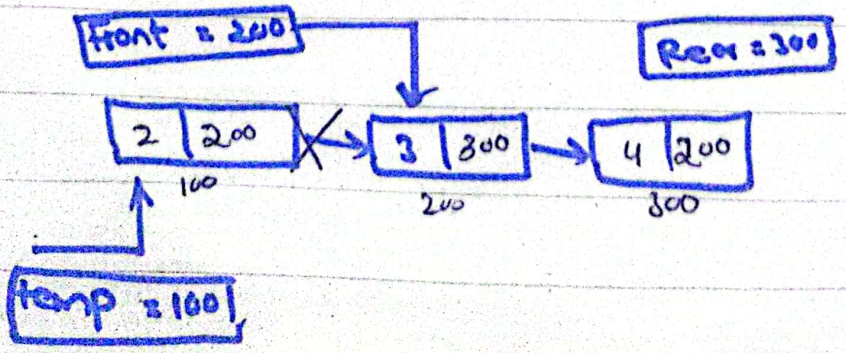
Then we simple point the Front or rear pointer to null and delete the temp pointer.

if we have a multiple node in link list.



Now Front and rear are not null then else condition run

```
else {
    Front = front -> next
    rear -> next = Front;
    delete temp;
}
```



Date: _____

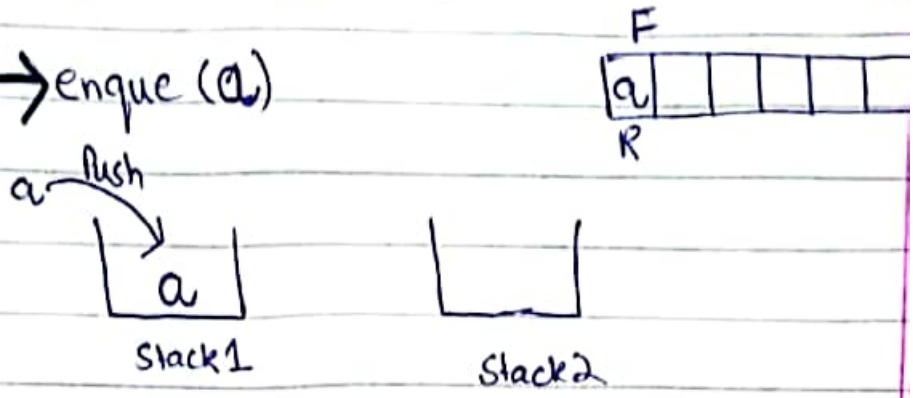
front point to the second node and we delete the temp which point the first node. so first node is deleted.

Queue using Stack

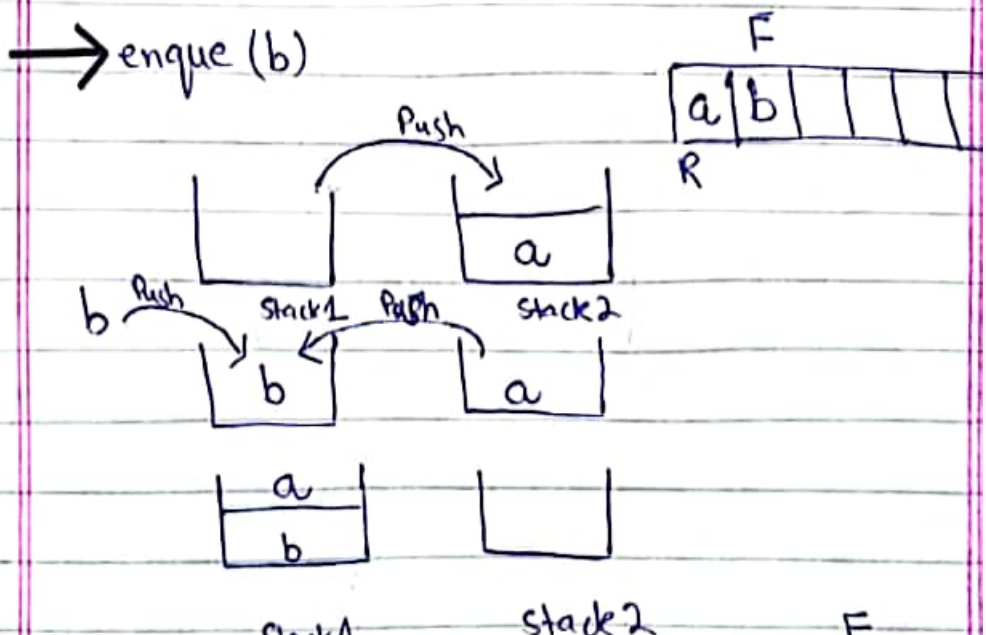
↳ FIFO (First in First Out)

1) By making Enqueue operation costly

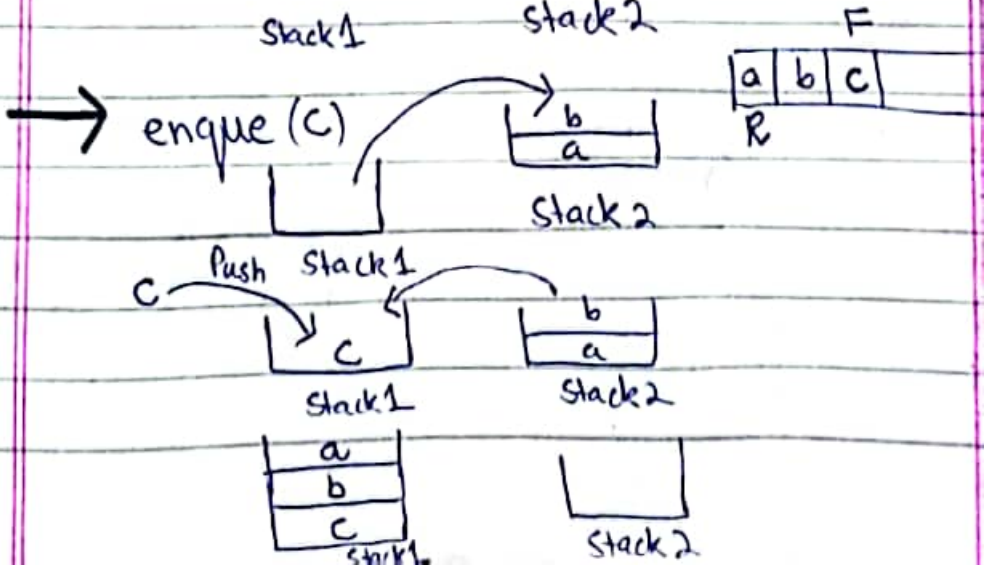
→ enque (a)



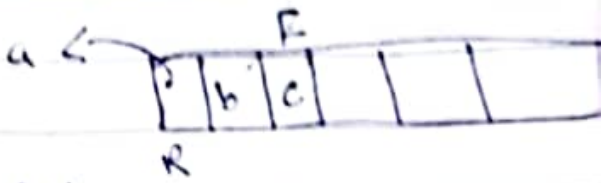
→ enque (b)

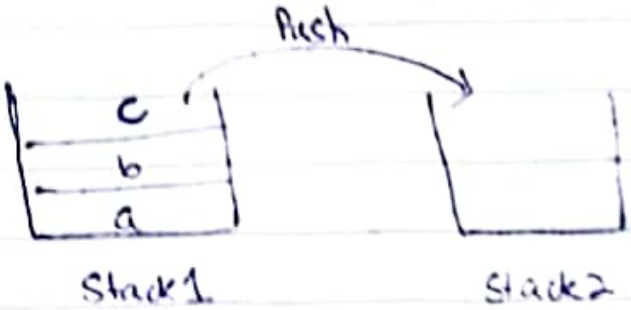


→ enque (c)

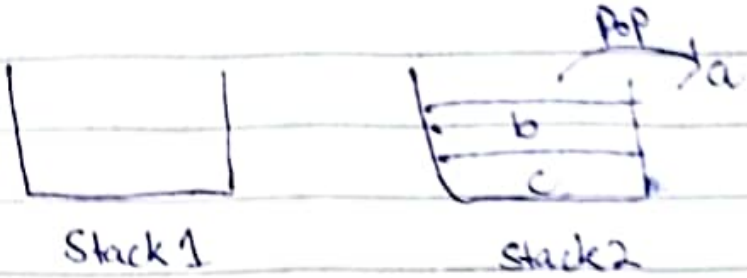


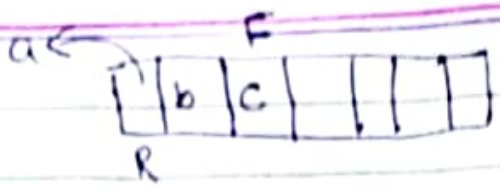
→ Dequeue


a
F
a
b
c
R

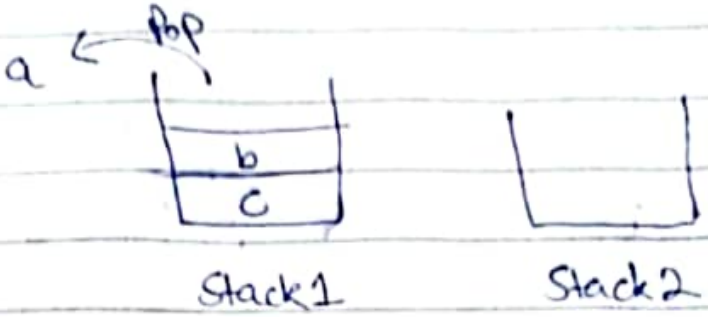

Push
c
b
a
Stack 1
Stack 2


a
b
c
Stack 1
Stack 2


Pop
a
b
c
Stack 1
Stack 2

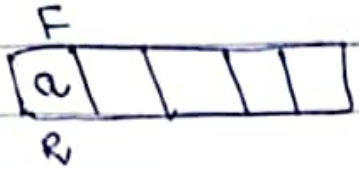


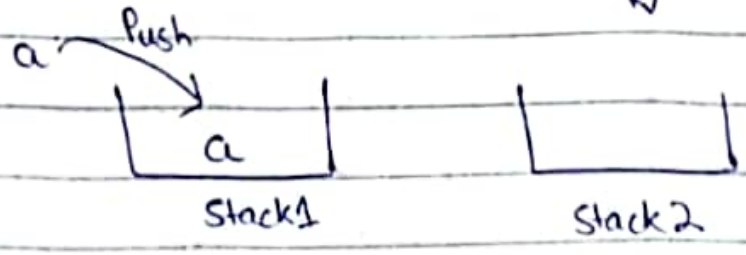
→ Dequeue



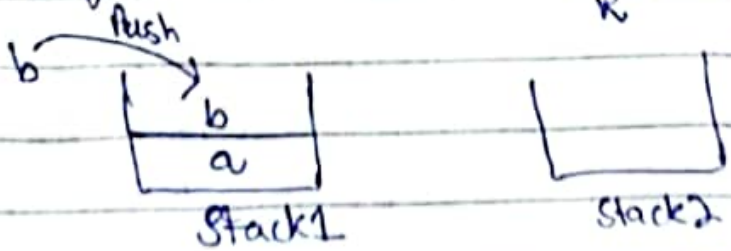
2) By Making Dequeue operation costly

→ enqueue (a)





→ equeue (b)



→ enqueue (c)