Queue DS

Stack :- Linear DS

LIFO

implement = array / Linked list

top

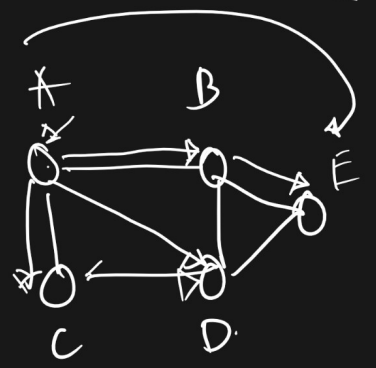
{ push ⇒ insert ⇒ top++ ⇒ arr[top] = num
{ pop ⇒ remove ⇒ arr[top] ⇒ top--

* Queue :-

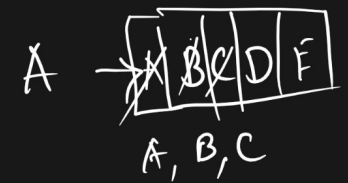
⇒ Linear Data Structure

⇒ Principle ⇒ FIFO

⇒ Two ends — insert — enqueue
remove — dequeue



BFS → Queue



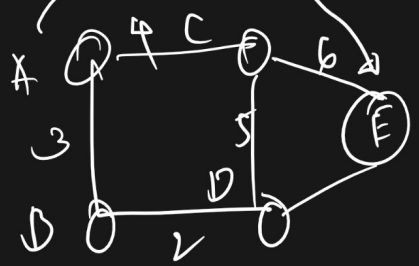
Appl^n — Ticket Booking System

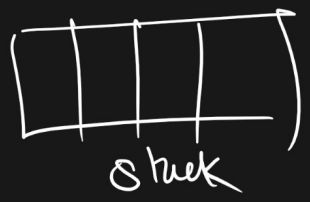
— Multitasking

— task scheduling

— graph traversing

↳ BFS ⇒ queue
↳ DFS ⇒ stack





Queue

Implement
- Array ✓
- Linked List ✓

Using Array :-

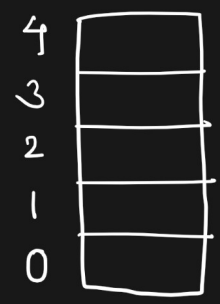
- size of array = 5

int arr = new int [5];

Two ends

int rear = -1

int front = -1



When array is empty

r & f = -1

r = f = -1

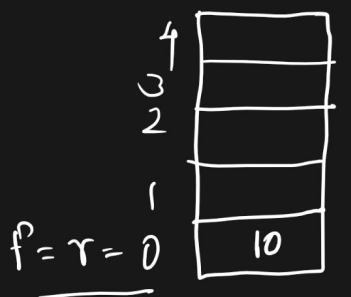
Operation :-

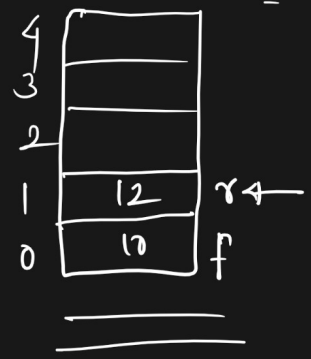
1) Insert / Enqueue :- (10)

```
if (front = rear = -1)
{  front ++; arr[rear] = num;
   rear ++;
}
else
{  rear ++;
   arr[rear] = num;
}
```

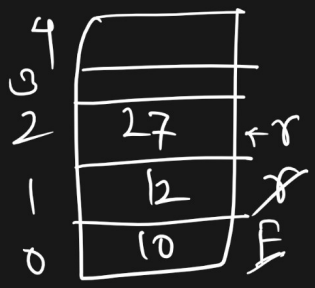
```
if (isfull())
T
{  "Queue is full"
}
else
{
}
```

enqueue(10) enqueue(12)



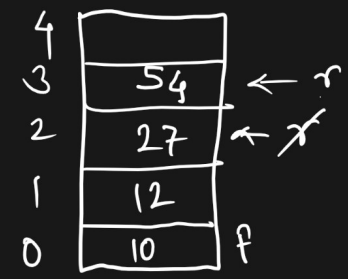


enqueue (27)



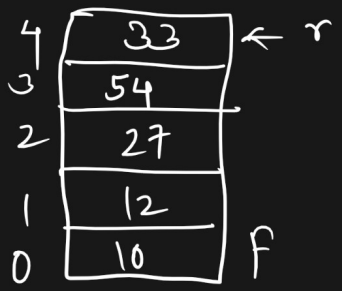
10 12 27

enqueue (54)

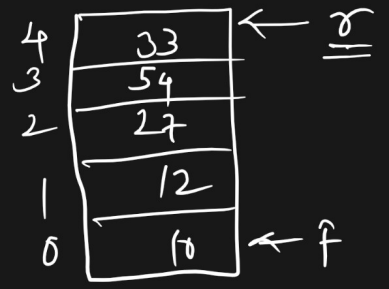


-1

enqueue (33)



enqueue (71)




```
boolean isFull() {
    if (rear = size-1)
        return true;
    else
        return false;
}
```

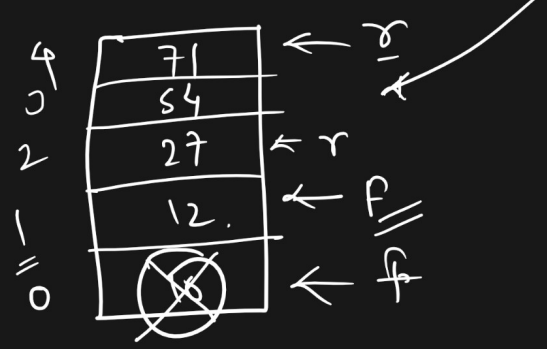
arr.length

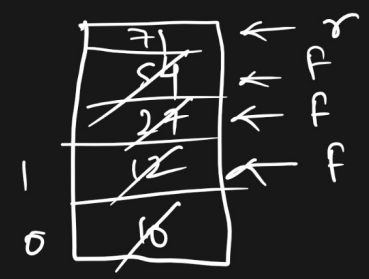
```
void enqueue() {
    if (isFull())
        { S.o.p. ("Queue is full"); }
    else {
        if (f = r = -1)
            { f++;
              r++;
            } arr[rear] = num;
        else {
            rear++
            arr[
```

A) insert (enqueue) — end: rear — rear++ ⇒ arr[rear] = num

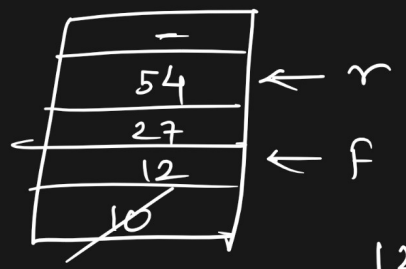
B) remove (dequeue) — end: front — arr[front] ⇒ front++

dequeue () num = arr[front] ; front ++





FIFO

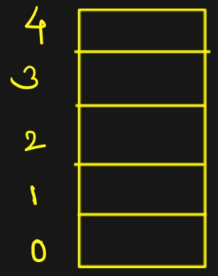


12 27 54

```
for(int i = front; i<= rear; i++)
{
}  S.op.(arr[i]);
```

---

arr



f = -1, r = -1

A) enqueue(37)

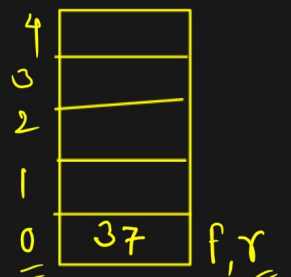
i) isfull() → F

f = -1, r = -1

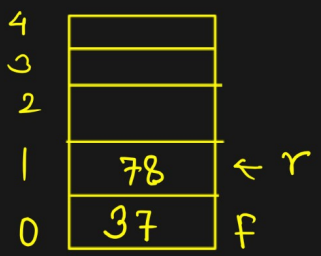
↓ ↓

0 0

f++ r++



B) enqueue(78)

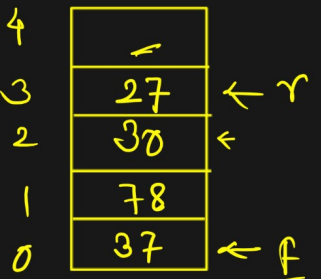


rear++

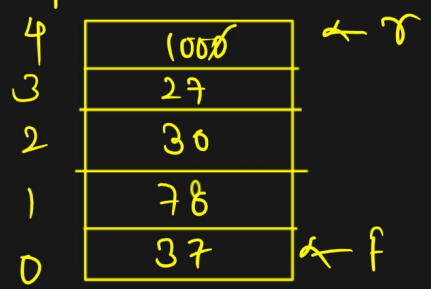
arr[rear] = 78

enqueue(30) ✓

enqueue(27) ✓

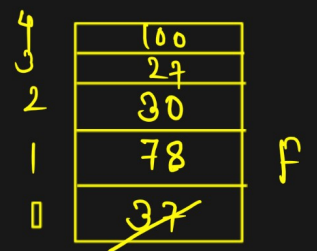


enqueue(100) r++



✗ enqueue(50) ✗

✓ dequeue() ⇒ arr[front] ⇒ front++

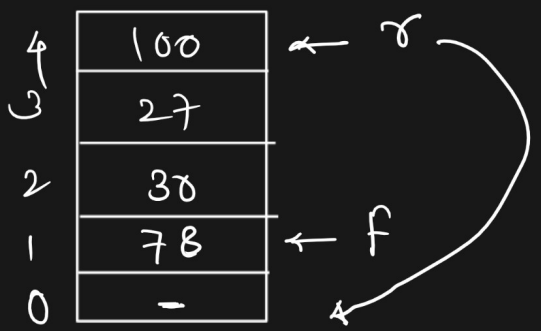


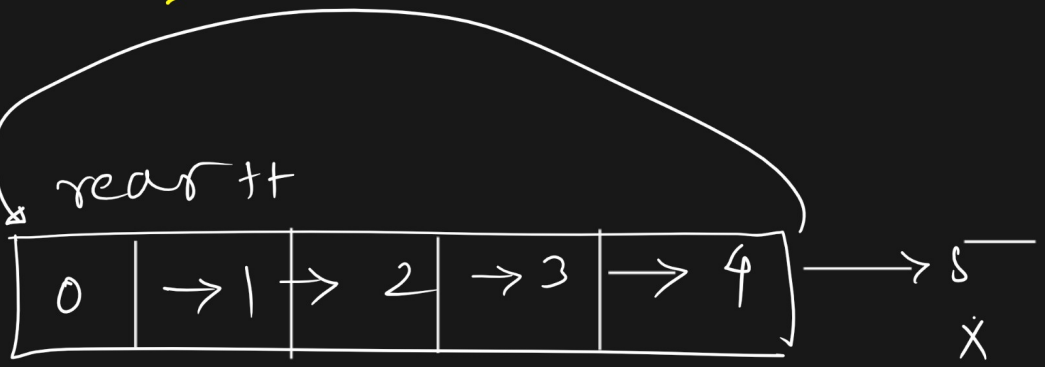
display()

78 30 27 100

enqueue(50) 5 ✗

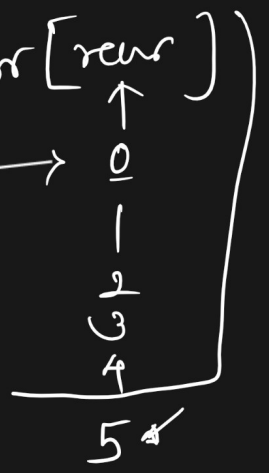
rear++





-1

+1



Circular Queue :-

insert ⇒ (enqueue)

end

rear = (rear+1) % arr.length

$0 \% 5 = 0$ $1 \% 5 = 1$ $2 \% 5 = 2$ $3 \% 5 = 3$ $4 \% 5 = 4$

$5 \% 5 = 0$

rear = -1

1st ele rear = (rear + 1) % 5
= (-1 + 1) % 5 = 0 % 5 = 0

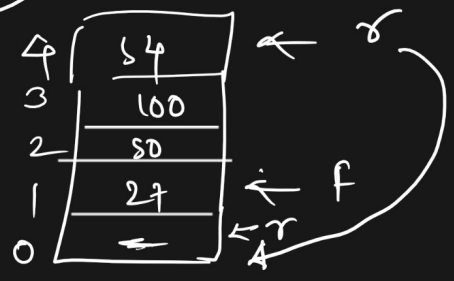
2nd ele rear = (rear + 1) % 5 = 1 % 5 = 1
(rear = 0)

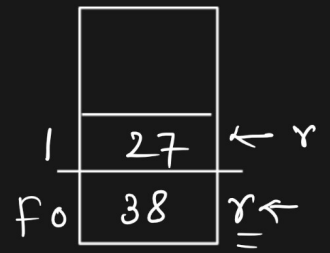
3rd rear = (rear + 1) % 5 = 2 % 5 = 2

4th rear = (rear + 1) % 5 = 3 % 5 = 3

5th rear = (rear + 1) % 5 = 4 % 5 = 4

enqueue 5 times

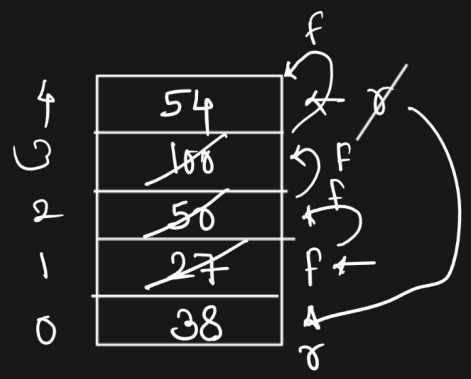




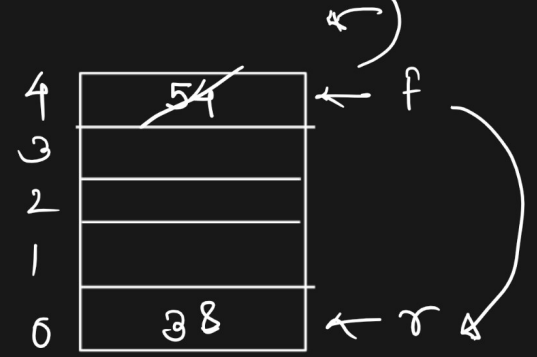
f = 1
r = -1

dequeue () → front ++

enqueue rear = (rear + 1) % 5 = 5 % 5 = 0
(rear = 4)



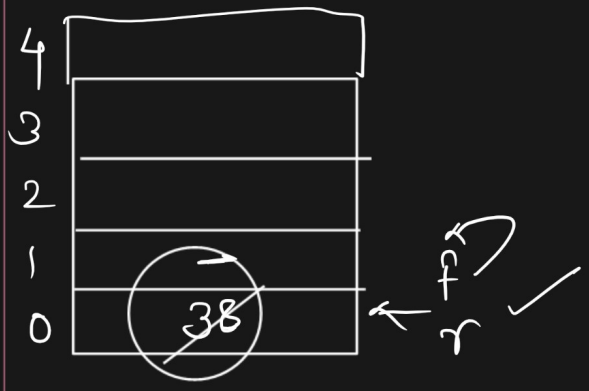
dequeue()
dequeue()
dequeue()



dequeue()

~~front = front + 1~~

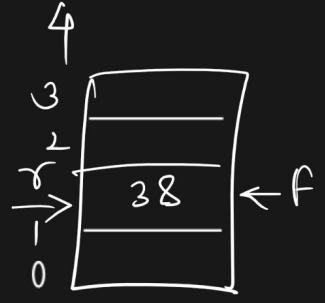
front = (front + 1) % arr.length;



r = f = -1

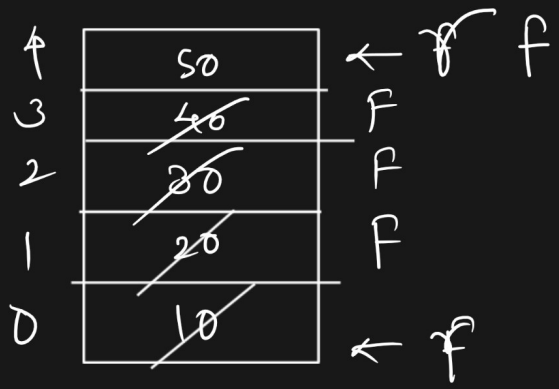
dequeue()

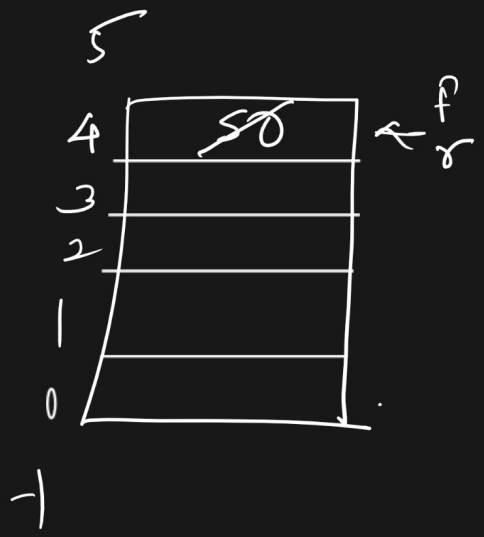
Cond? if (f == r)



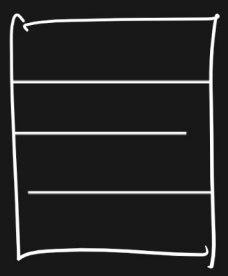

```
dequeue()
{
  if (isEmpty())
  { ——— }
  else
  {
    arr[front] -> 38
    front = (front + 1) % size
  }
}
```

```
if (f == r)
  f = r = -1;
```





dequeue()

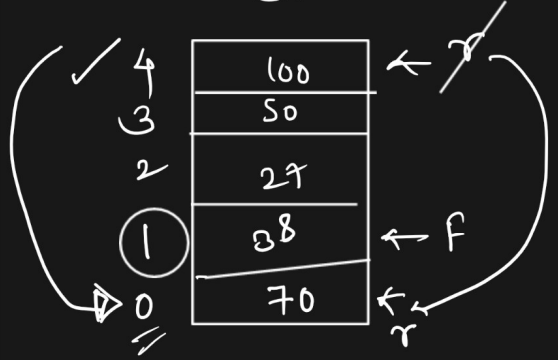


```
if (f==r)
{
    f=r=-1;
}
else
{
    front=(front+1)%size;
}
```

front = -1, rear = -1

isfull() ?

CircularQueue



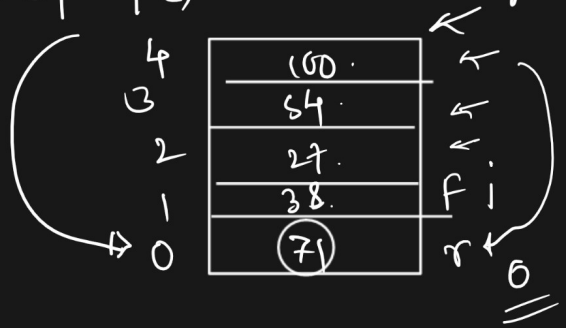
enqueue(101)

isFull((rear+1) % ar.length == f)

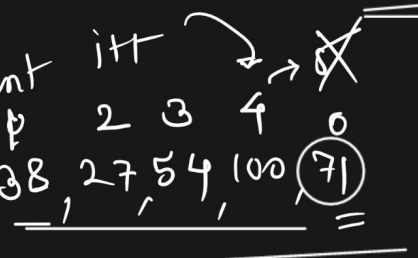
0+1 5

1 % 5 == 1

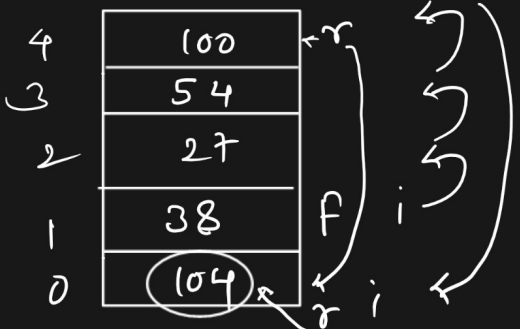
Display() :-



i = front



i = (i+1) % size



```
if (i == rear)
        break;
else
    i=(i+1) % size;
```

Display ⇒ 38, 27, 54, 100

enqueue(104)

display ⇒ 38, 27, 54, 100, 104