Data Structures – CST 201
Module - 2

Syllabus

- Polynomial representation using Arrays
- Sparse matrix
- Stacks
  - Evaluation of Expressions
- **Queues**
  - Circular Queues
  - Priority Queues
  - Double Ended Queues,
- Linear Search
- Binary Search

QUEUE

- Queue is a linear data structure
- Queue is an ordered collection of homogenous data elements where the insertion and deletion takes place at two extreme ends called as front end and rear end
- The data in queue is processed in the same order as it had entered.
- So it is a **First In First Out- FIFO** Memory

QUEUE- Real Time Applications

- Queuing in front of a counter
- Traffic control at a turning point
- Process synchronization in multi-user environment
- Resource sharing in a computer centre

Queuing in front of a counter

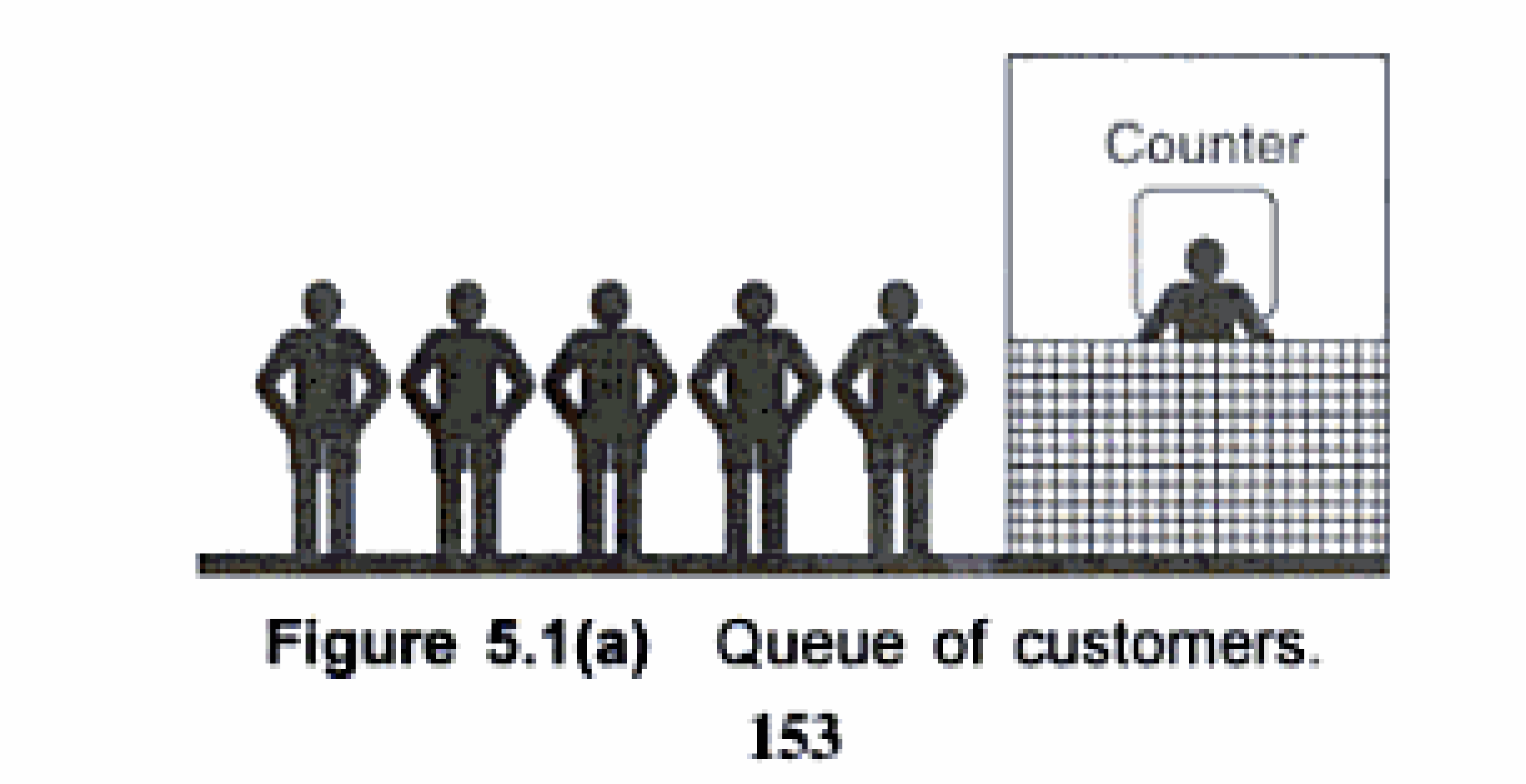


Figure 5.1(a) Queue of customers.

153

Traffic control at a turning point

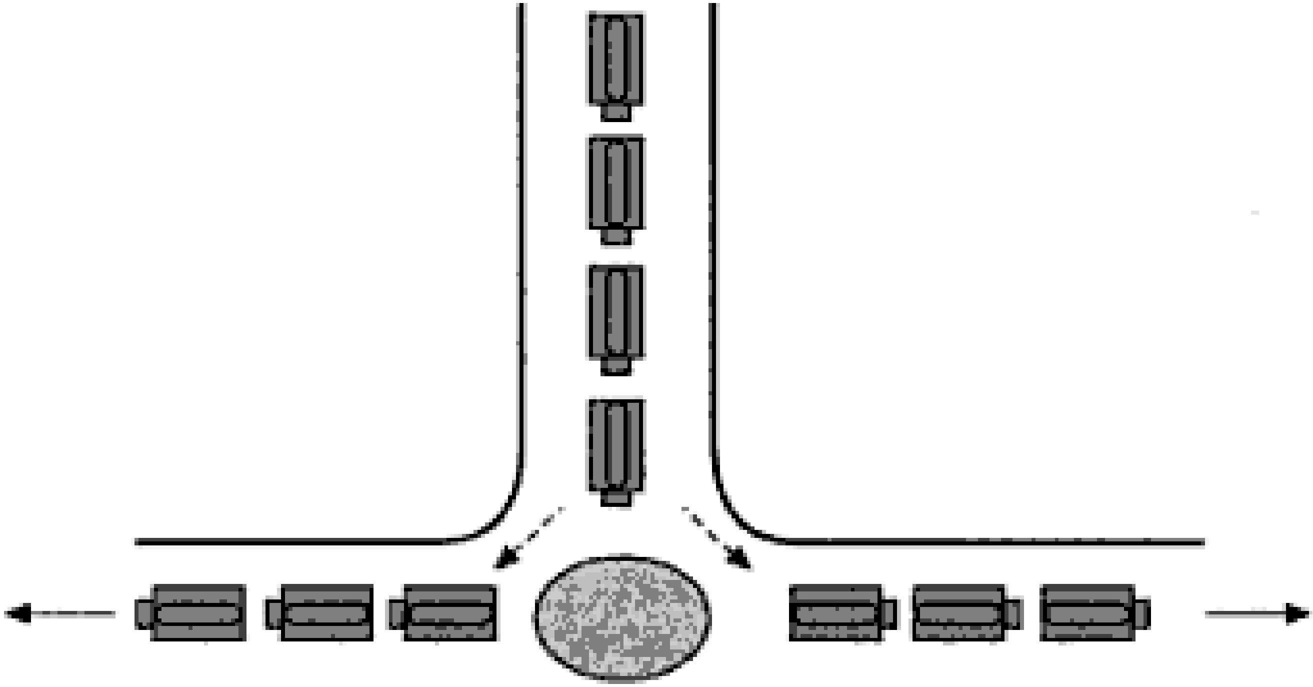


Figure 5.1(b) Traffic passing at a turning point.

Process synchronization in multi-user environment

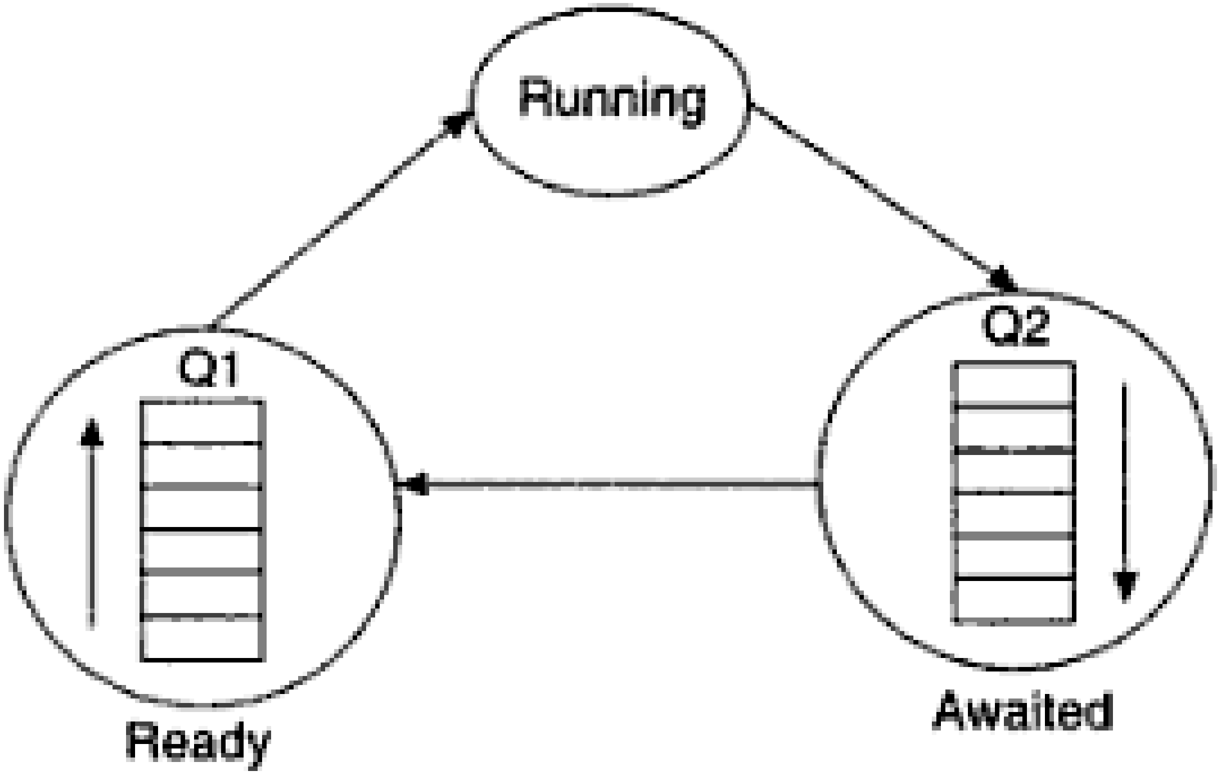


Figure 5.1(c) Queues of processes.

Resource sharing in a computer centre

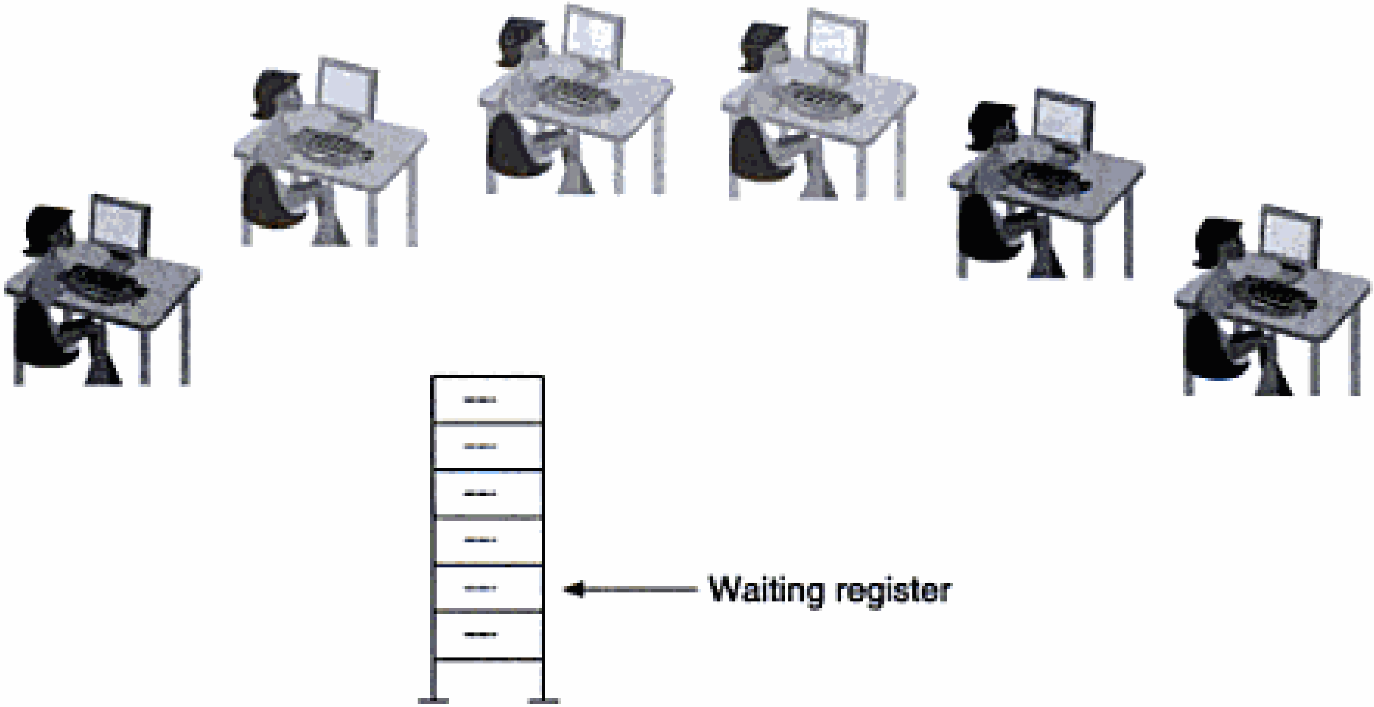


Figure 5.1(d) A waiting queue of users in a computer centre.

QUEUE- Basic Terminologies

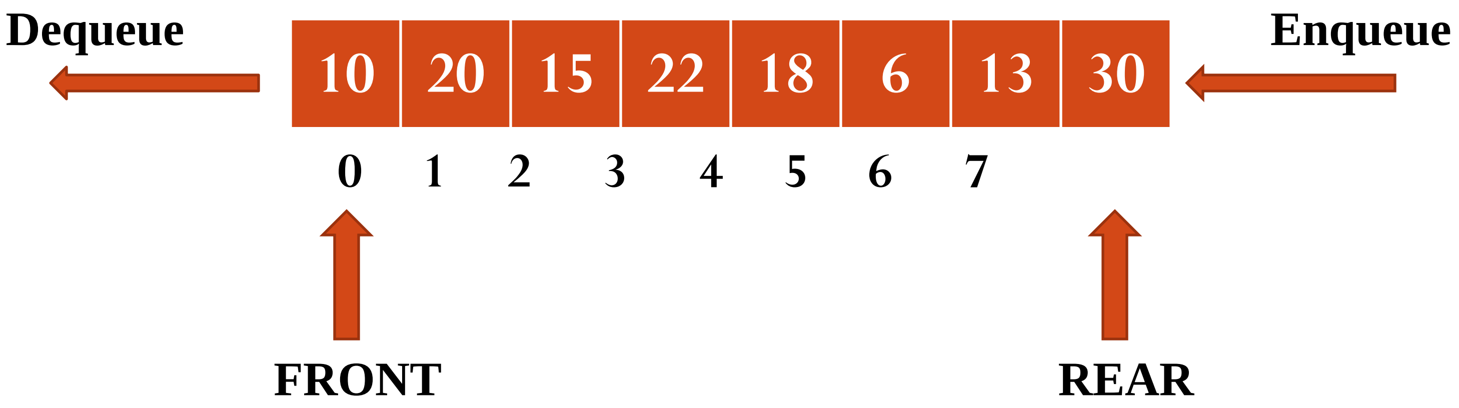
- **ENQUEUE**- Insertion in the QUEUE
- **DEQUEUE**-Deletion in the QUEUE
- **REAR**- Where INSERTION takes place
- **FRONT**-Where DELETION takes place
- **ITEM**- An Element in QUEUE
- **LENGTH / SIZE**- Total Number of elements that queue can accommodate

QUEUE- Operations

- **ENQUEUE**: Insert an element into Queue
- **DEQUEUE**: Delete an element from the Queue
- **DISPLAY**: Display the contents of the Queue

QUEUE- Representations

- Two Representations
  - Array Representation



Dequeue

10 20 15 22 18 6 13 30

Enqueue

0 1 2 3 4 5 6 7

FRONT

REAR

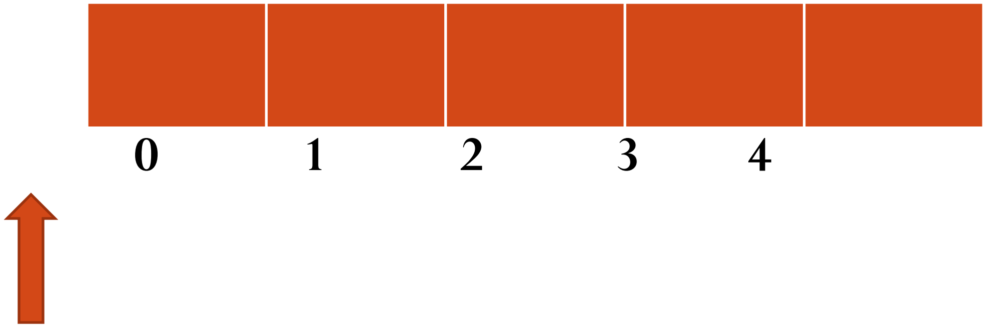
  - Linked List Representation

QUEUE – ENQUEUE Algorithm

int A[5];

If FRONT=-1 Or REAR=-1 then

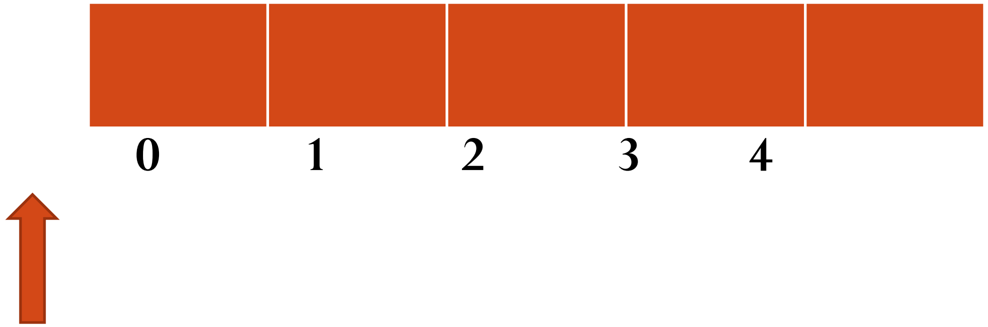
Queue is EMPTY



FRONT=-1

REAR=-1

ENQUEUE 10



0 1 2 3 4

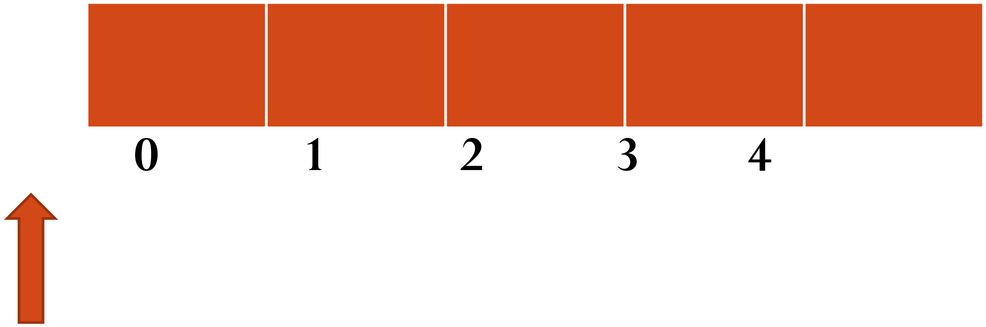
FRONT=-1

REAR=-1

FRONT = 0

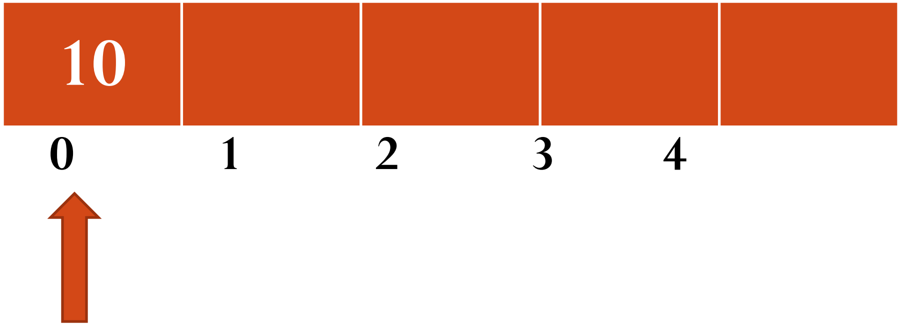
REAR = REAR + 1

A[REAR] = 10



ENQUEUE 10

FRONT=-1
REAR=-1



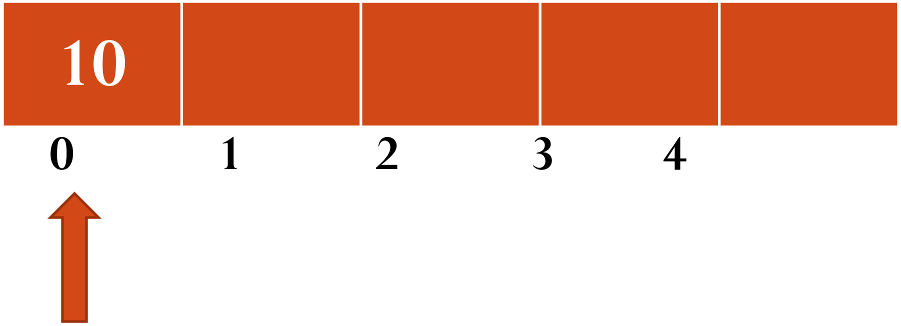
10

0 1 2 3 4

FRONT=0

REAR=0

ENQUEUE 20



10

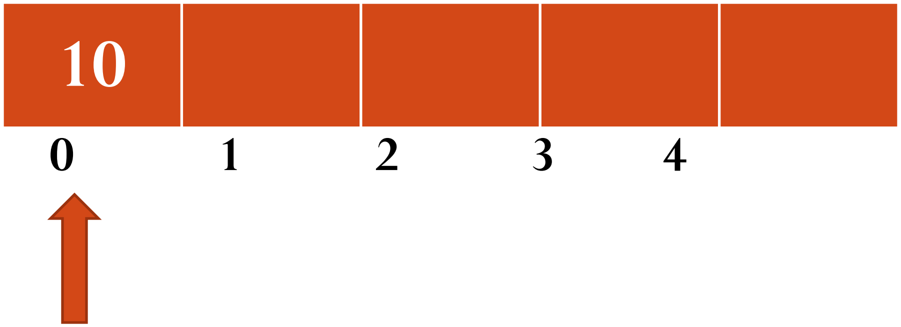
0 1 2 3 4

FRONT=0

REAR=0

REAR = REAR + 1

A[REAR] = 20



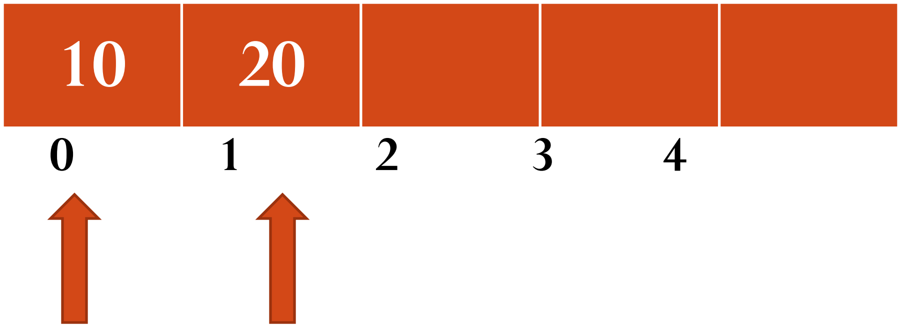
ENQUEUE 20

FRONT=0

REAR=0

REAR = REAR + 1

A[REAR] = 20



10 20

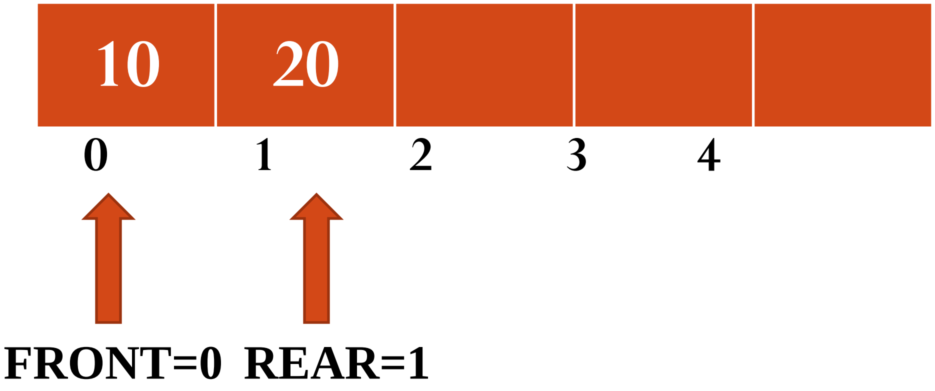
0 1 2 3 4

FRONT=0 REAR=1

ENQUEUE 20

REAR = REAR + 1

A[REAR] = 30



10 20

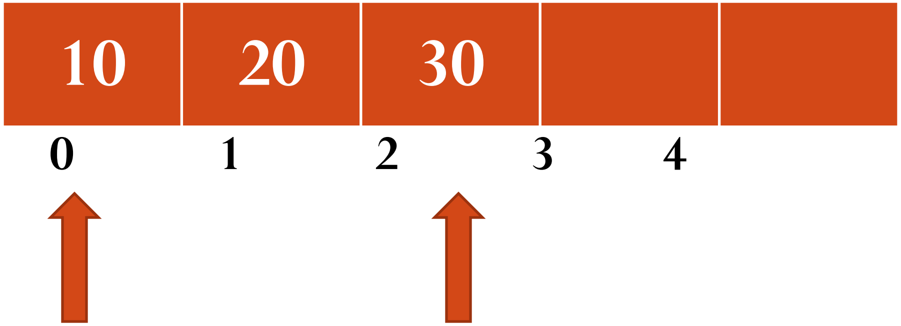
0 1 2 3 4

FRONT=0 REAR=1

ENQUEUE 30

REAR = REAR + 1

A[REAR] = 30



10	20	30		
0	1	2	3	4

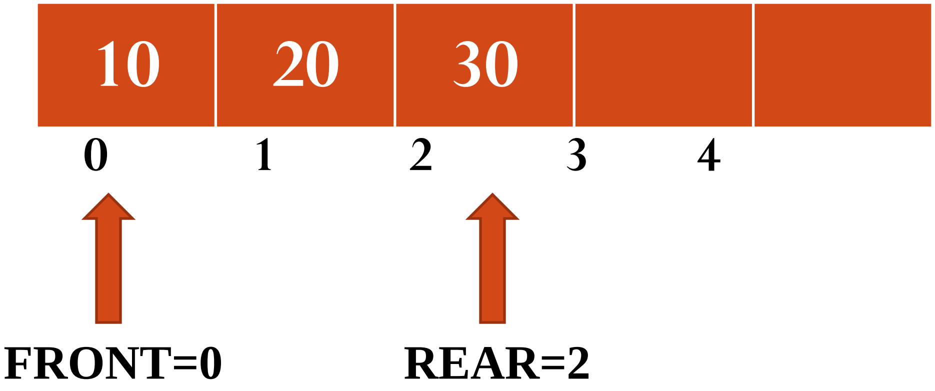
FRONT=0

REAR=2

ENQUEUE 30

REAR = REAR + 1

A[REAR] = 40



10	20	30		

0 1 2 3 4

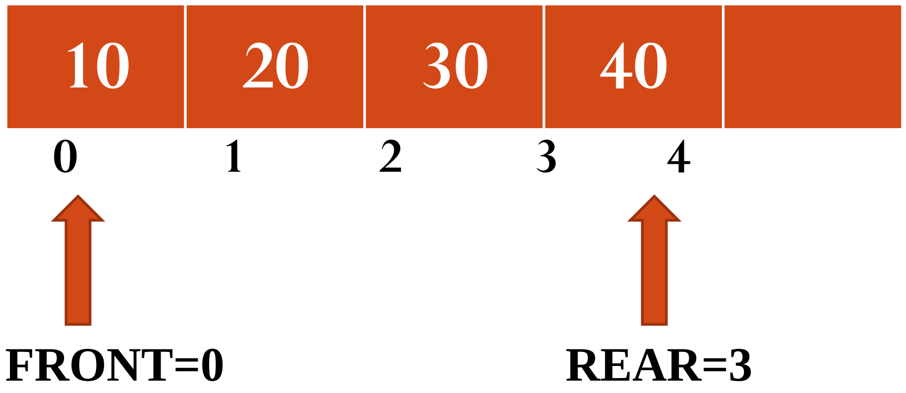
FRONT=0

REAR=2

ENQUEUE 40

REAR = REAR + 1

A[REAR] = 40



10	20	30	40	

0 1 2 3 4

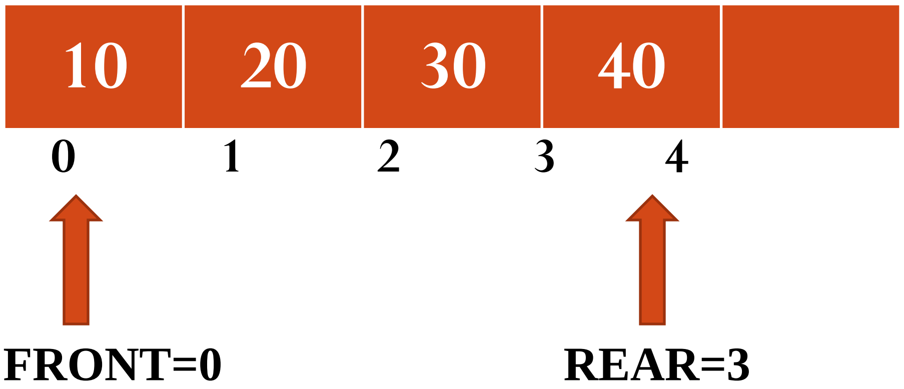
FRONT=0

REAR=3

ENQUEUE 40

REAR = REAR + 1

A[REAR] = 50



10 20 30 40

0 1 2 3 4

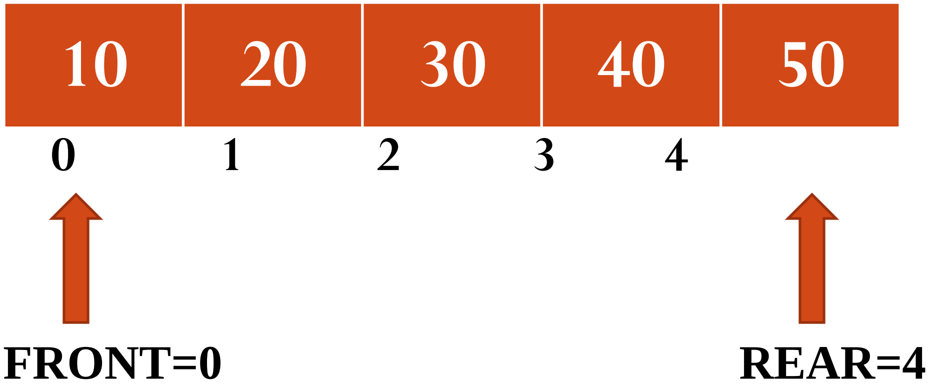
FRONT=0

REAR=3

ENQUEUE 50

REAR = REAR + 1

A[REAR] = 50



10 20 30 40 50

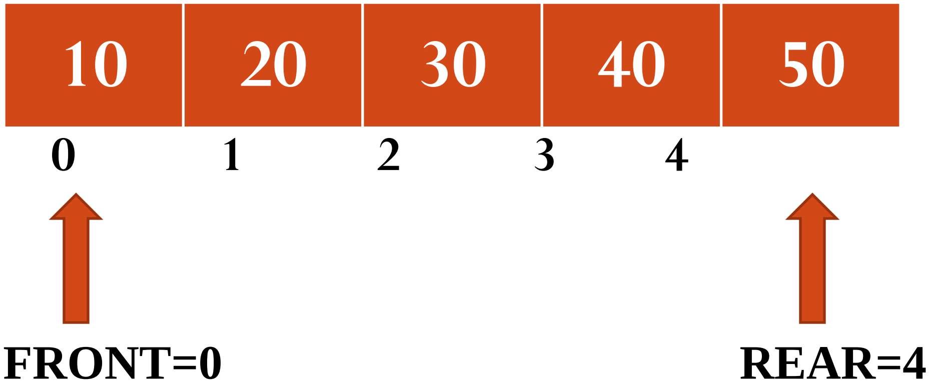
0 1 2 3 4

FRONT=0

REAR=4

ENQUEUE 50

```
If REAR = SIZE – 1 then
        Print “Queue is FULL”
```



10	20	30	40	50
0	1	2	3	4

FRONT=0

REAR=4

ENQUEUE 60

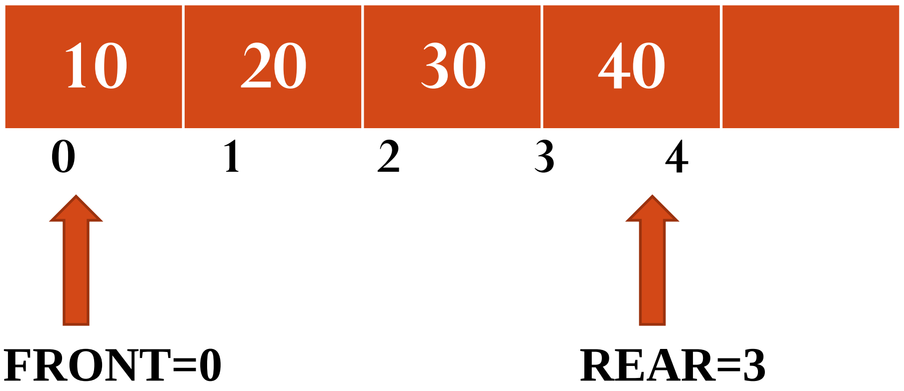
QUEUE – ENQUEUE

Algorithm ENQUEUE(ITEM)

```
{       if  REAR = SIZE – 1 then
                Print “Queue is FULL”
        else If REAR= -1 then           //Currently Queue is empty
        {       FRONT = 0
                REAR = 0
                A[REAR] = ITEM
        }
        else
        {       REAR = REAR + 1
                A[REAR] = ITEM
        }
}
```

QUEUE – DEQUEUE Algorithm

DEQUEUE



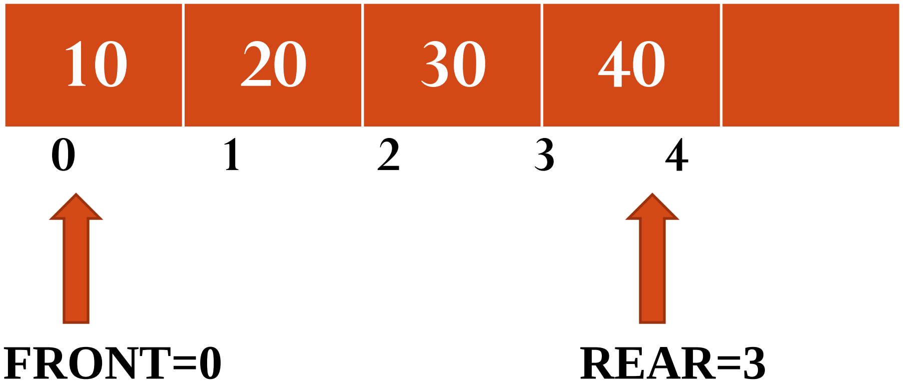
10 20 30 40

0 1 2 3 4

FRONT=0

REAR=3

FRONT = FRONT + 1



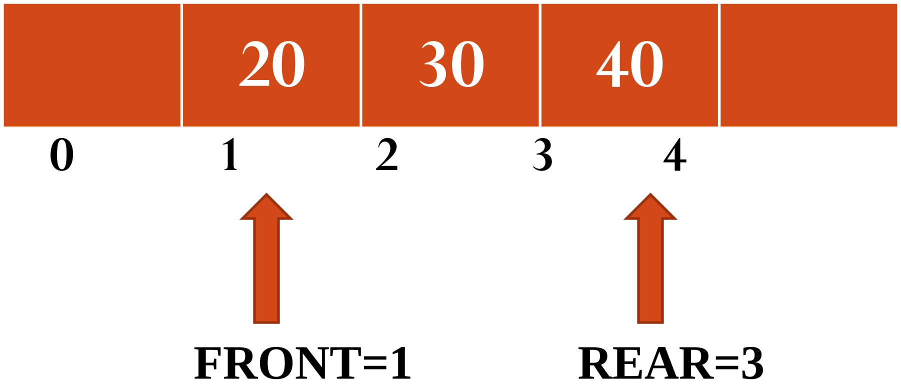
10	20	30	40	
0	1	2	3	4

FRONT=0

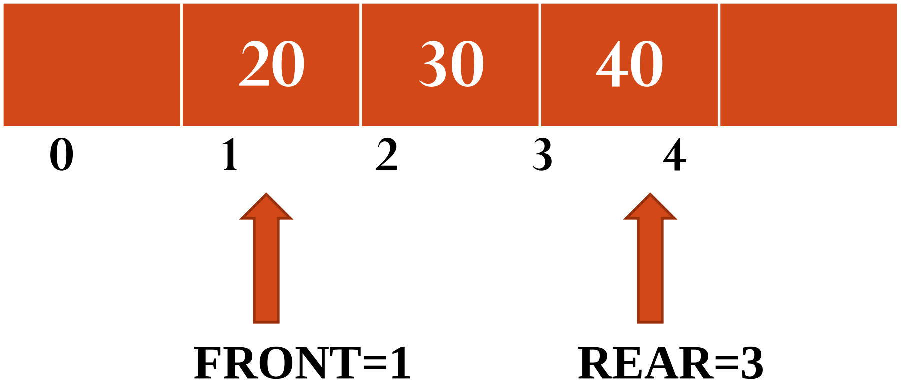
REAR=3

DEQUEUE

FRONT = FRONT + 1



FRONT = FRONT + 1



	20	30	40	

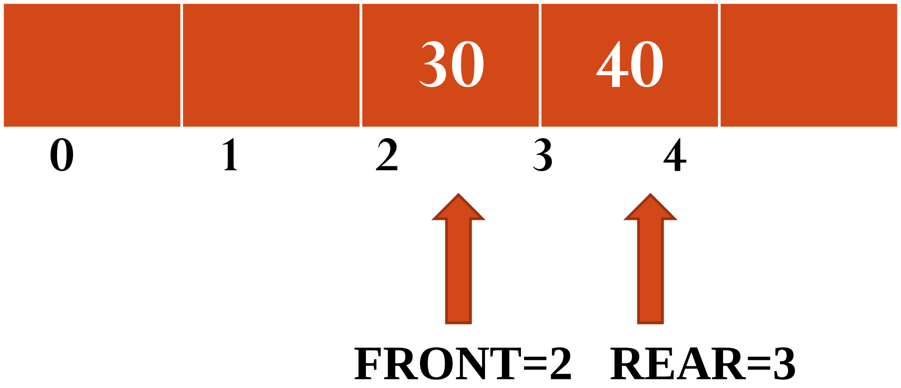
0 1 2 3 4

FRONT=1

REAR=3

DEQUEUE

FRONT = FRONT + 1



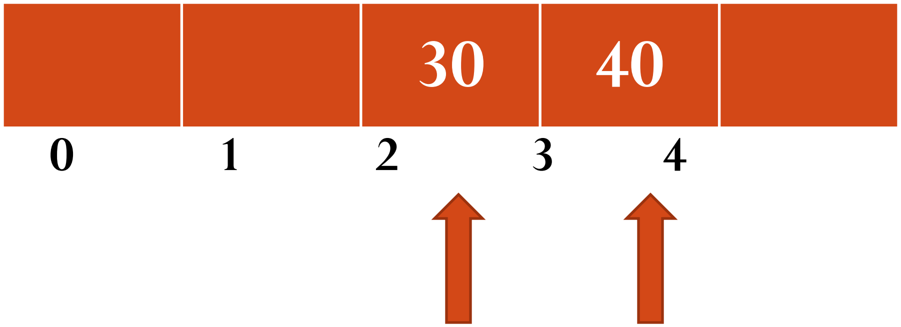
30 40

0 1 2 3 4

FRONT=2 REAR=3

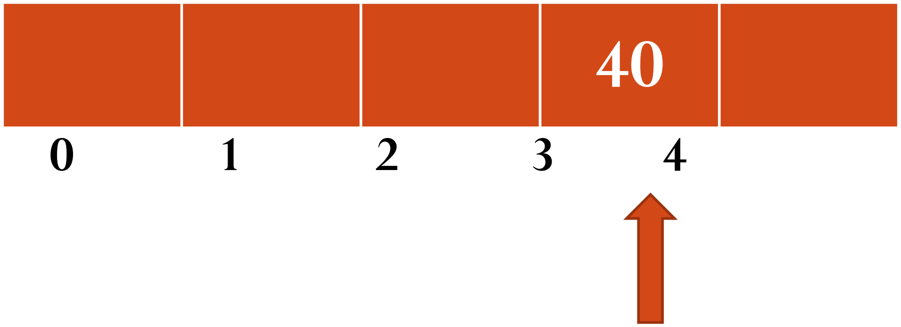
FRONT = FRONT + 1

DEQUEUE



FRONT=2 REAR=3

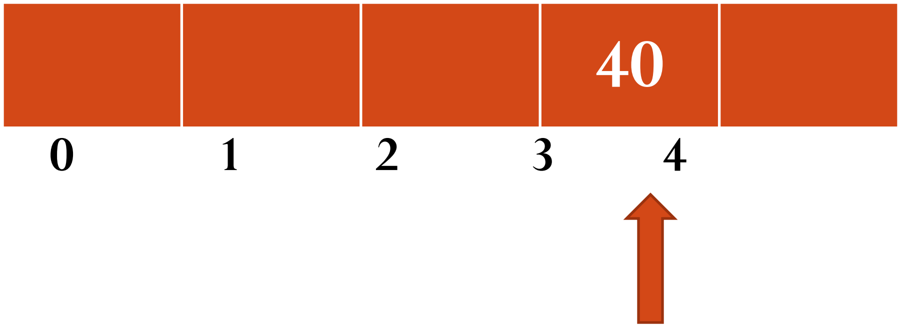
FRONT = FRONT + 1



			40	
0	1	2	3	4

FRONT=3
REAR=3

DEQUEUE

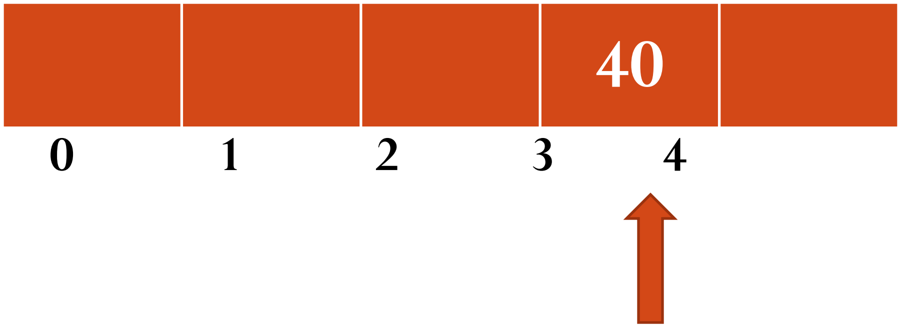


FRONT=3
REAR=3

If FRONT = REAR then

FRONT=-1

REAR = -1



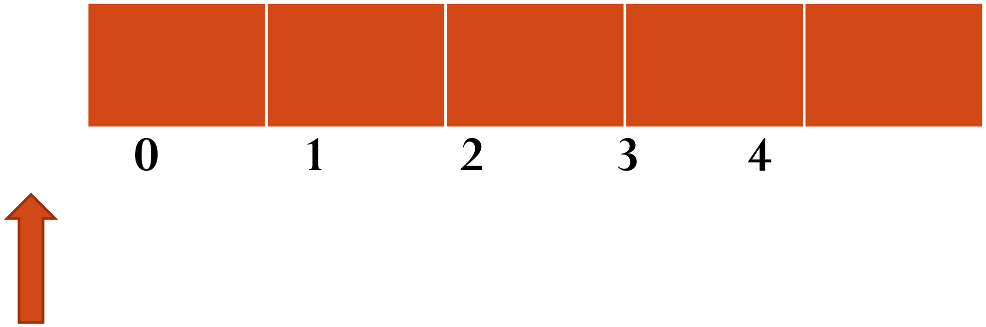
DEQUEUE

FRONT=3
REAR=3

If FRONT = REAR then

FRONT=-1

REAR = -1



FRONT=-1
REAR=-1

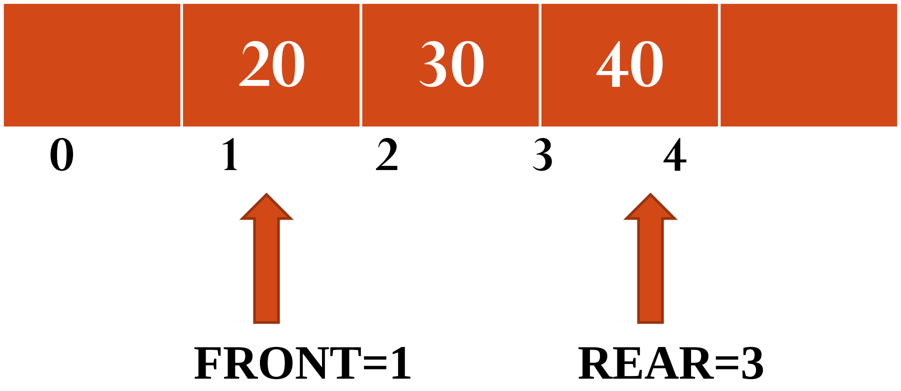
QUEUE – DEQUEUE

Algorithm DEQUEUE()

```
{         if  FRONT = – 1 then
                    Print “Queue is EMPTY”
          else if REAR = FRONT then  //Queue contains only one element
          {         Print “The deleted item is “ A[FRONT]
                    FRONT = REAR = -1
          }
          else
          {          Print “The deleted item is “ A[FRONT]
                    FRONT = FRONT + 1
          }
}
```

QUEUE – DISPLAY Algorithm

```
For i=FRONT to REAR do
        Print A[i]
```



QUEUE – DISPLAY

```
Algorithm DISPLAY()
{
        if  FRONT = – 1 then
                Print “Queue is EMPTY”
        else
        {
                for i=FRONT to REAR do
                        Print A[i]
        }
}
```

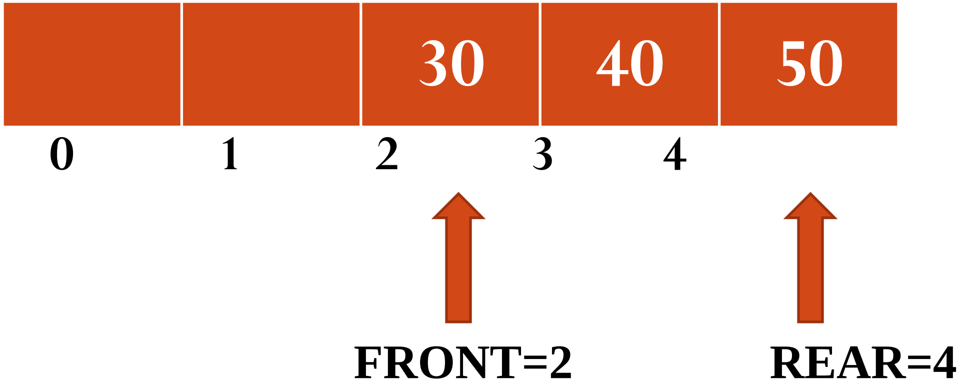
QUEUE- Various States

1. Queue is Empty: FRONT=-1 & REAR=-1

2. Queue is Full: REAR=SIZE – 1

3. Total elements in a queue =REAR - FRONT + 1

QUEUE- Disadvantage



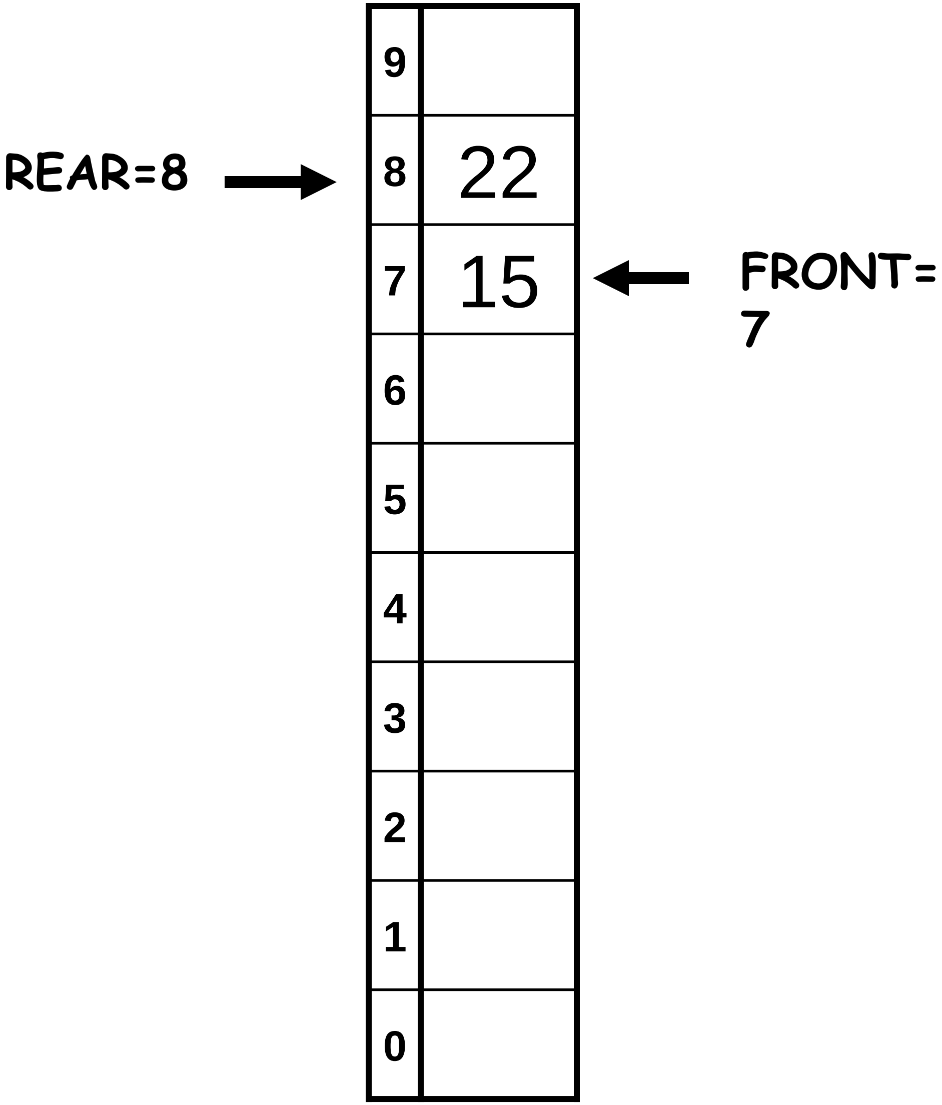
- For a queue represented using an array, when the REAR pointer reaches the end, the insertion will be denied even if room is available at the front

Let us trace the above algorithm with queue LENGTH =10. Suppose the current state of the queue is FRONT=7 and REAR=8. 10 operations are requested as under

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

Let us trace the above algorithm with queue LENGTH =10. Suppose the current state of the queue is FRONT=7 and REAR=8. 10 operations are requested as under

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

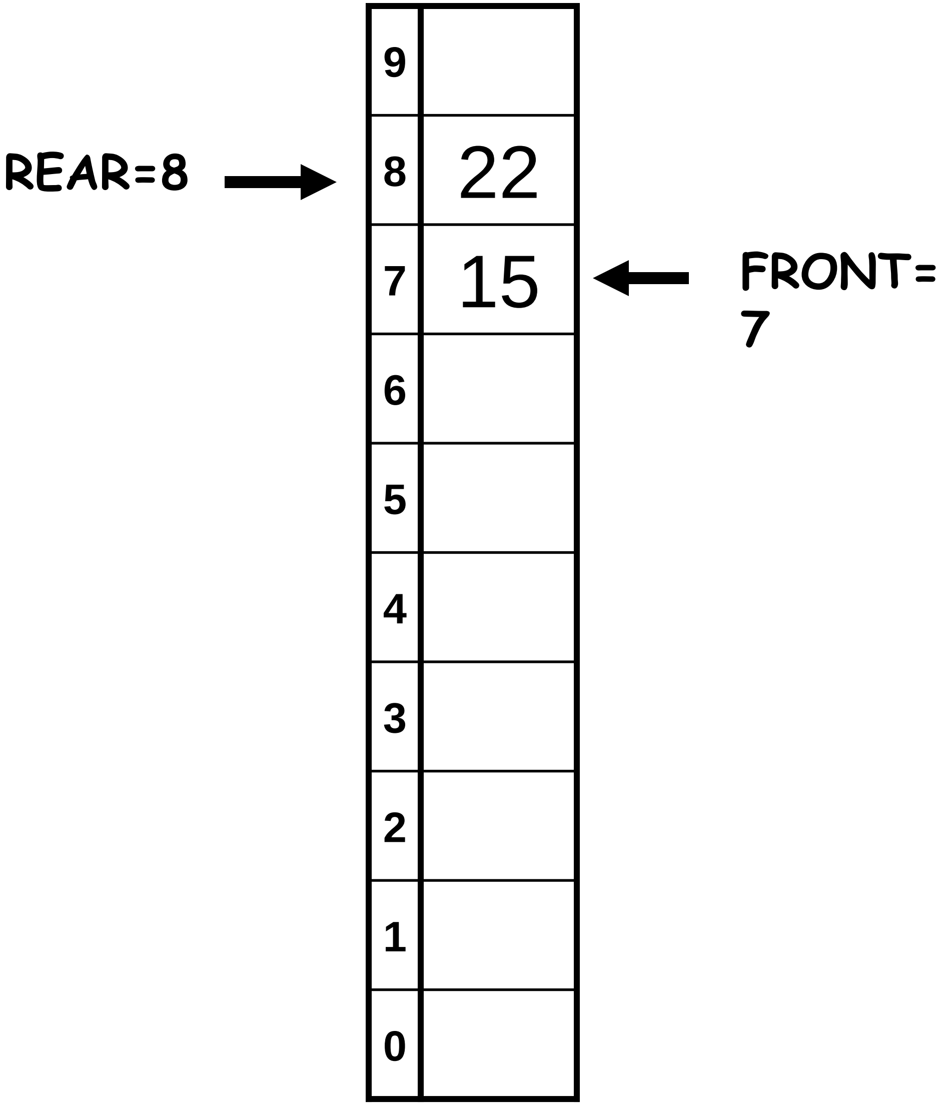


REAR=8

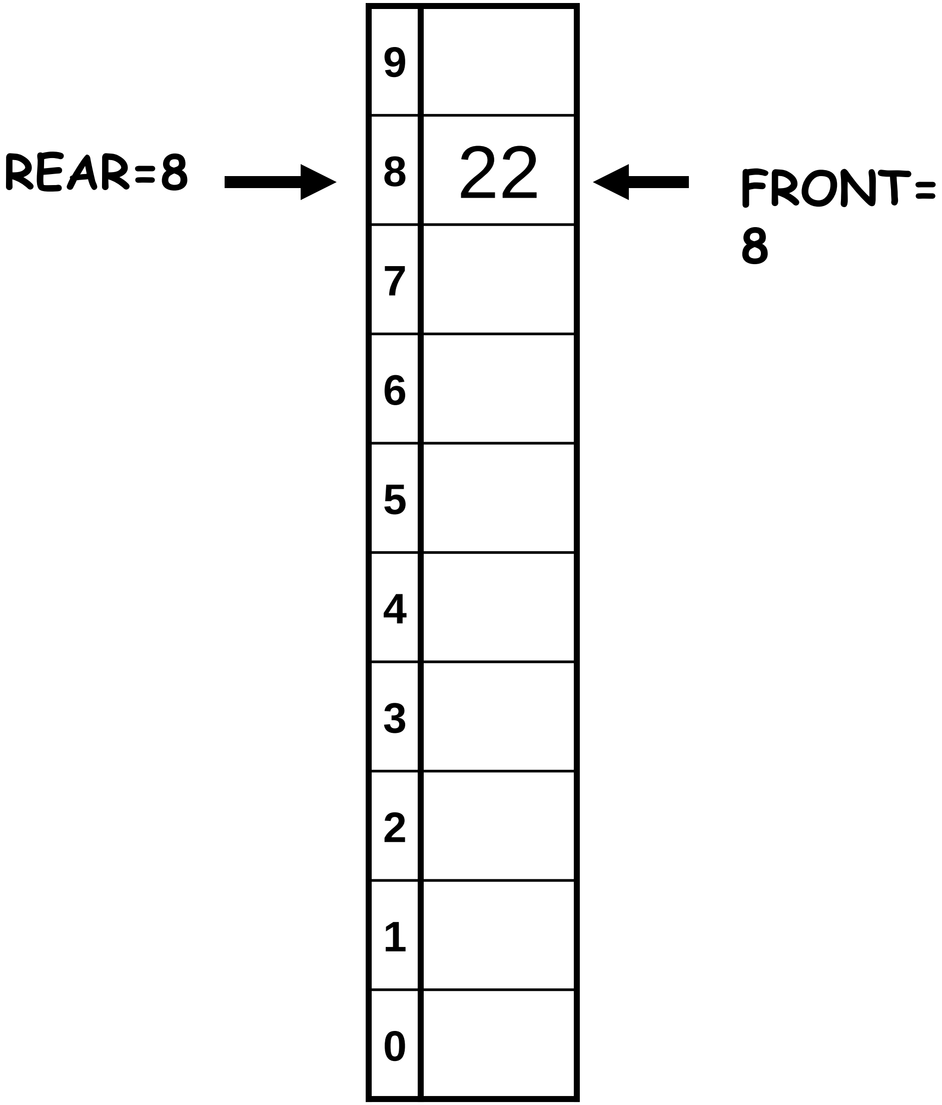
FRONT= 7

9	
8	22
7	15
6	
5	
4	
3	
2	
1	
0	

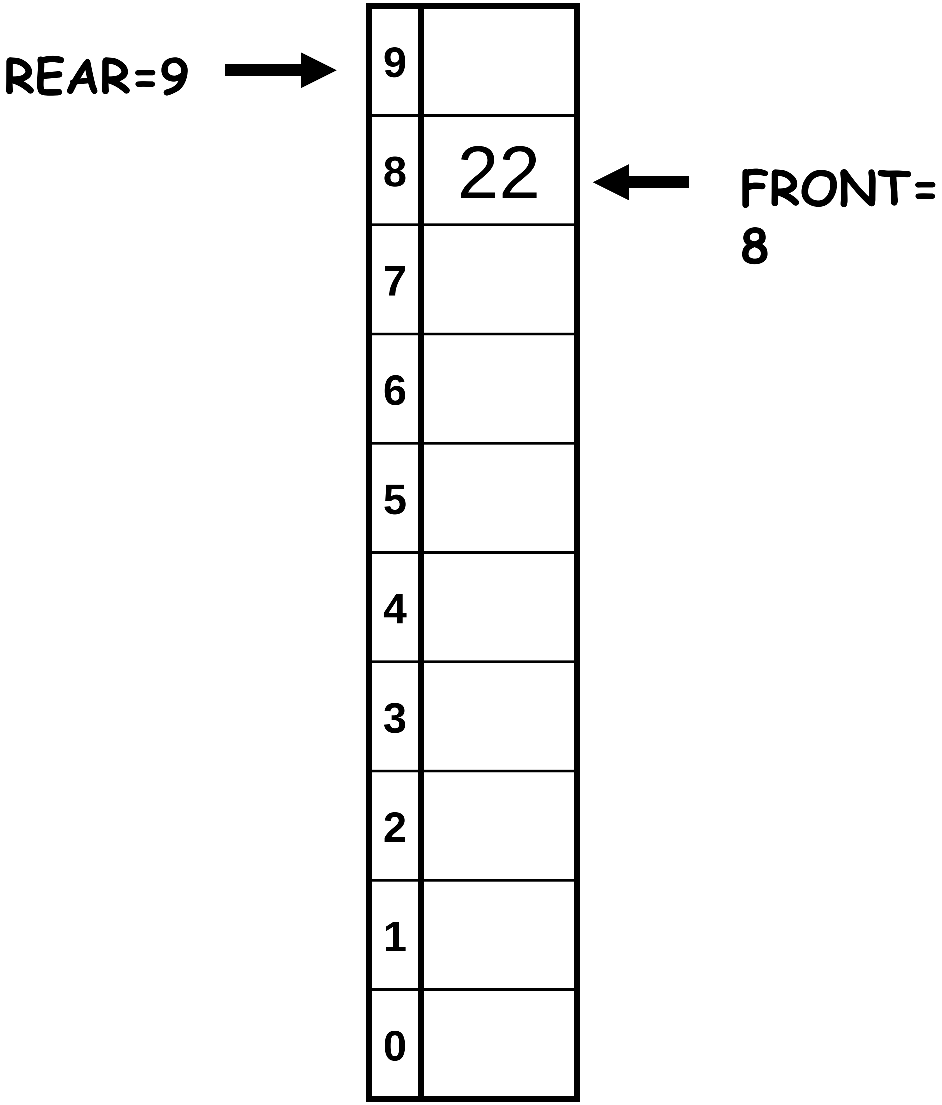
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



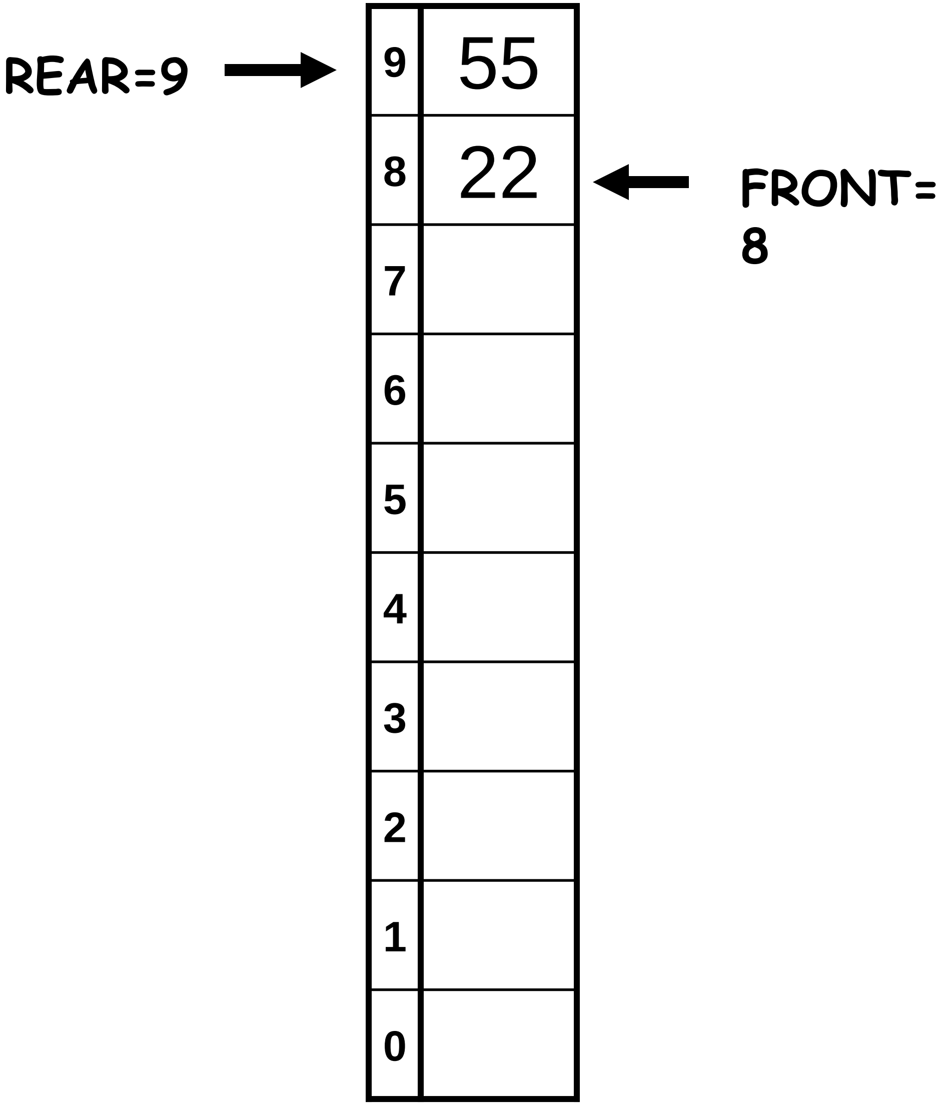
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



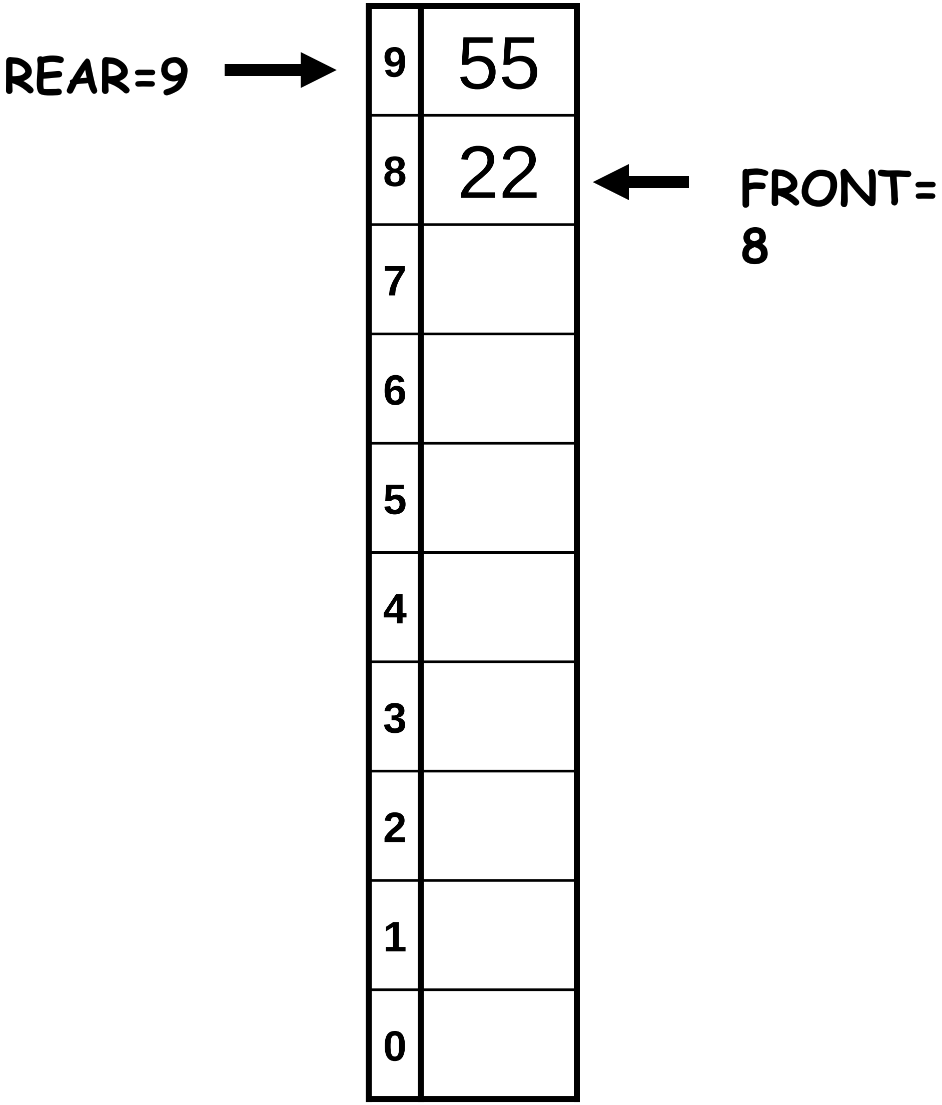
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



REAR=9

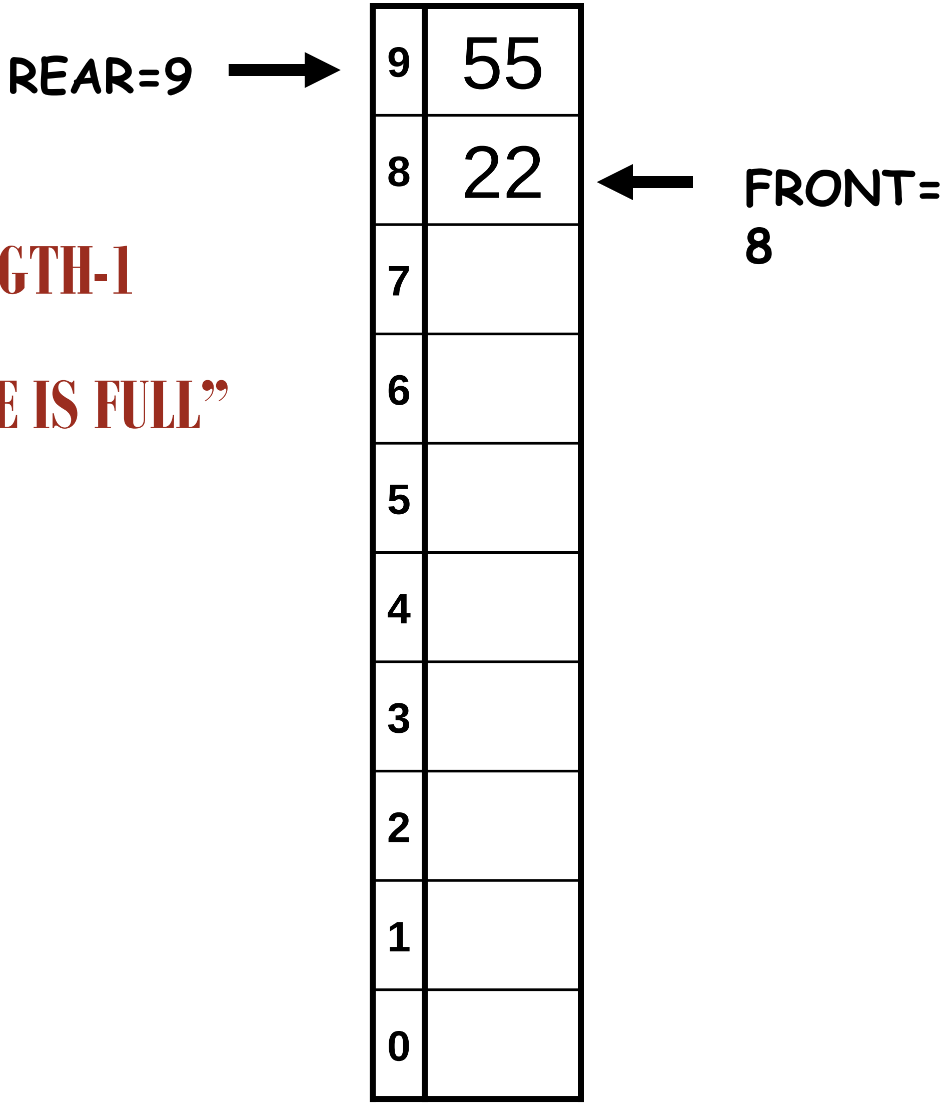
FRONT= 8

9	55
8	22
7	
6	
5	
4	
3	
2	
1	
0	

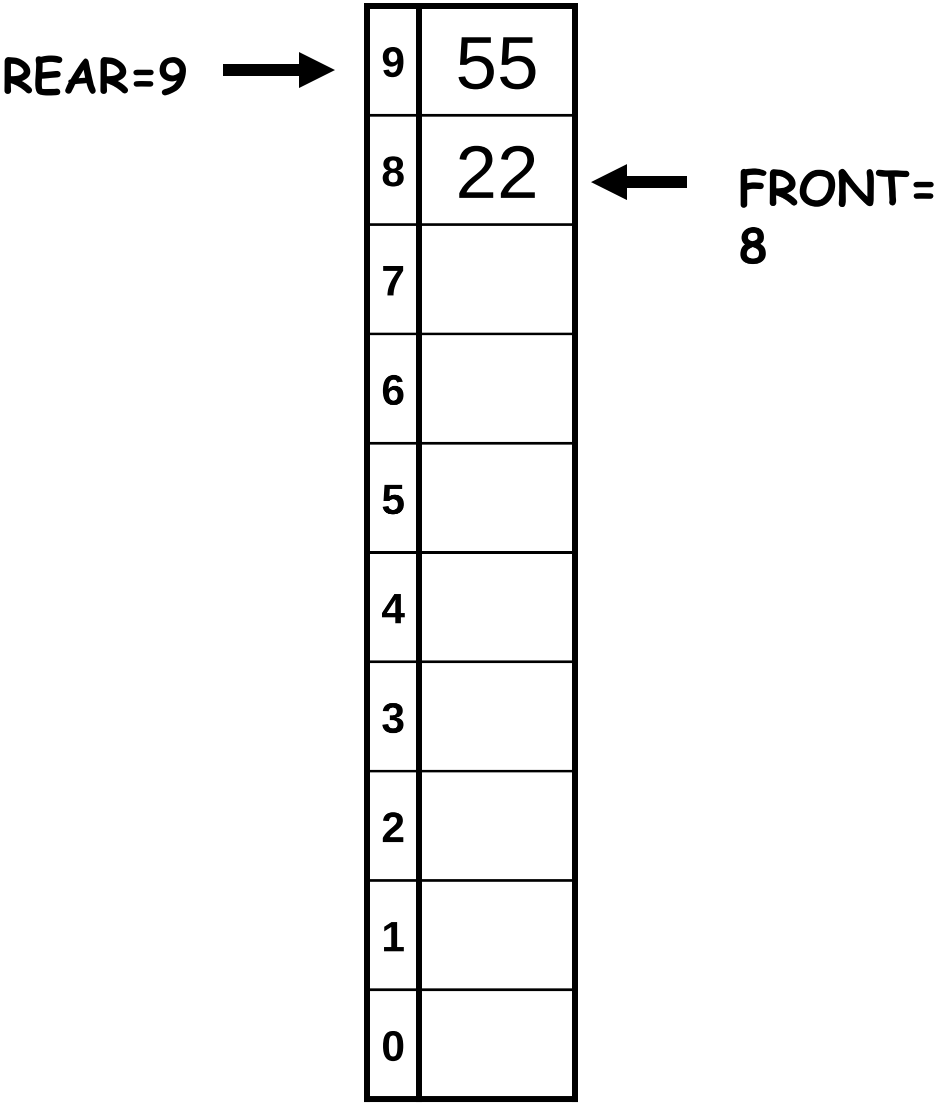
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

REAR = LENGTH-1

Print “QUEUE IS FULL”



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

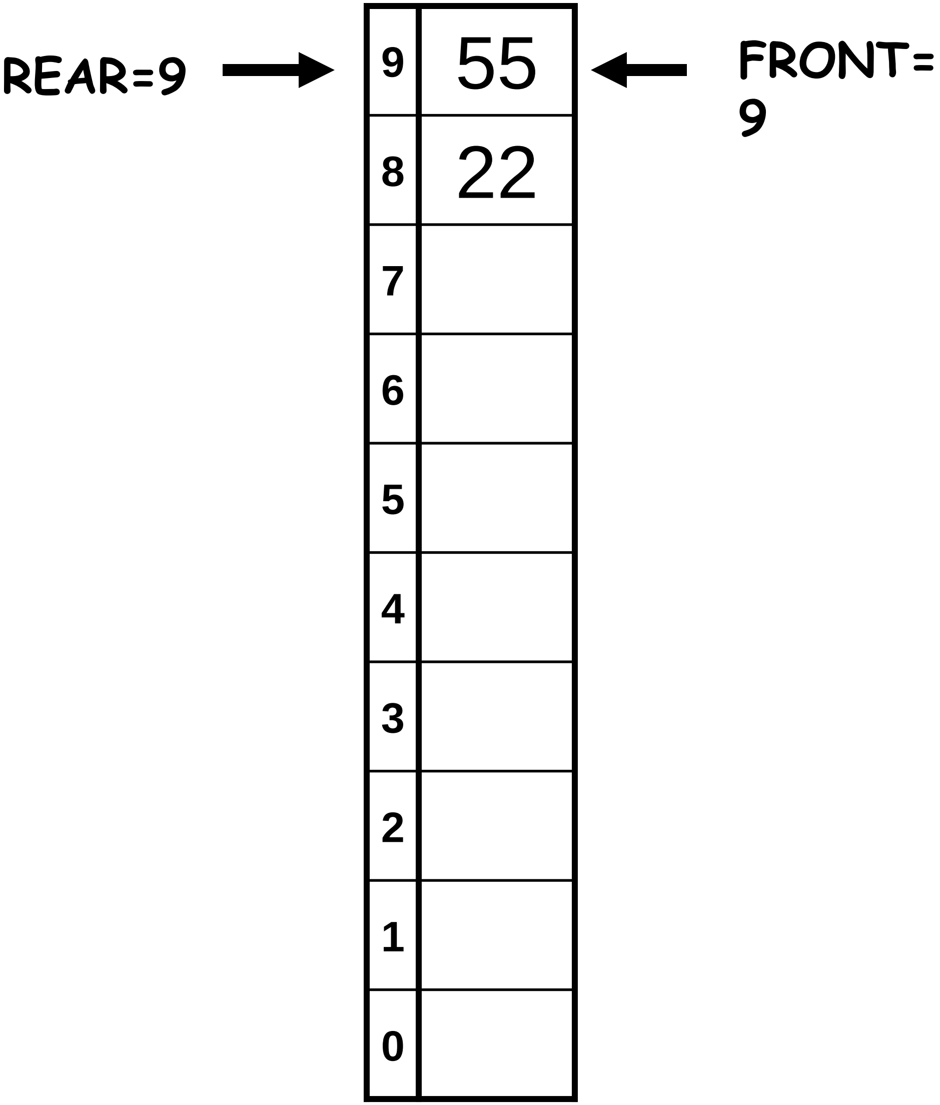


REAR=9

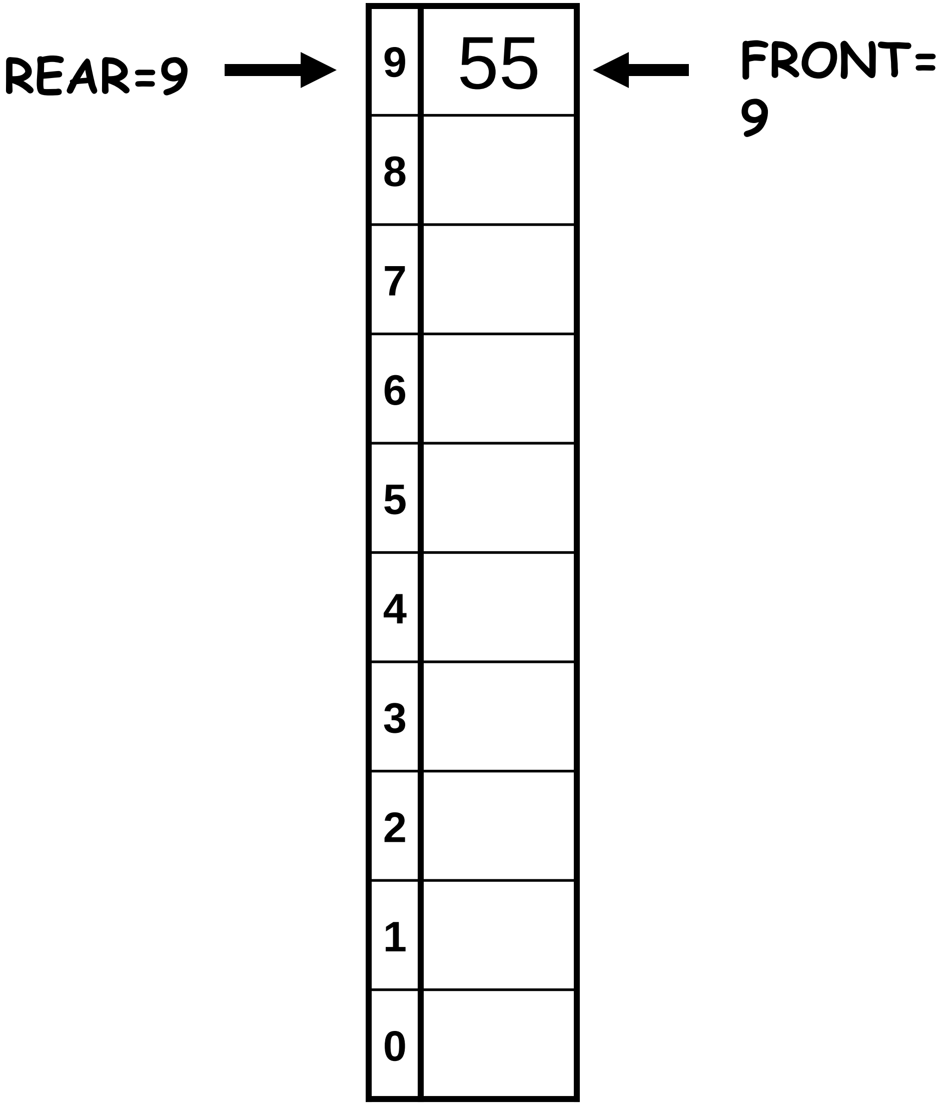
FRONT= 8

9	55
8	22
7	
6	
5	
4	
3	
2	
1	
0	

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

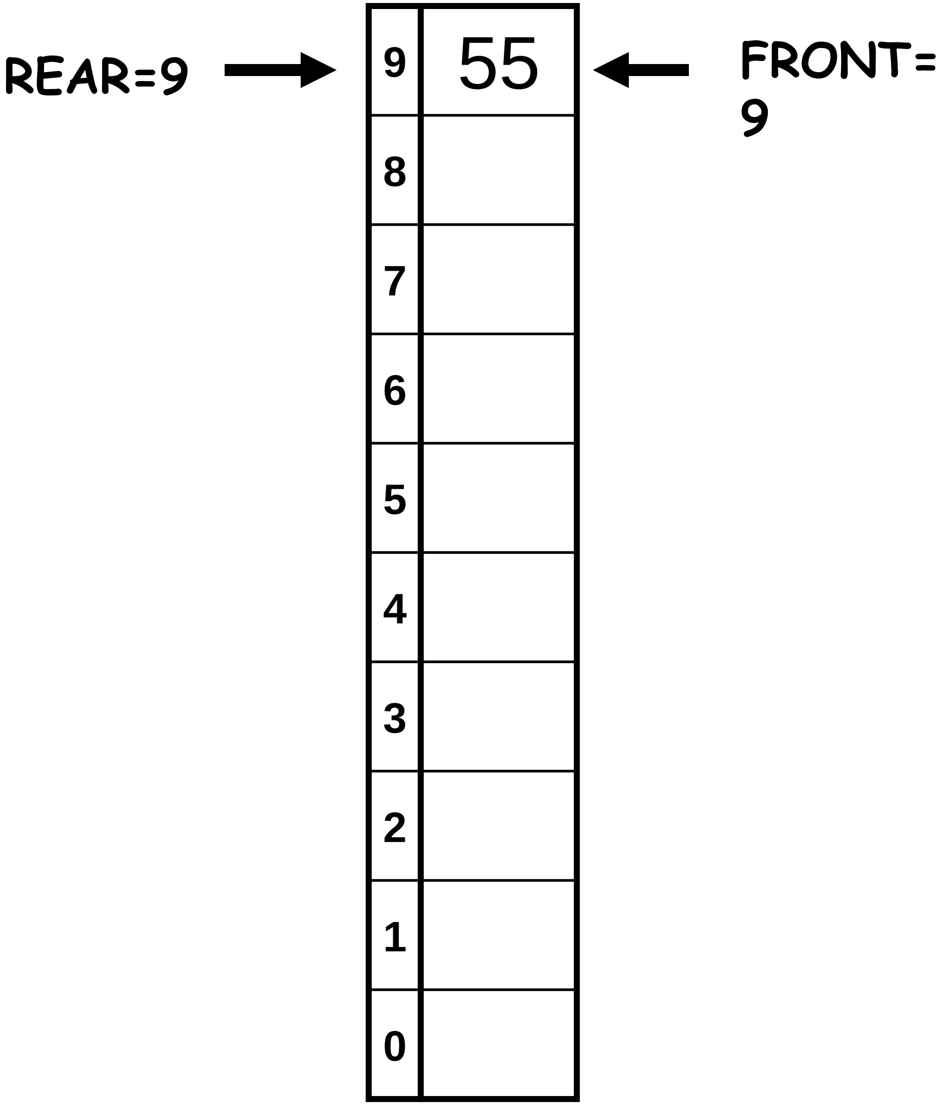


REAR=9

FRONT=
9

9	55
8	
7	
6	
5	
4	
3	
2	
1	
0	

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



REAR=9

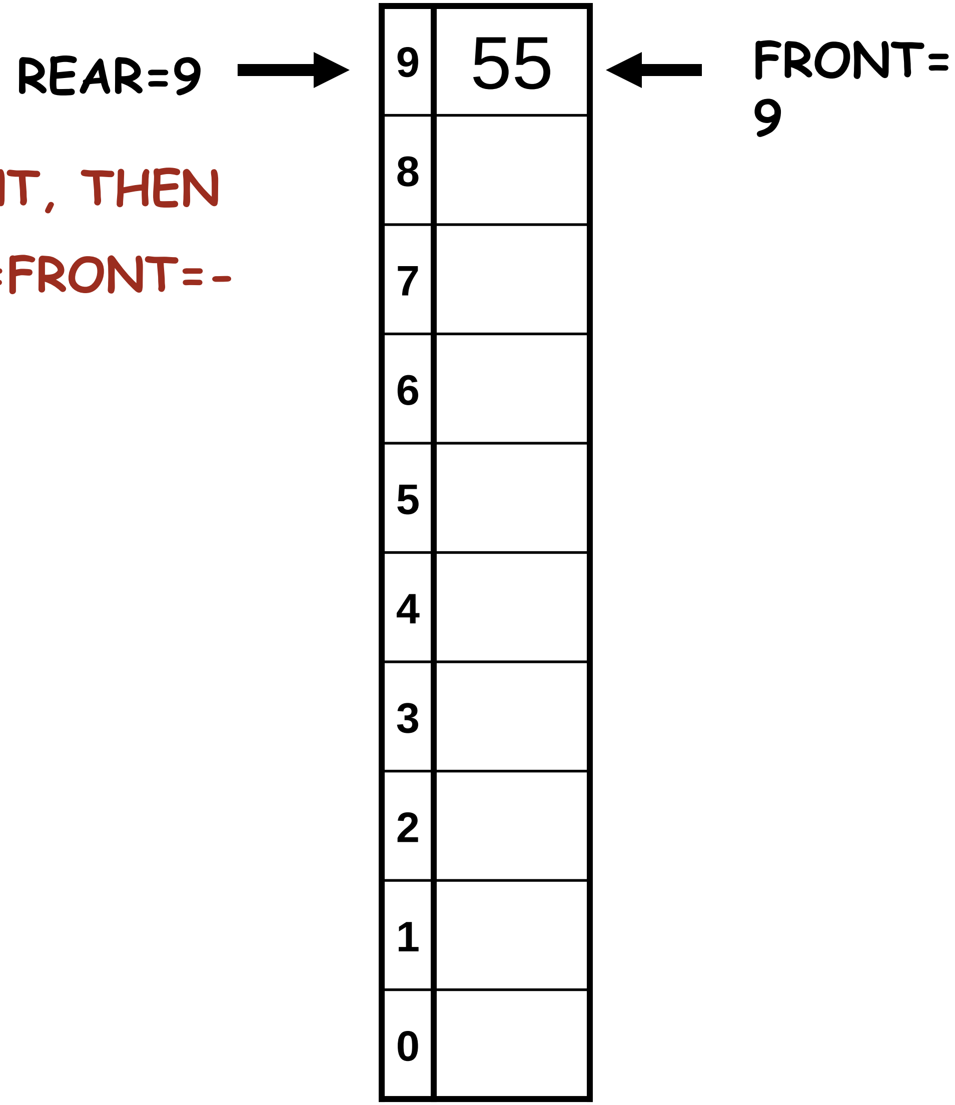
9	55
8	
7	
6	
5	
4	
3	
2	
1	
0	

FRONT= 9

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

REAR=FRONT, THEN

SET REAR=FRONT=-1



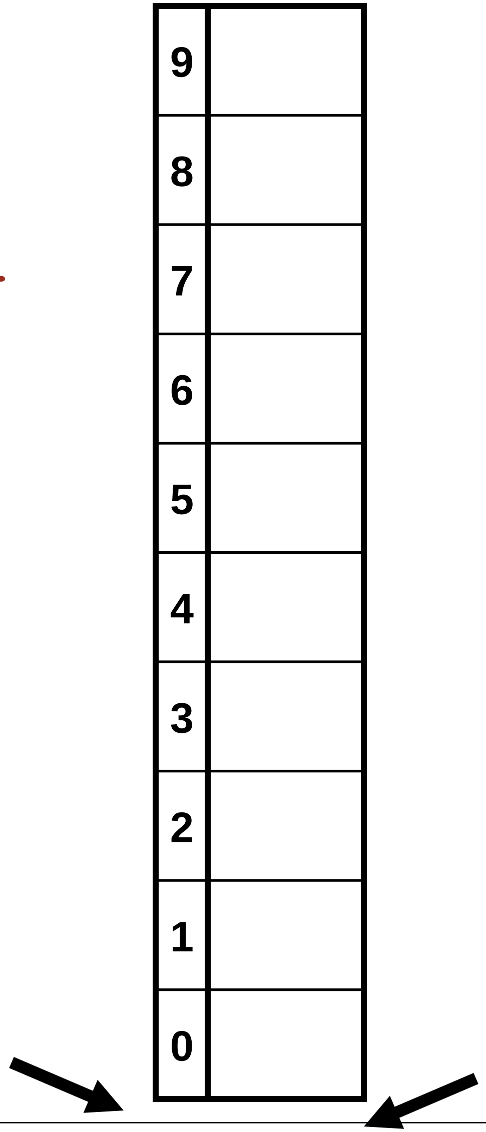
REAR=9

FRONT=9

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

REAR=FRONT, THEN

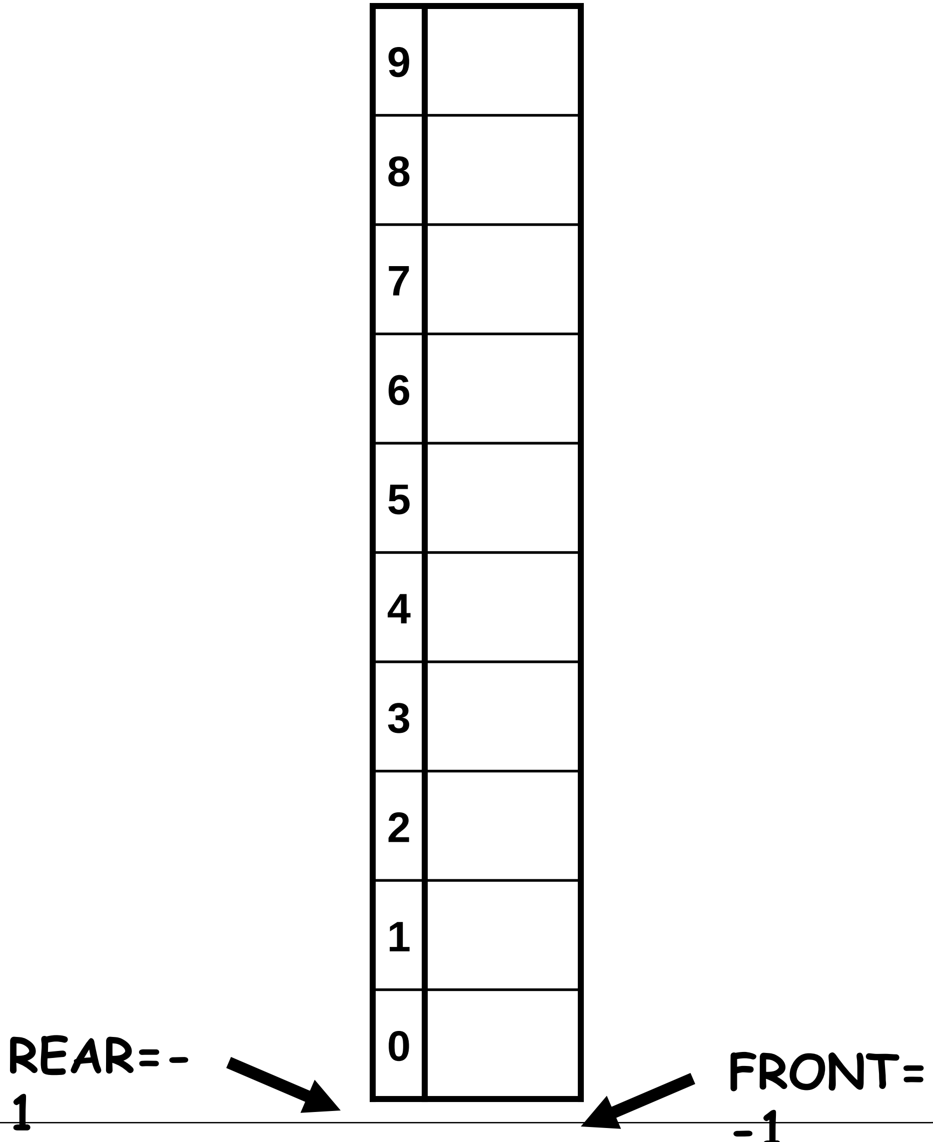
SET REAR=FRONT=-1



REAR=-1

FRONT=-1

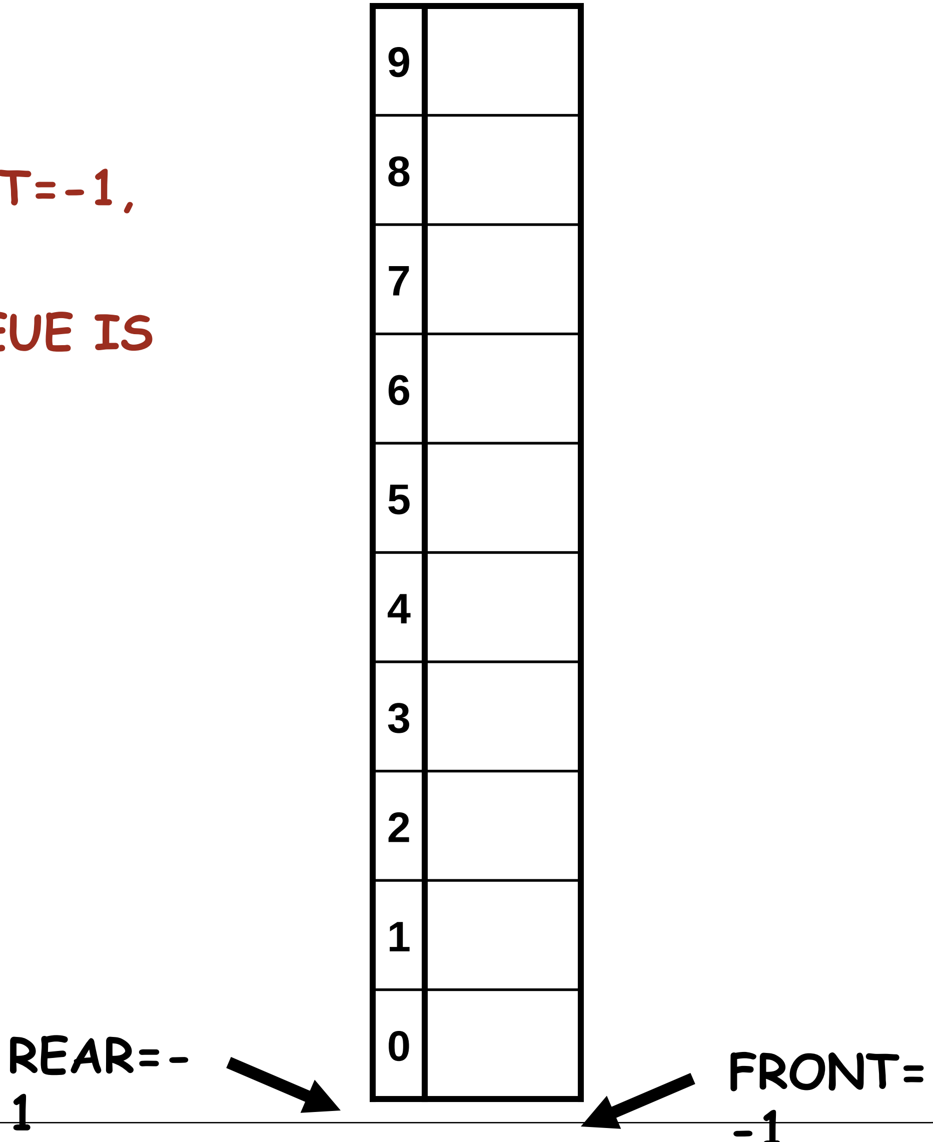
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



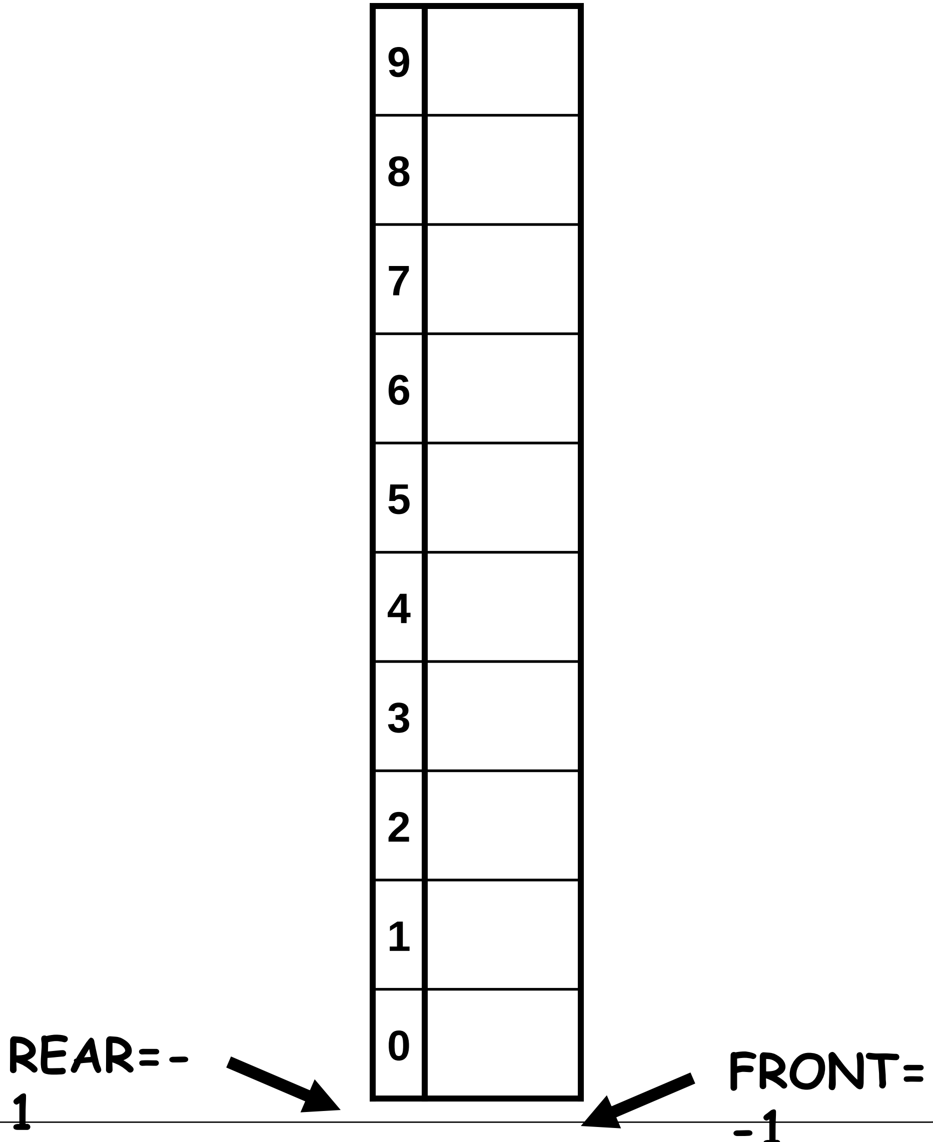
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

REAR=FRONT=-1, THEN

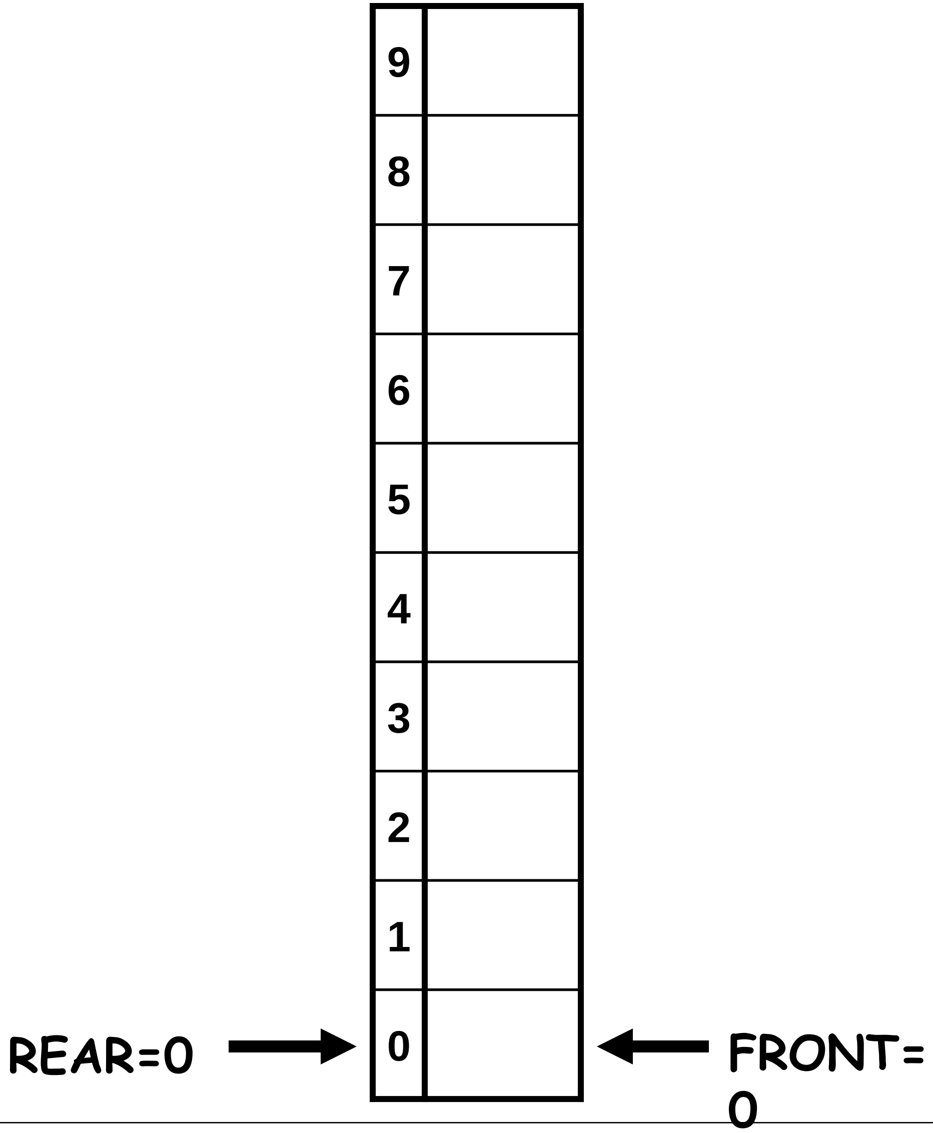
PRINT "QUEUE IS EMPTY"



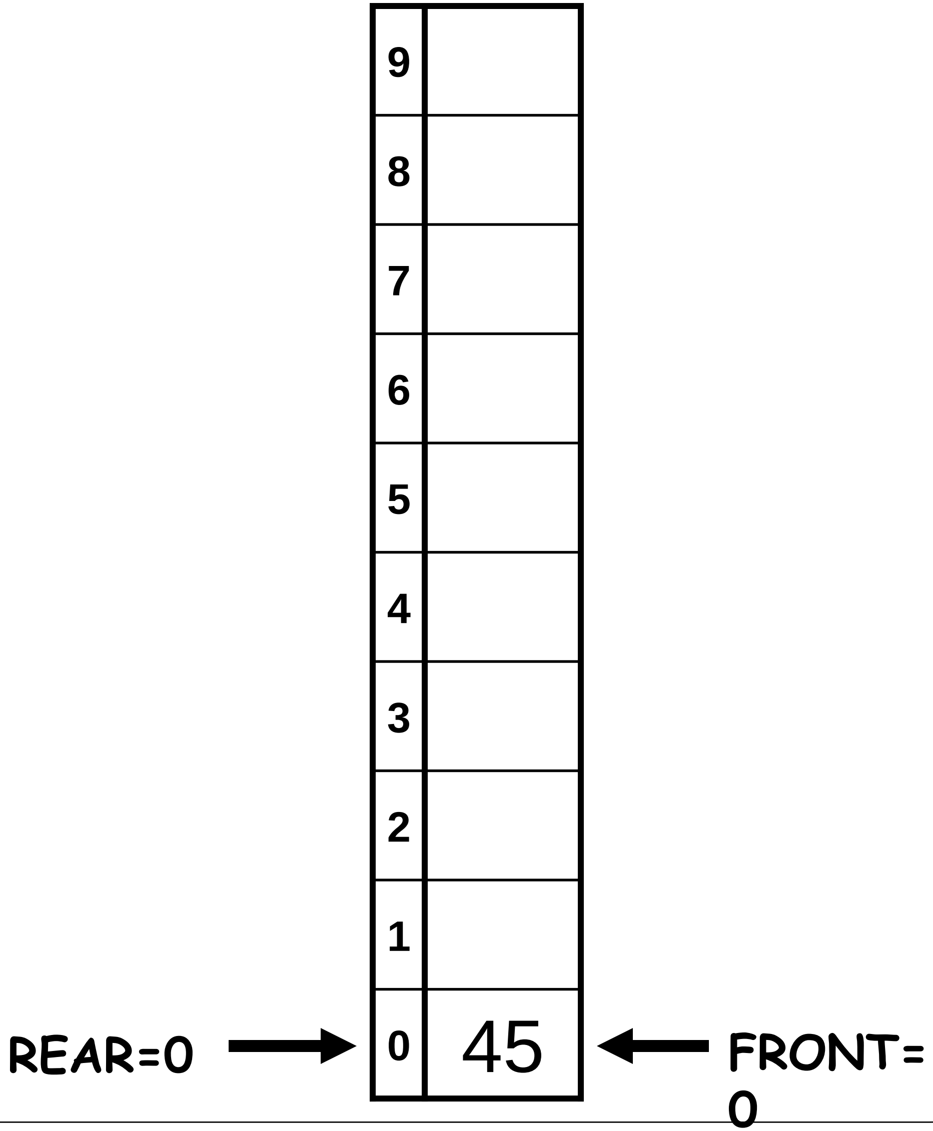
1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

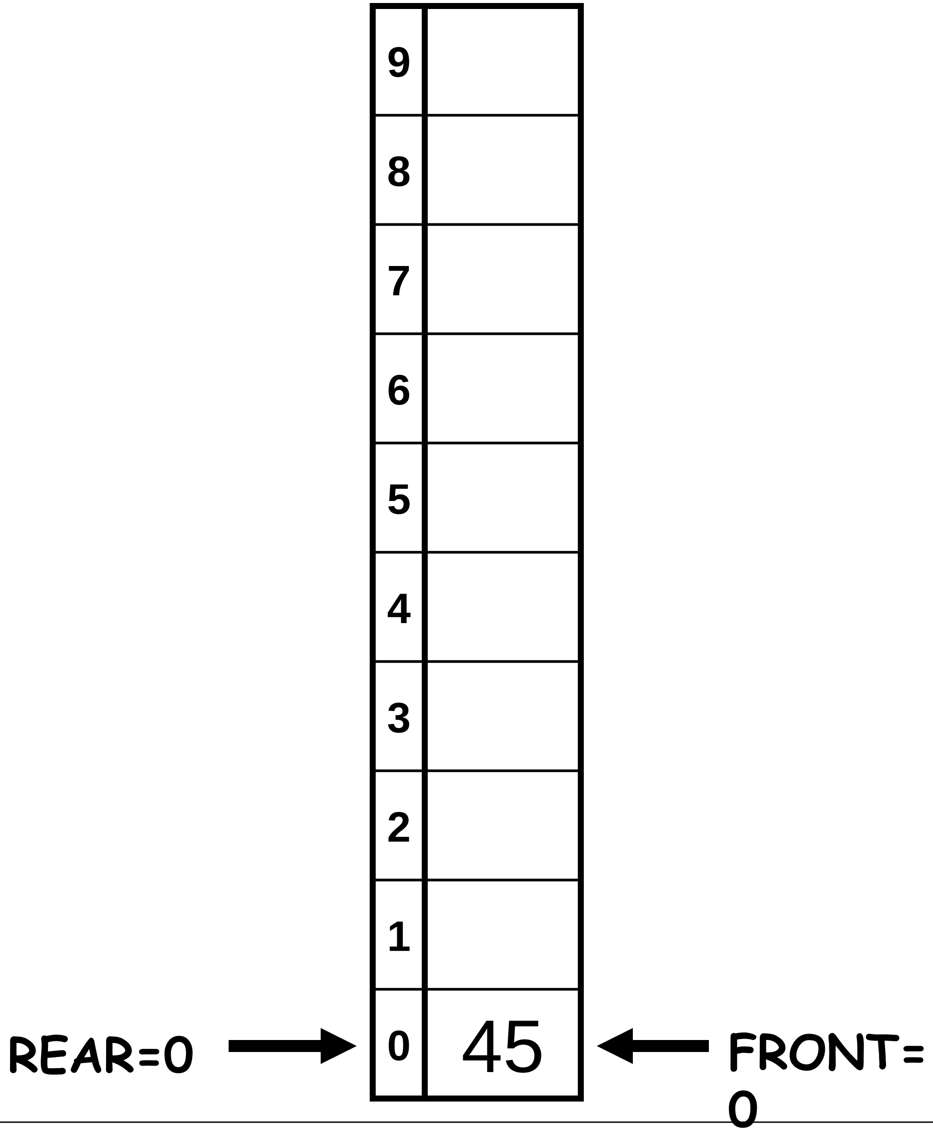


9	
8	
7	
6	
5	
4	
3	
2	
1	
0	45

REAR=0

FRONT=
0

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

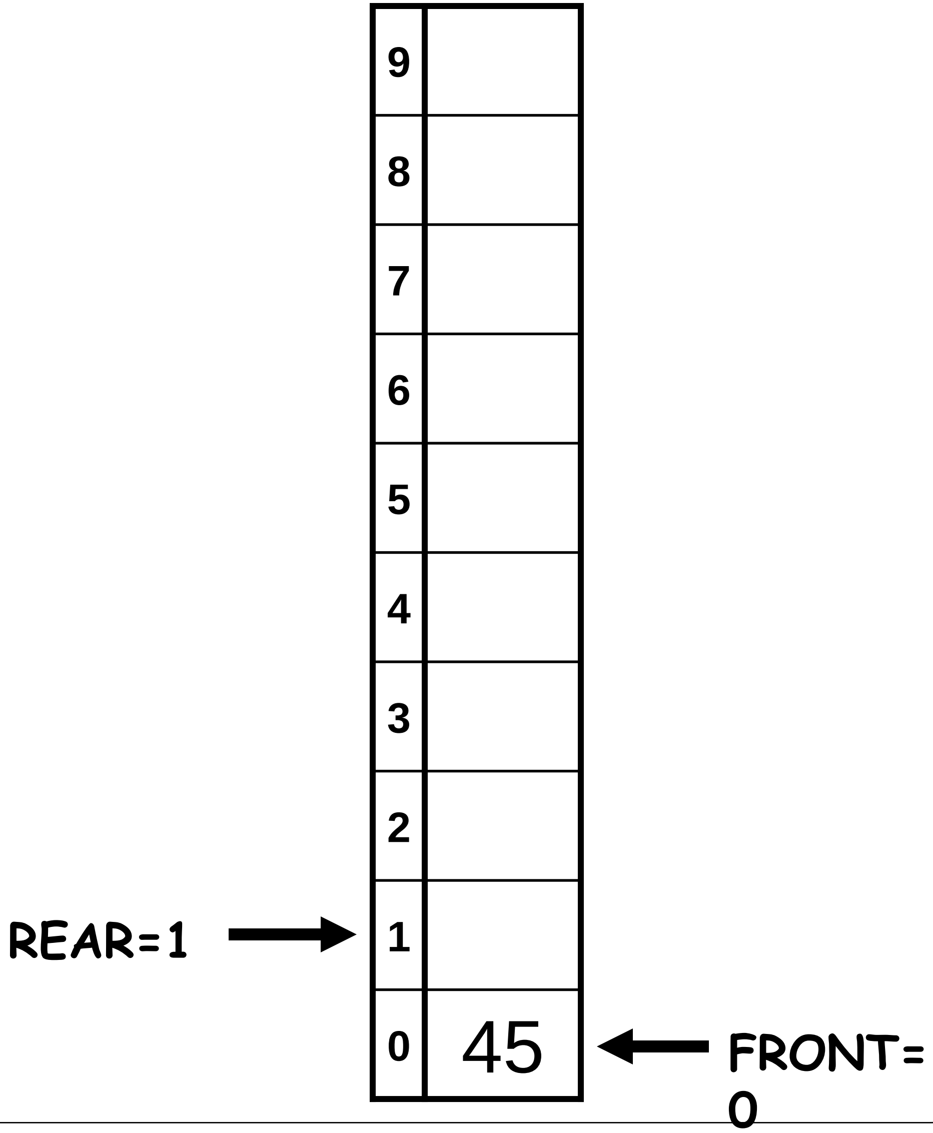


9
8
7
6
5
4
3
2
1
0 45

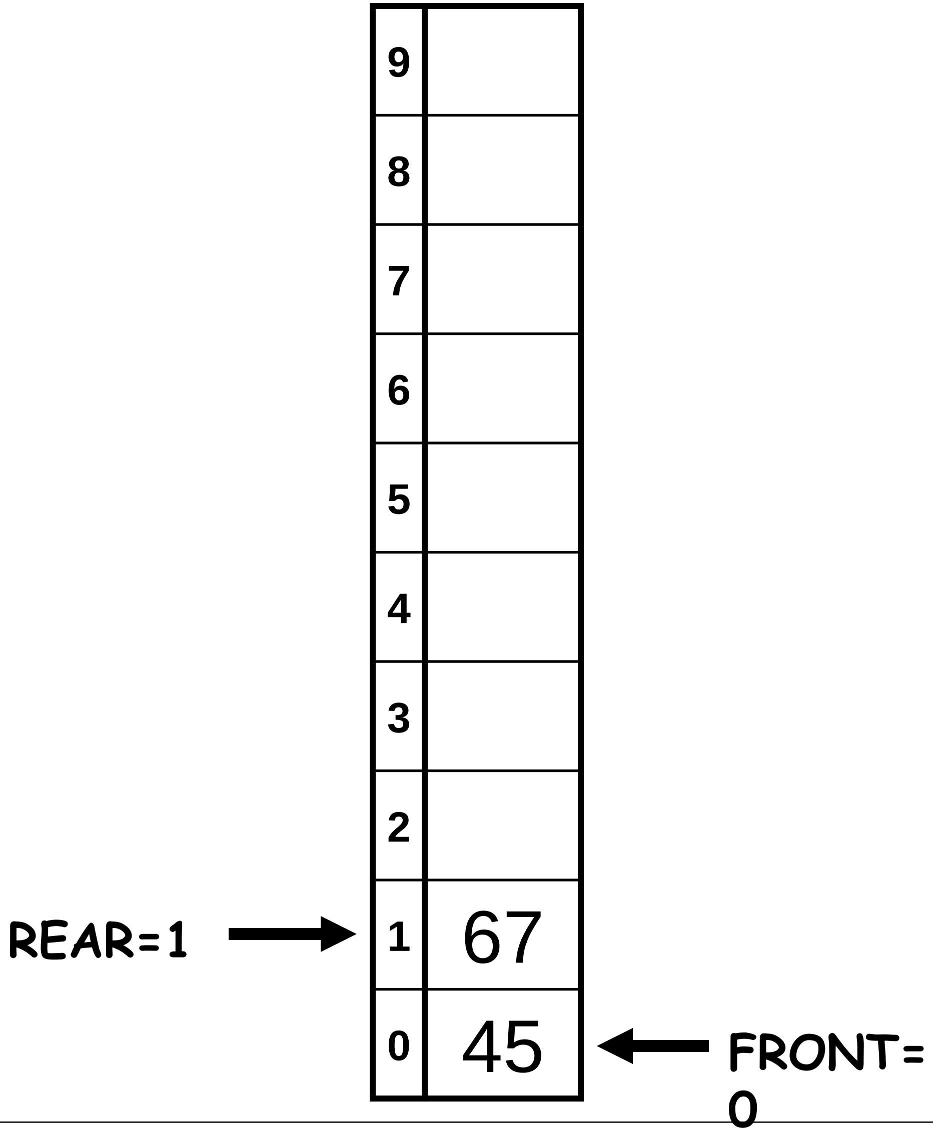
REAR=0

FRONT= 0

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

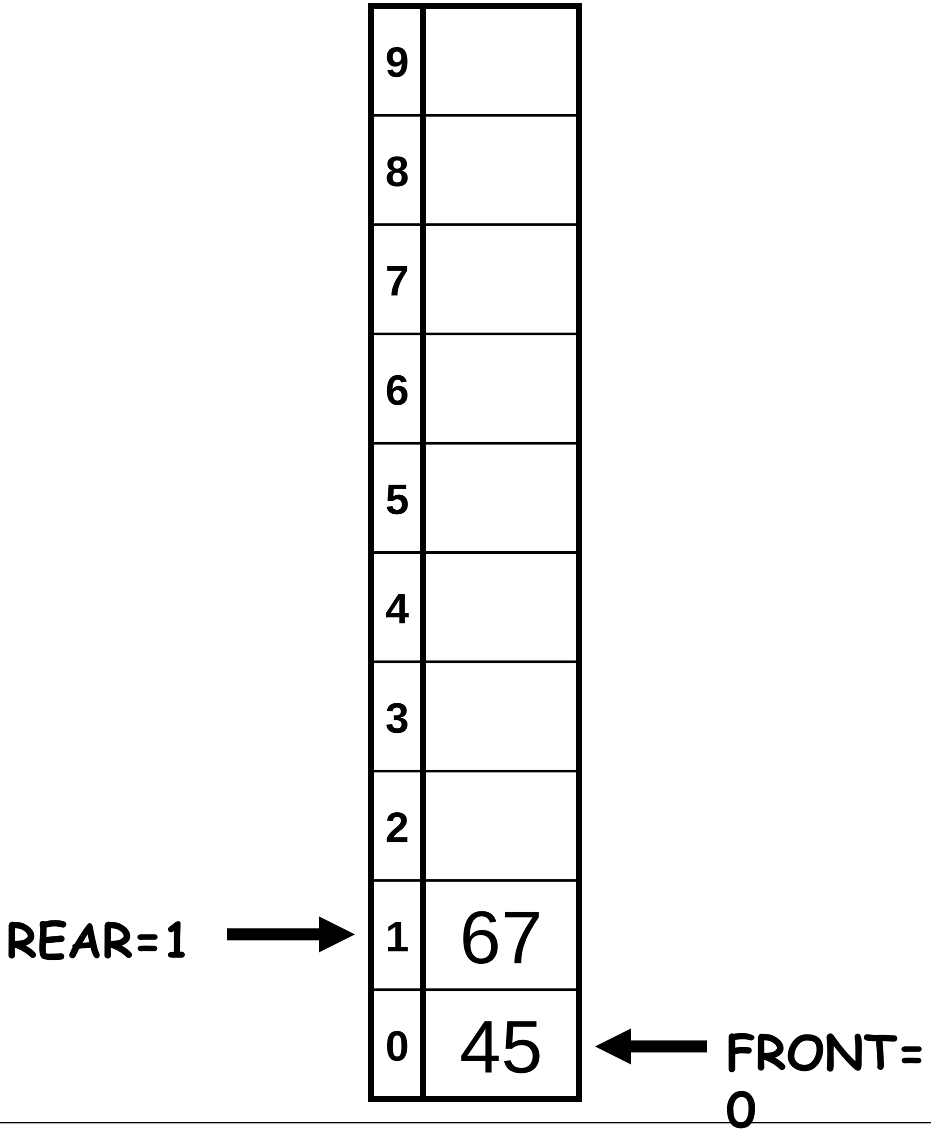


9
8
7
6
5
4
3
2
1 67
0 45

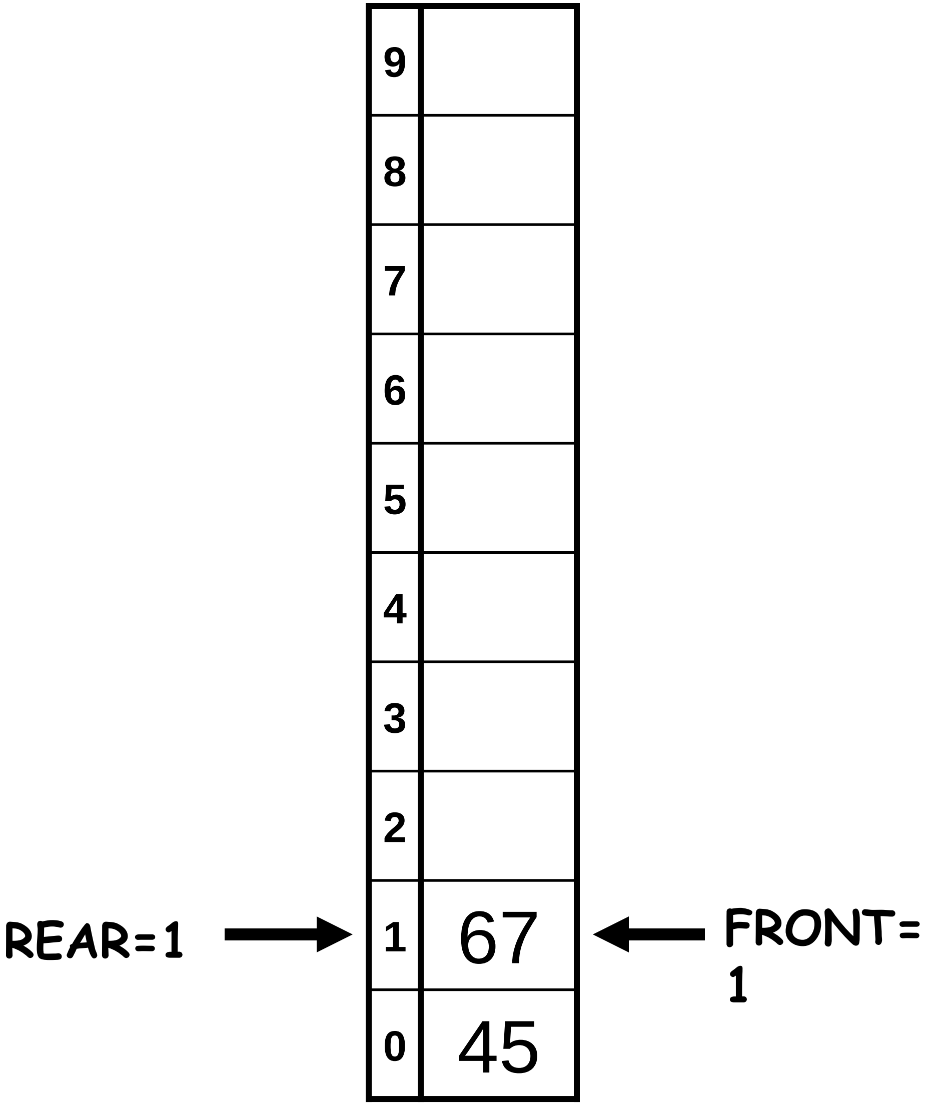
REAR=1

FRONT=
0

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

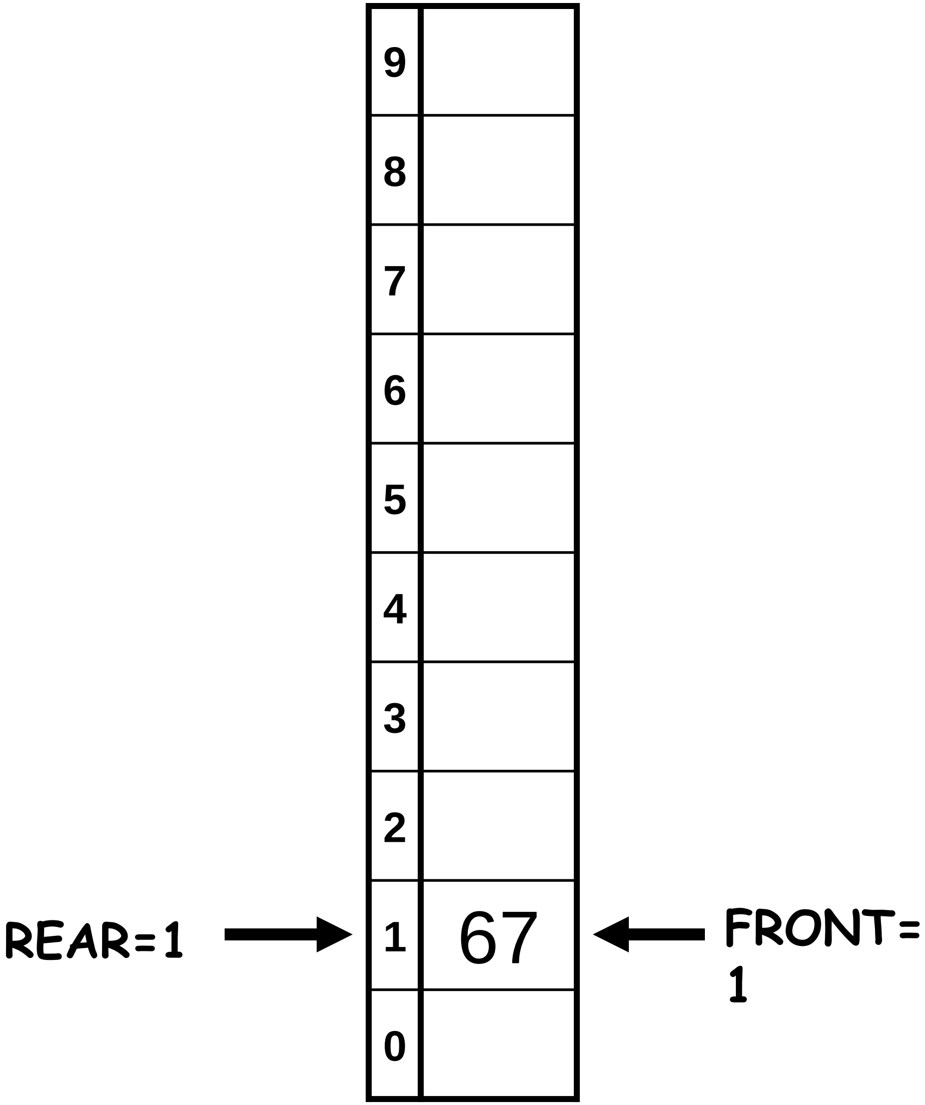


9	
8	
7	
6	
5	
4	
3	
2	
1	67
0	45

REAR=1

FRONT= 1

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

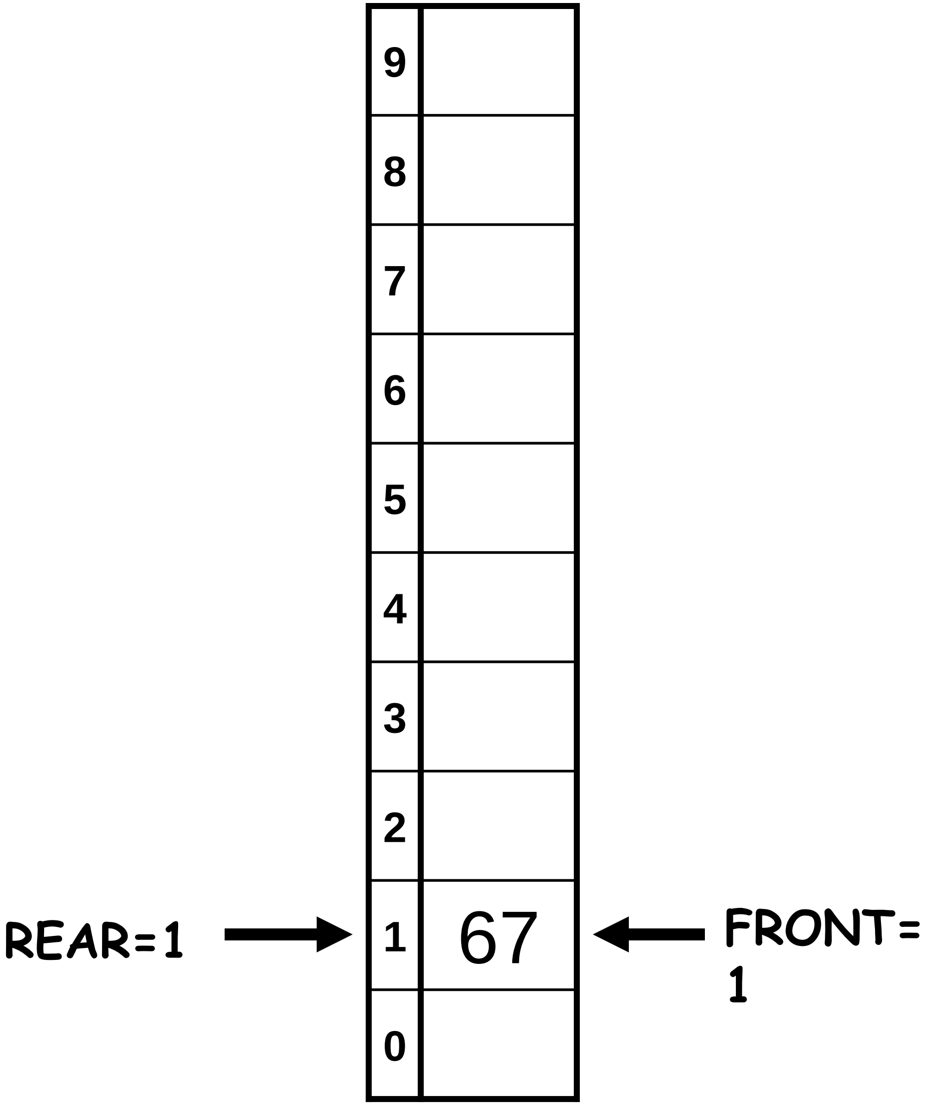


9	
8	
7	
6	
5	
4	
3	
2	
1	67
0	

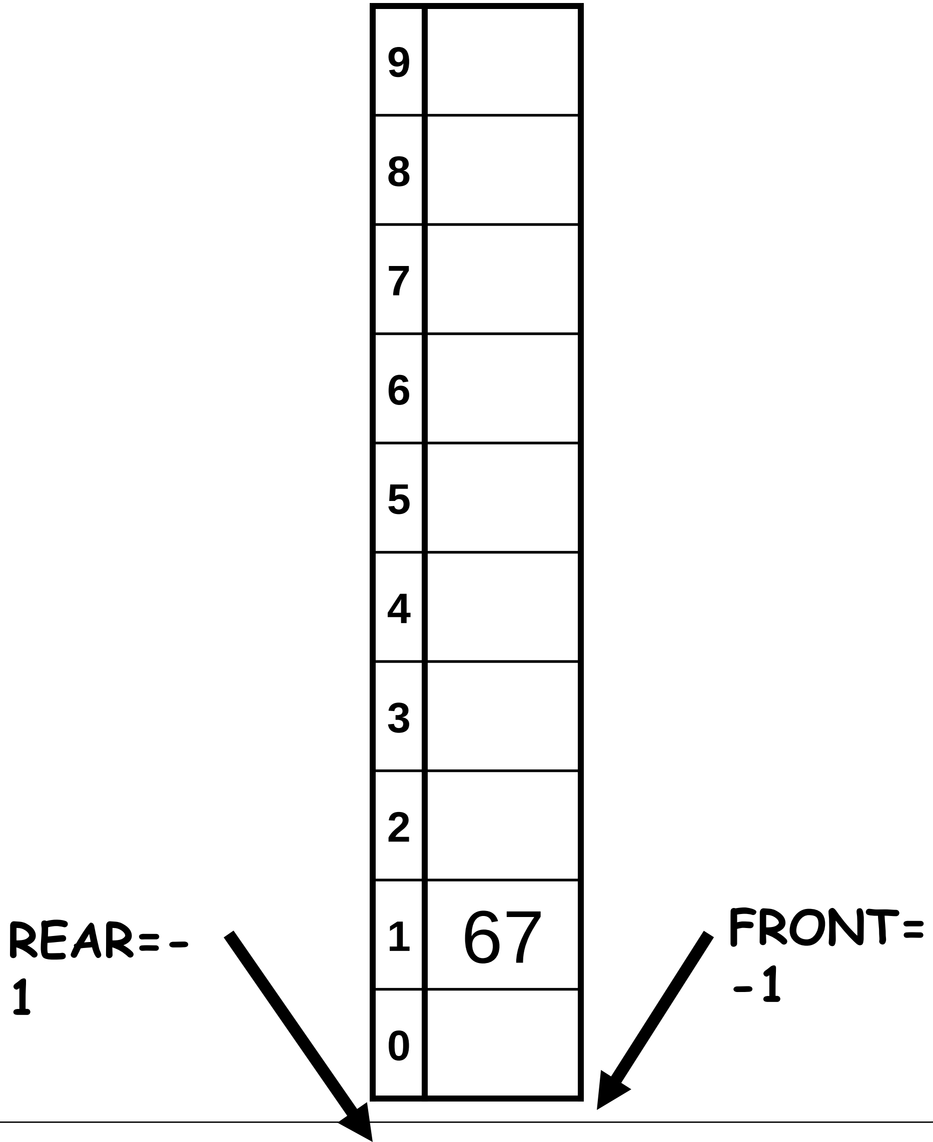
REAR=1

FRONT=
1

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE

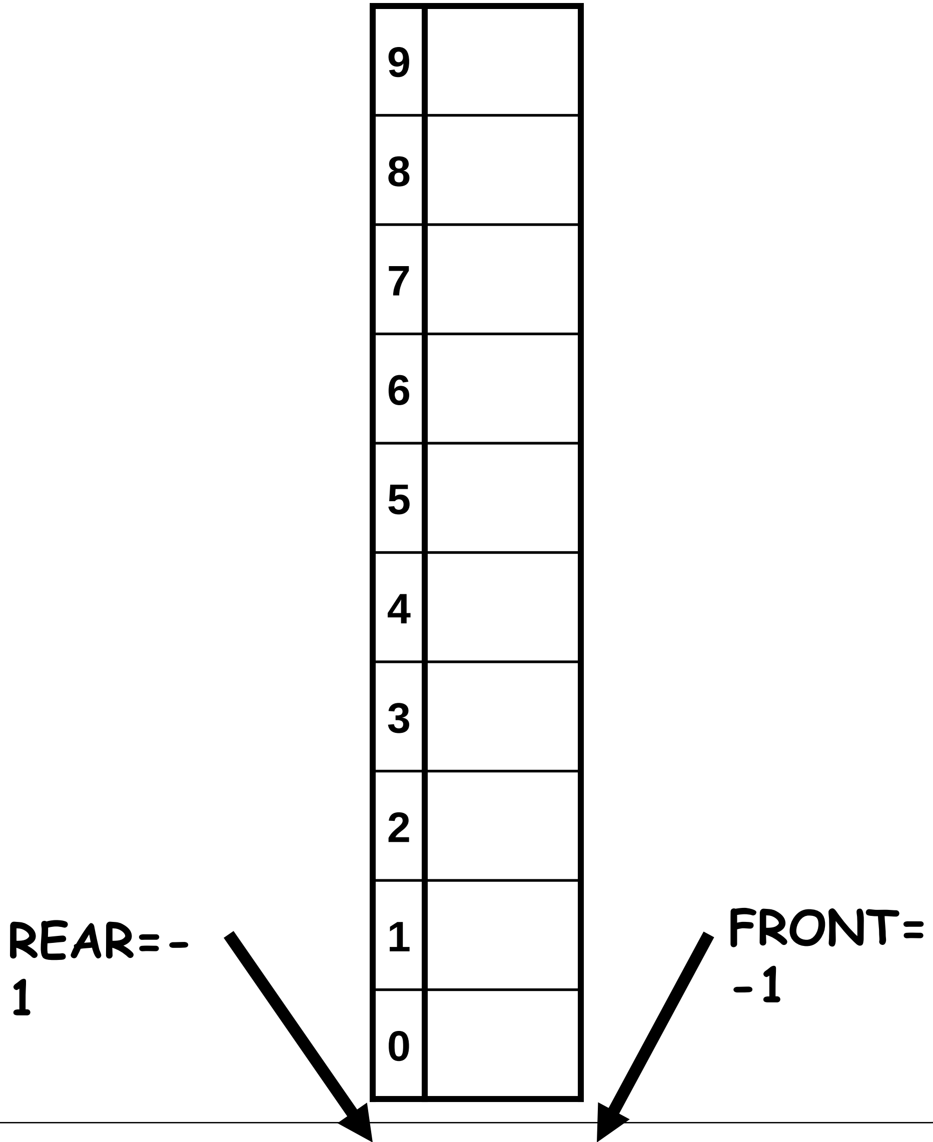


9	
8	
7	
6	
5	
4	
3	
2	
1	67
0	

REAR=-1

FRONT=-1

1. DEQUEUE
2. ENQUEUE
3. ENQUEUE
4. DEQUEUE
5. DEQUEUE
6. DEQUEUE
7. ENQUEUE
8. ENQUEUE
9. DEQUEUE
10. DEQUEUE



REAR=-
1

FRONT=
-1

- We can see that with this representation queue may not be full, still a request for insertion operation is denied
- This is simply a wastage of storage
- This type of representation can be recommended for an application where the queue is emptied at certain intervals