Data Structure
BCS - III

By
Islam Zada
Lecturer in Department of Software Engineering

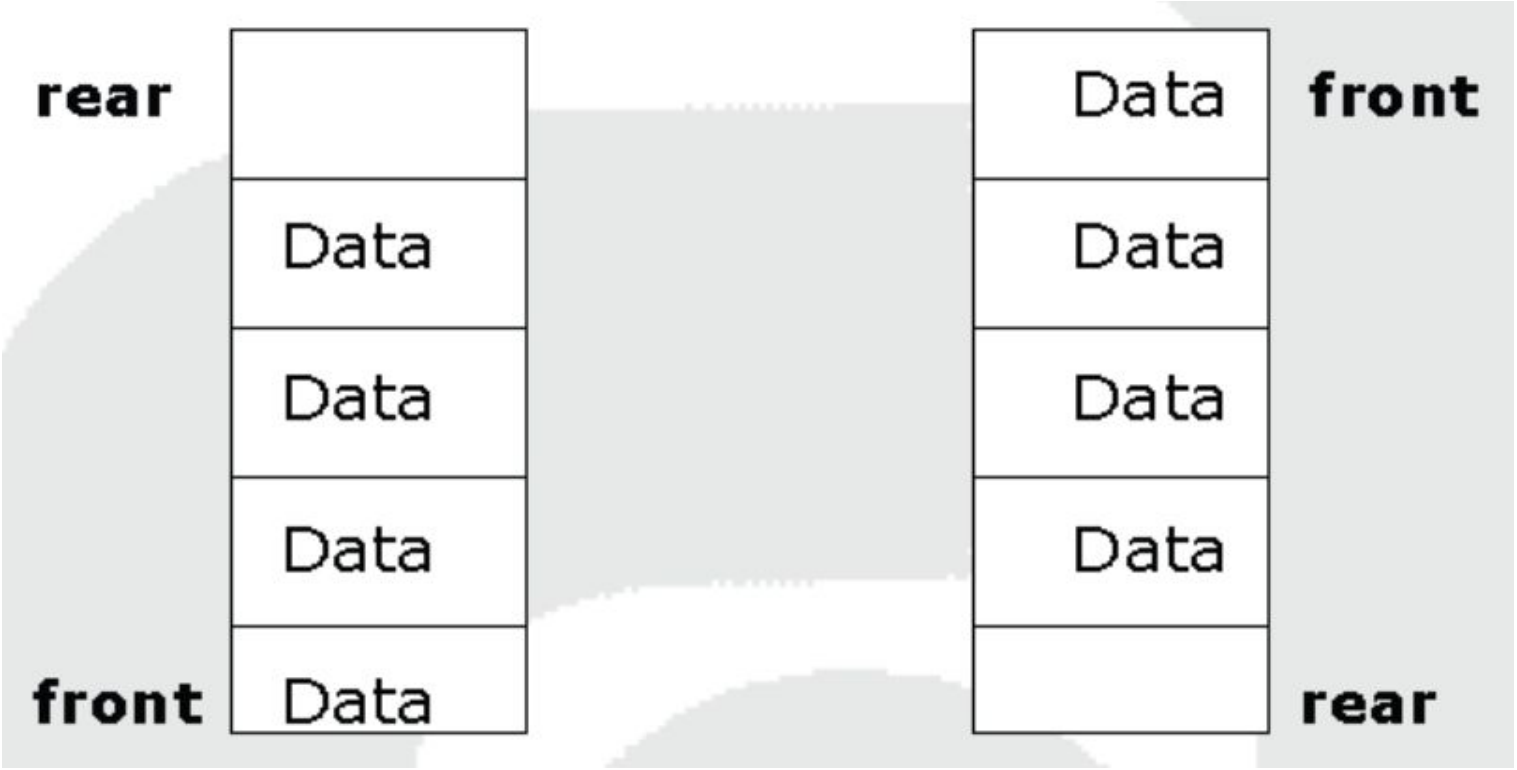
Queues

QUEUE

- A linear data structure into which items can only be inserted at one end called rear and removed from the other called front.
- Queue is very useful in computer science.
- We define a queue to be a list in which all additions to the list is made at the one end & all deletions from the list is made at other end.
- Queue are also called First In First Out list of FIFO for sort.

QUEUE

- We may draw queue in any one of the forms as given below



Queue Operations

Enqueue(X) – insert X at the *rear* of the queue.

Dequeue() --remove the *front* element from queue.

Front() -- return front element.

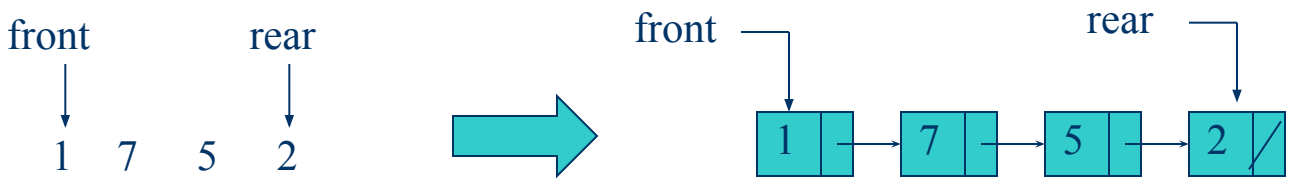
IsEmpty() -- return TRUE if queue is empty, FALSE otherwise

IMPLEMENTING QUEUE

- There are mainly two types of queue,
  - Priority queue.
  - Circular queue.
- Queue can be implemented using either
  - Linked list or
  - Array

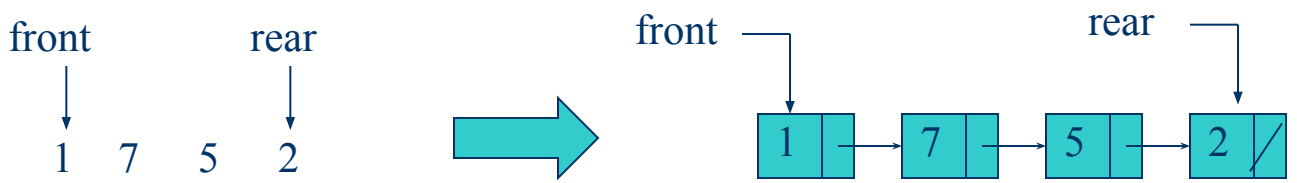
Implementing Queue

- Using linked List:

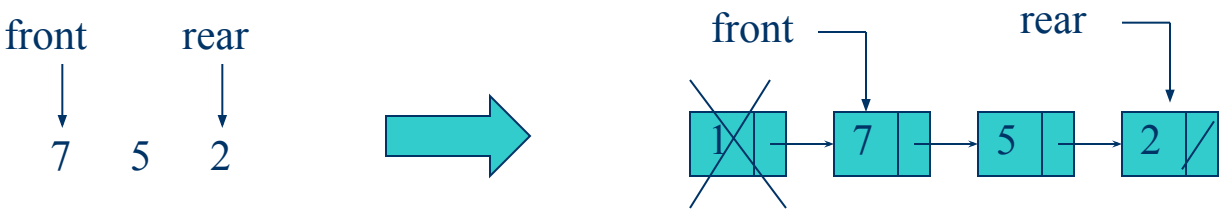


Implementing Queue

- Using linked List:

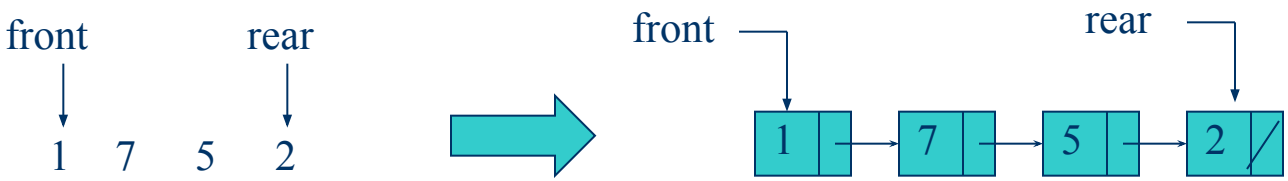


dequeue()



Implementing Queue

- Using linked List:

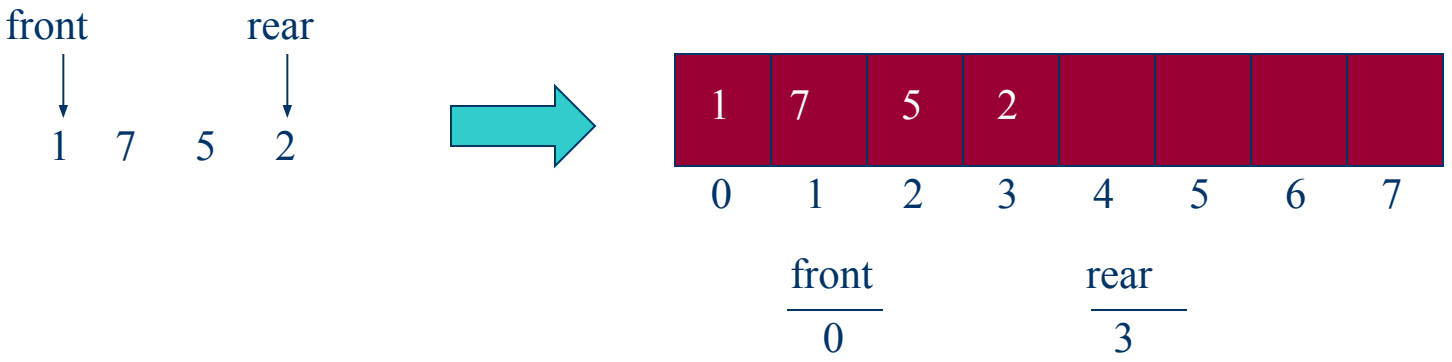


enqueue(9)

Queue using Array

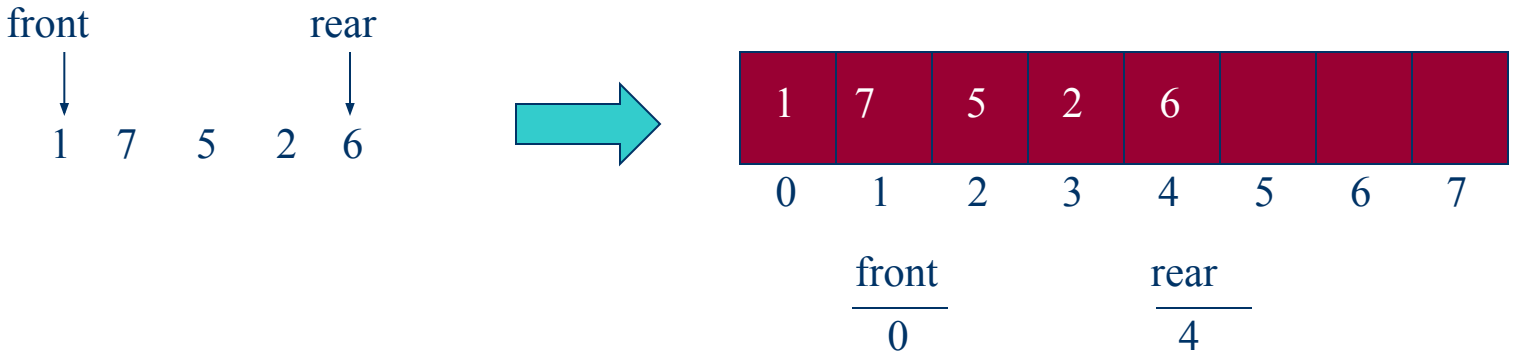
- If we use an array to hold queue elements, both insertions and removal at the front (start) of the array are expensive.
- This is because we may have to shift up to “n” elements.
- For the stack, we needed only one end; for queue we need both.
- To get around this, we will not shift upon removal of an element.

Queue using Array



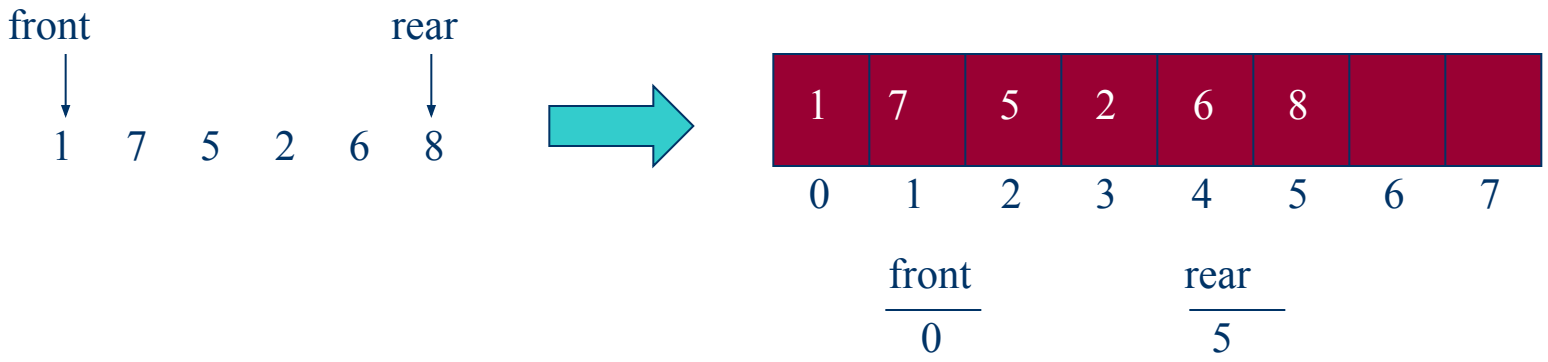
Queue using Array

enqueue(6)



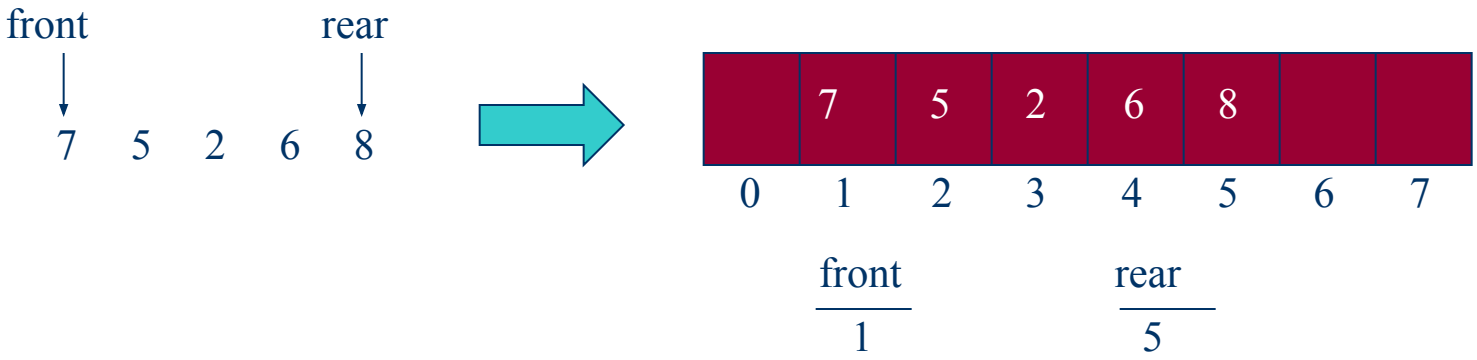
Queue using Array

enqueue(8)



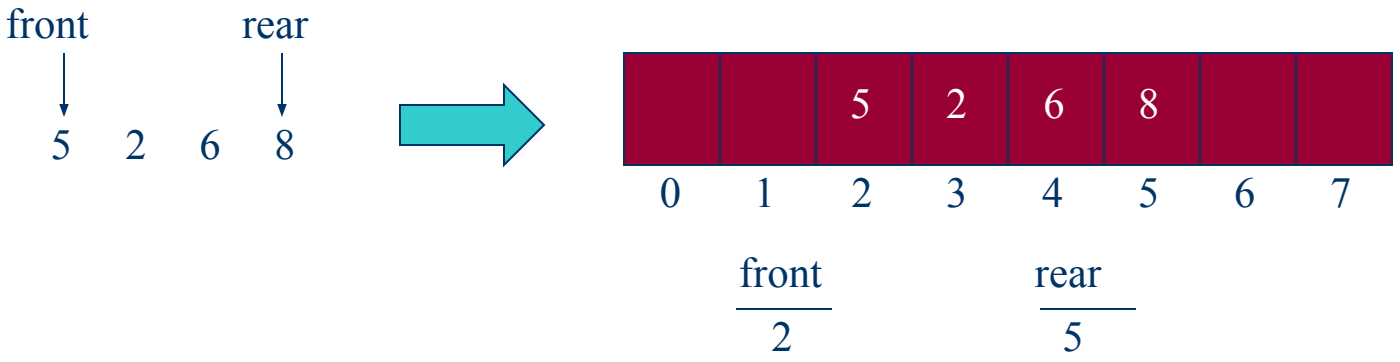
Queue using Array

dequeue()



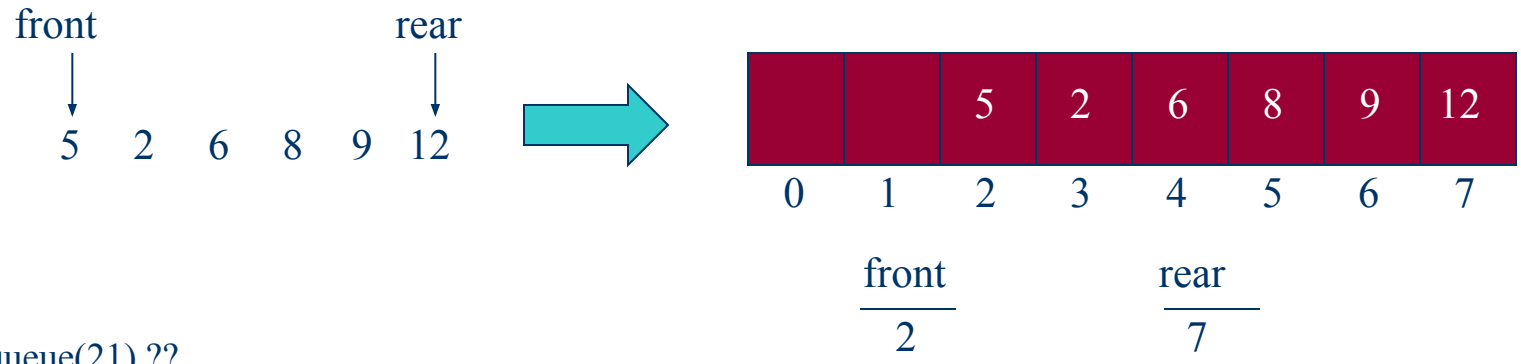
Queue using Array

dequeue()



Queue using Array

enqueue(9)
enqueue(12)



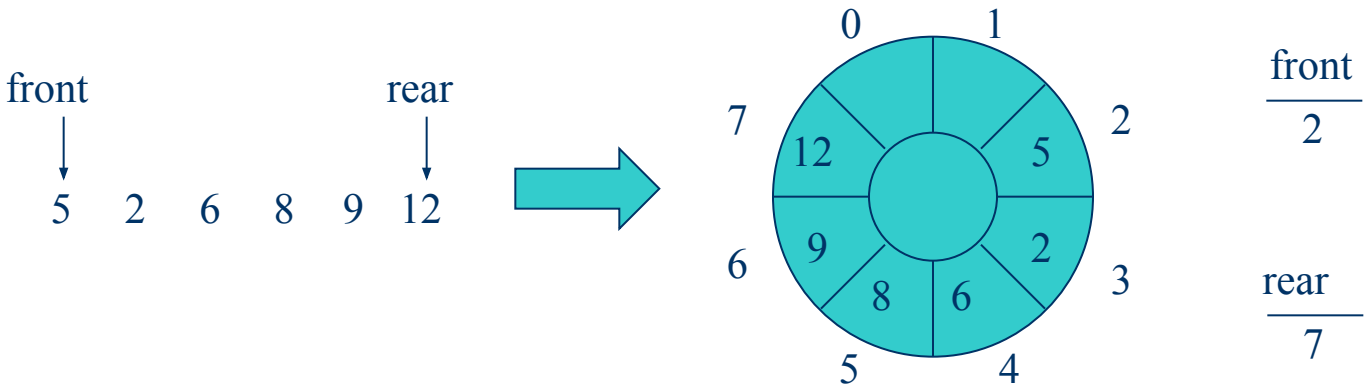
enqueue(21) ??

Queue using Array

- We have inserts and removal running in constant time but we created a new problem.
- Cannot insert new elements even though there are two places available at the start of the array.
- Solution: allow the queue to “wrap around”.

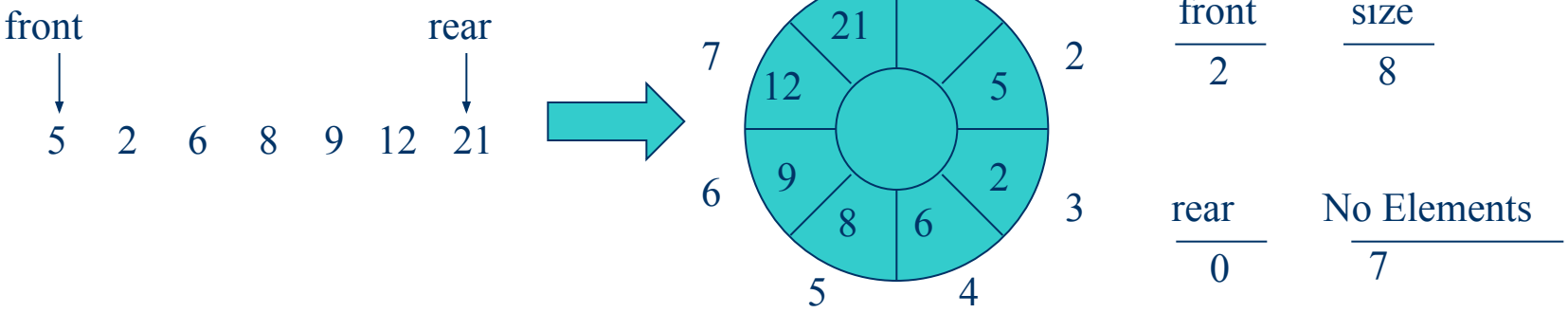
Queue using Array

- Basic idea is to picture the array as a *circular array*.



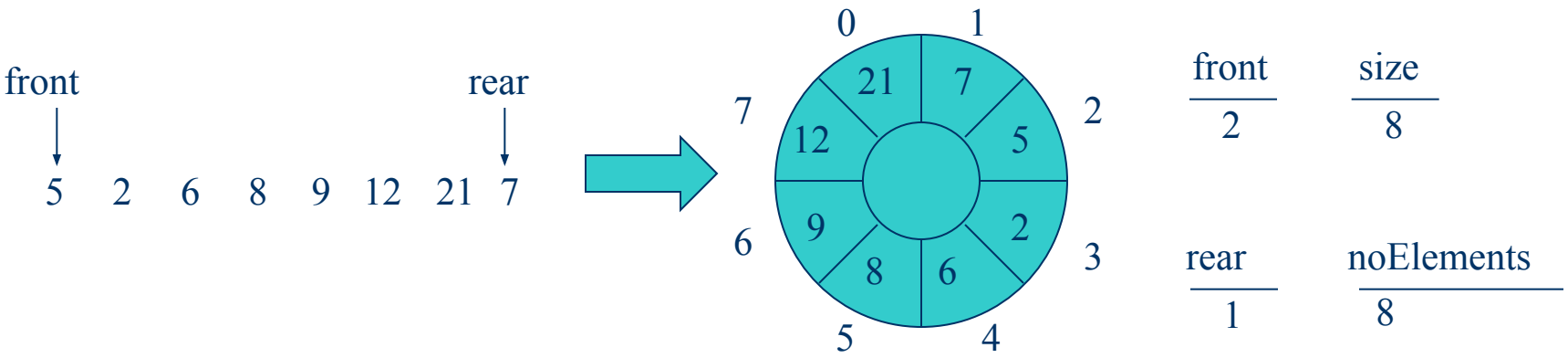
Queue using Array

enqueue(21)



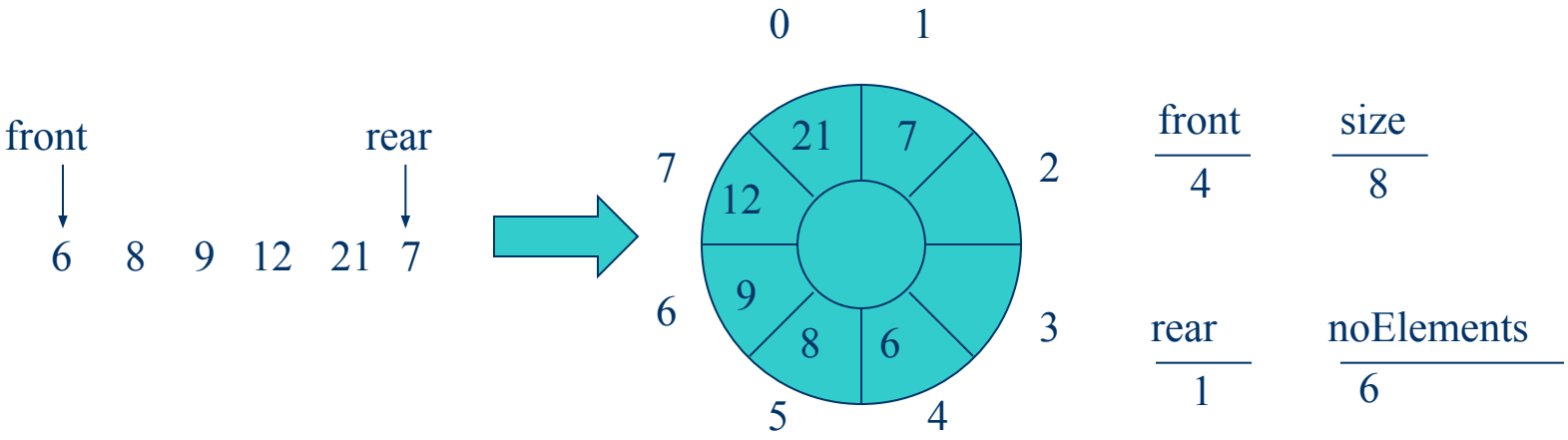
Queue using Array

enqueue(7)



Queue using Array

dequeue()

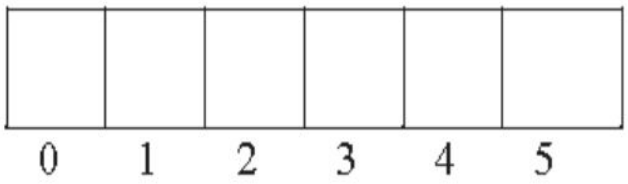


Circular Array

MaxSize = 6

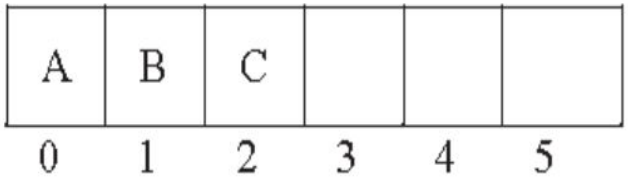
(a) Initially QUEUE is Empty

Front = -1
Rear = -1
Count = 0



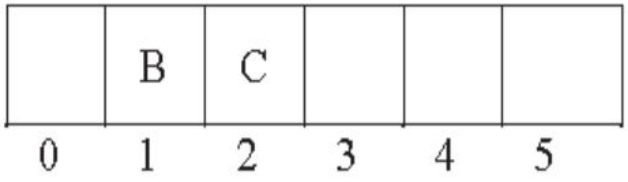
(b) A, B, C are Enqueued / Inserted

Front = 0
Rear = 2
Count = 3



(c) A is Deleted / Dequeue

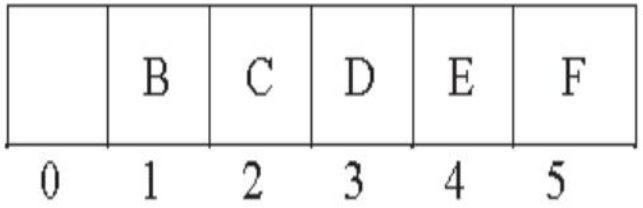
Front = 1
Rear = 2
Count = 2



Circular Array

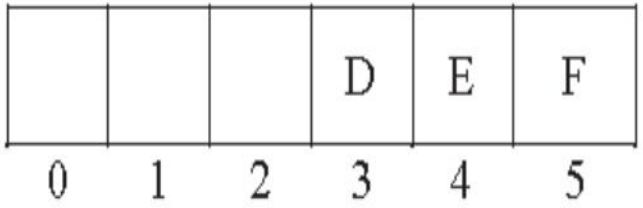
(d) D, E, F are Enqueued / Inserted

Front = 1
Rear = 5
Count = 5



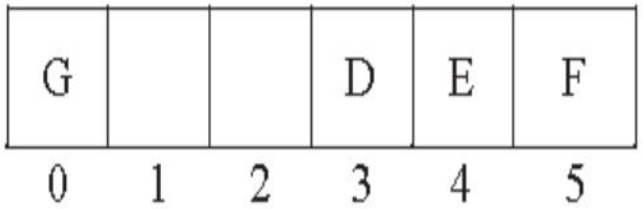
(e) B and C are Deleted / Dequeue

Front = 3
Rear = 5
Count = 3



(f) G is Enqueued / Inserted

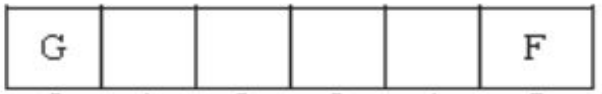
Front = 3
Rear = 0
Count = 4



Circular Array

(g) D and E are Deleted / Dequeue

Front = 5
Rear = 0
Count = 2



(h) H and I are Enqueued / Inserted

Front = 5
Rear = 2
Count = 4

(i) F is Deleted / Dequeue

Front = 0
Rear = 2
Count = 3

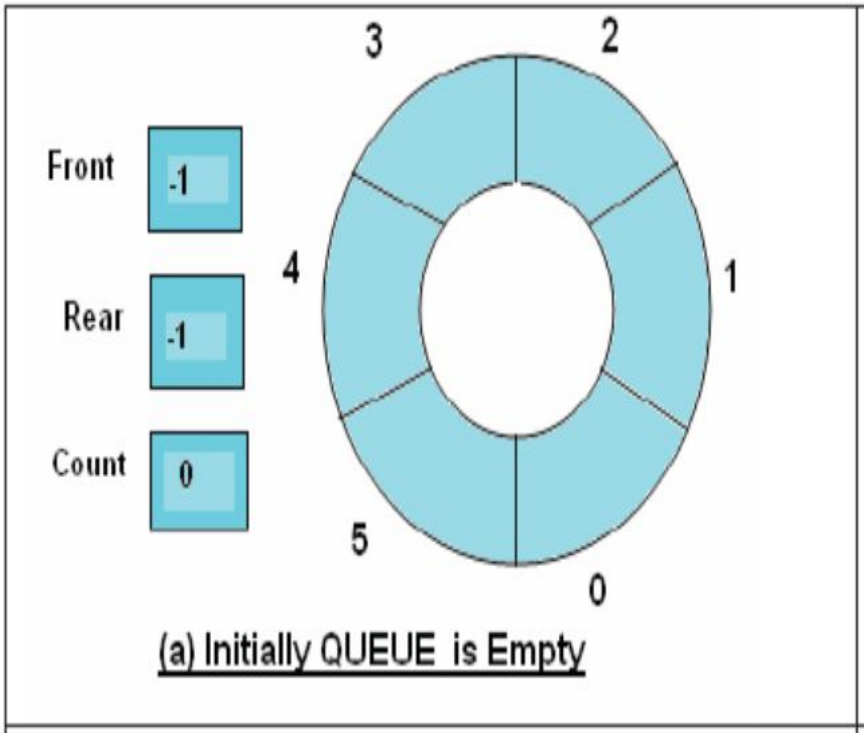
(j) G and H are Deleted / Dequeue

Front = 2
Rear = 2
Count = 1

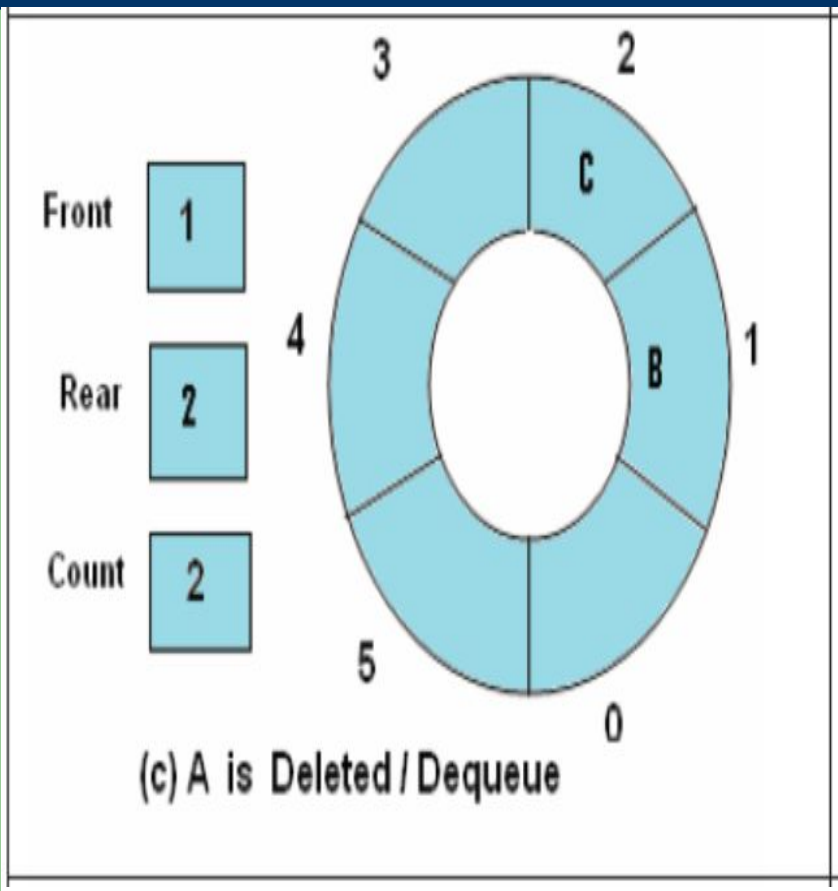
(k) I is Deleted. Queue is ***Empty***

Front = -1
Rear = -1
Count = 0

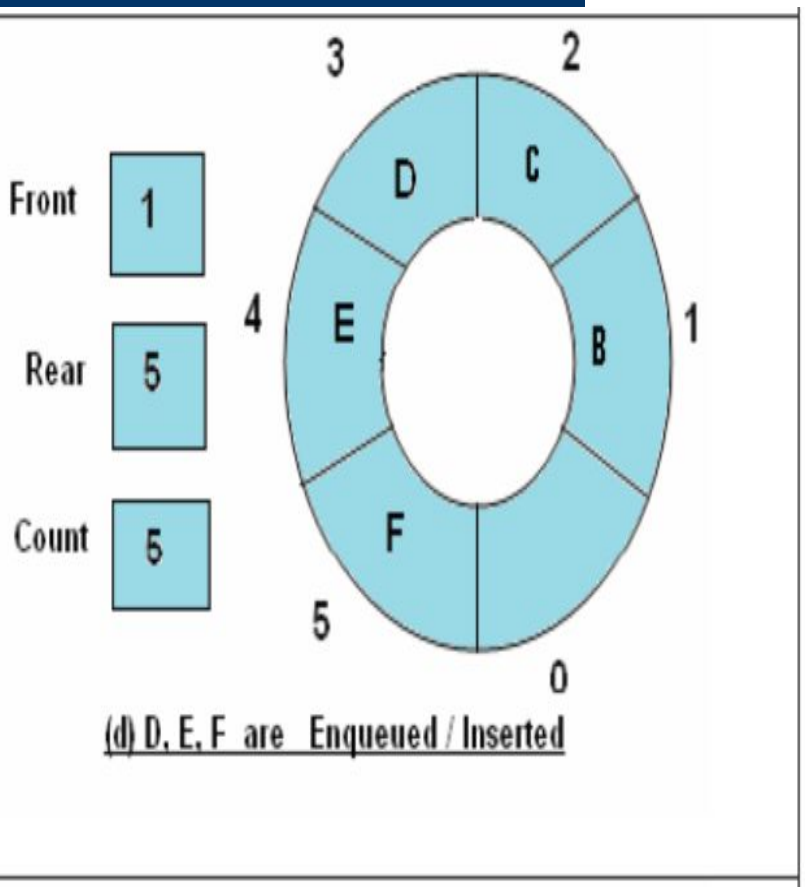
Circular Array



Circular Array

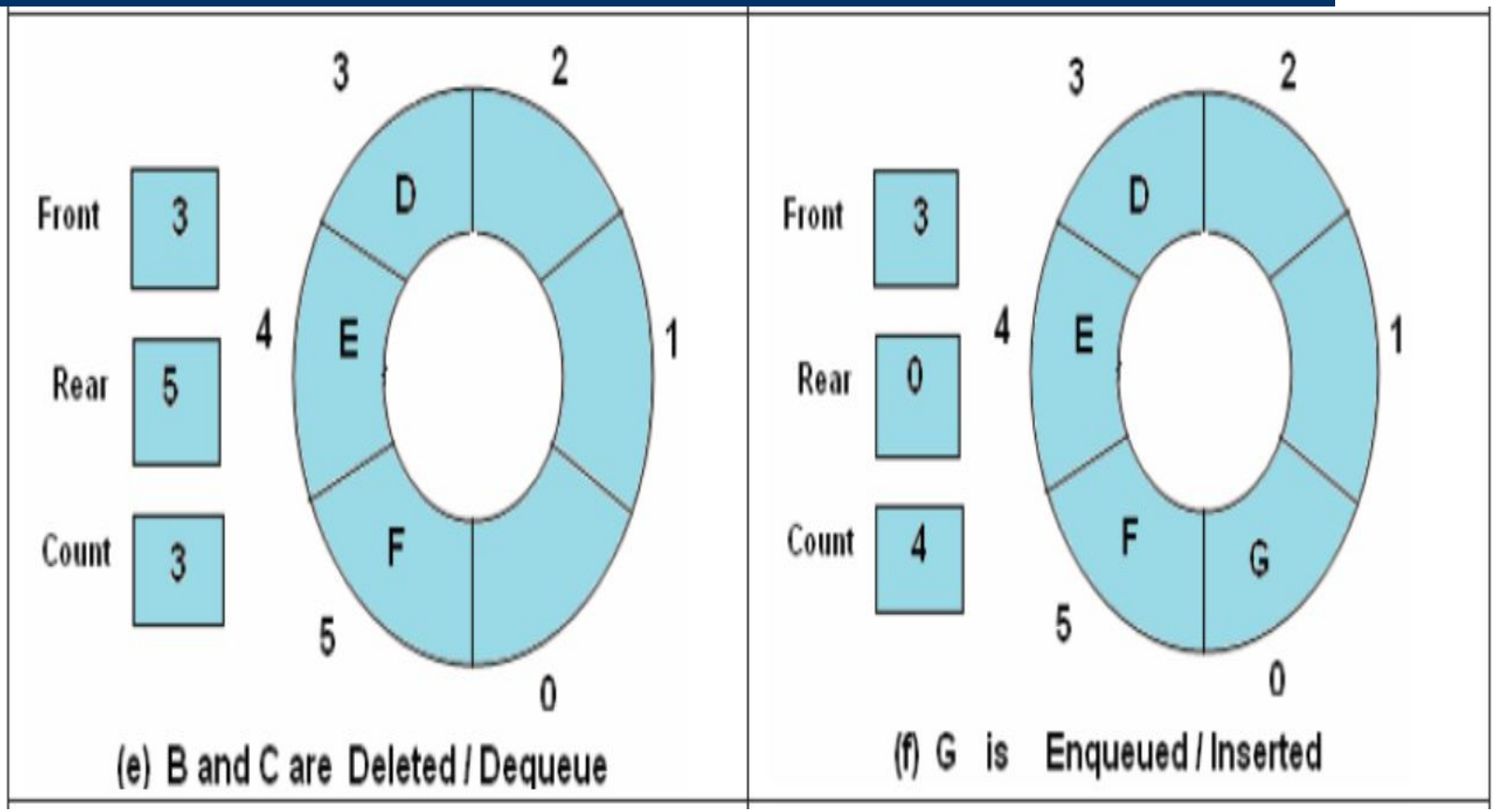


(c) A is Deleted / Dequeue

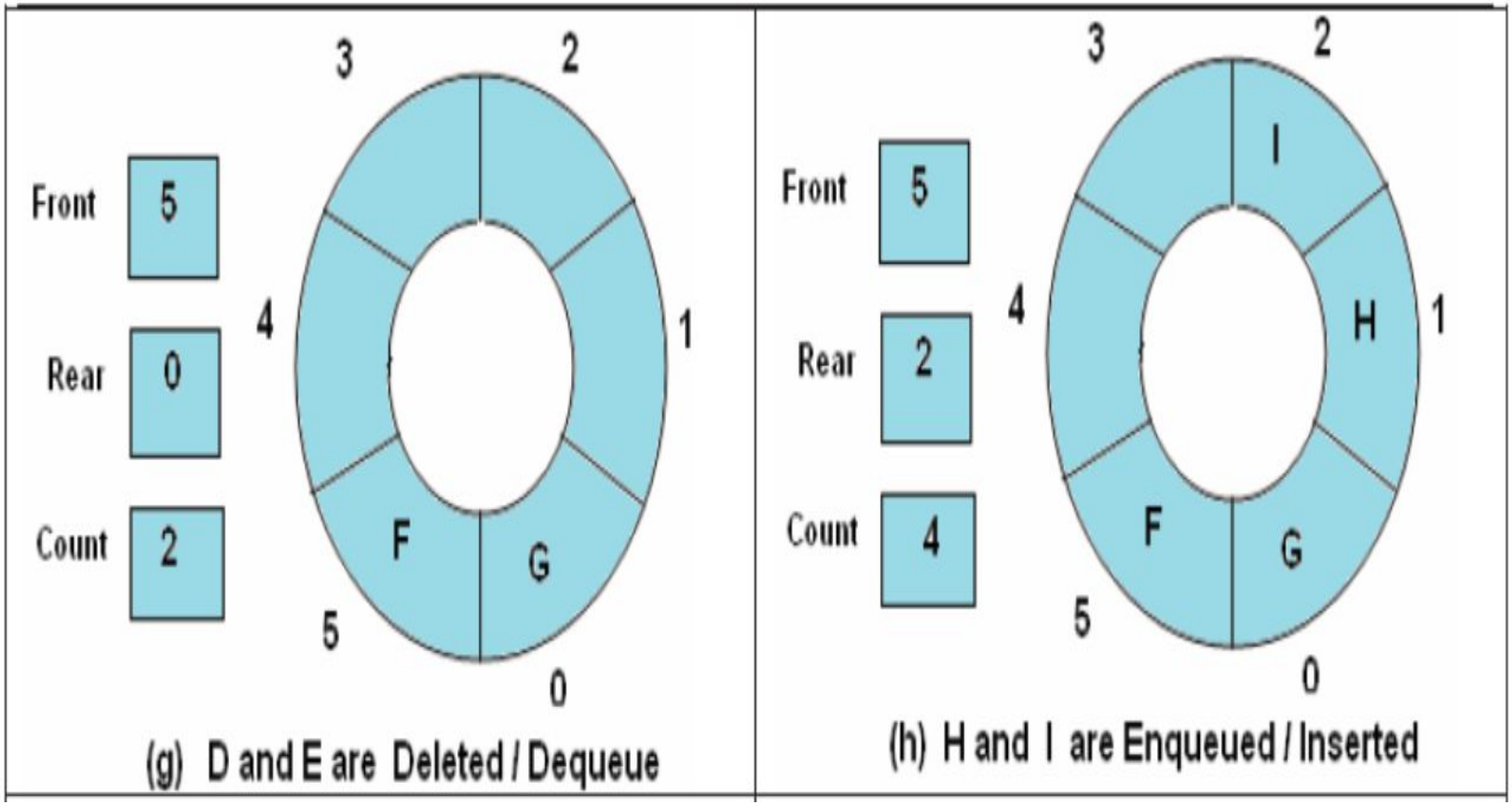


(d) D, E, F are Enqueued / Inserted

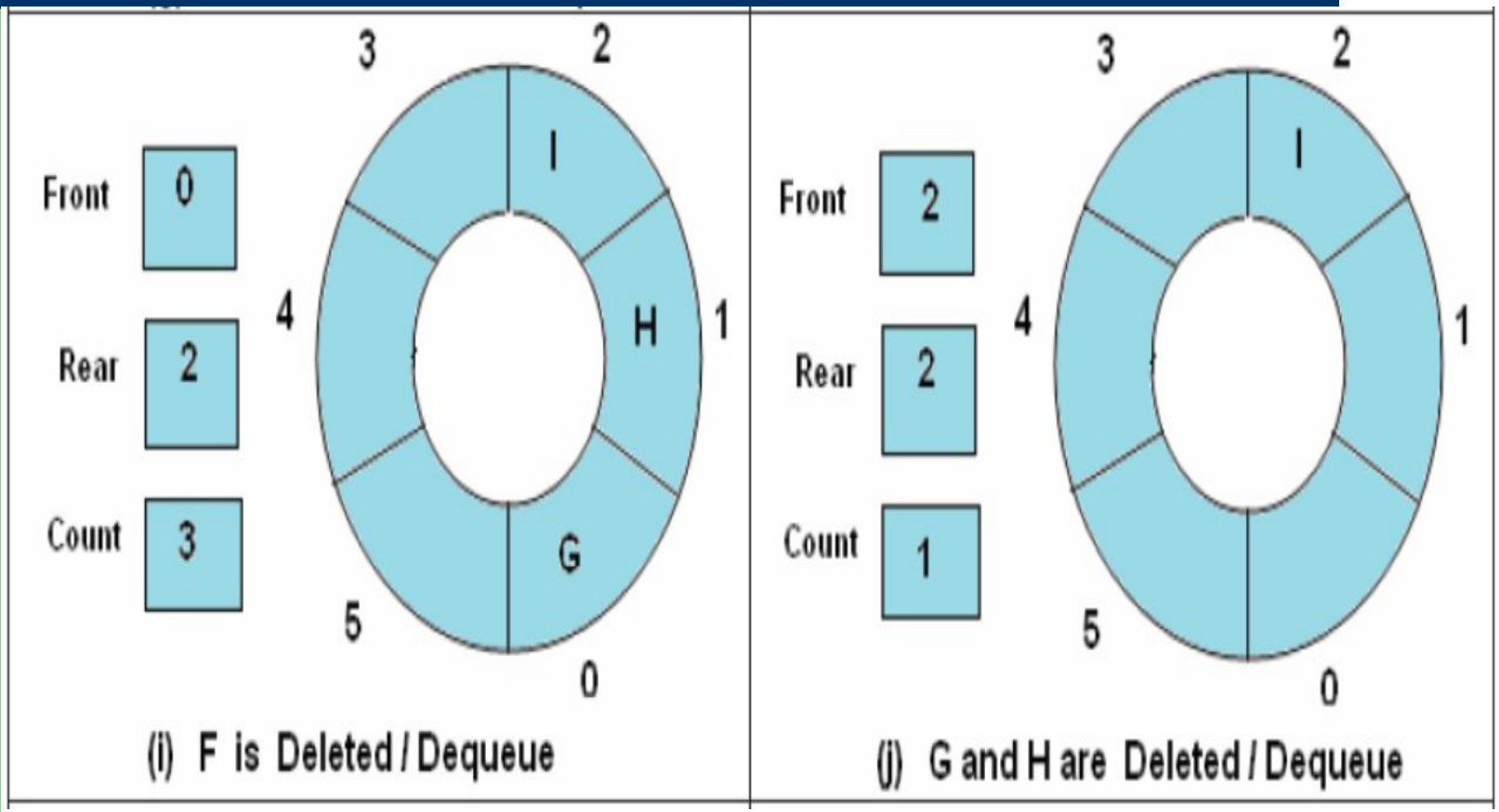
Circular Array



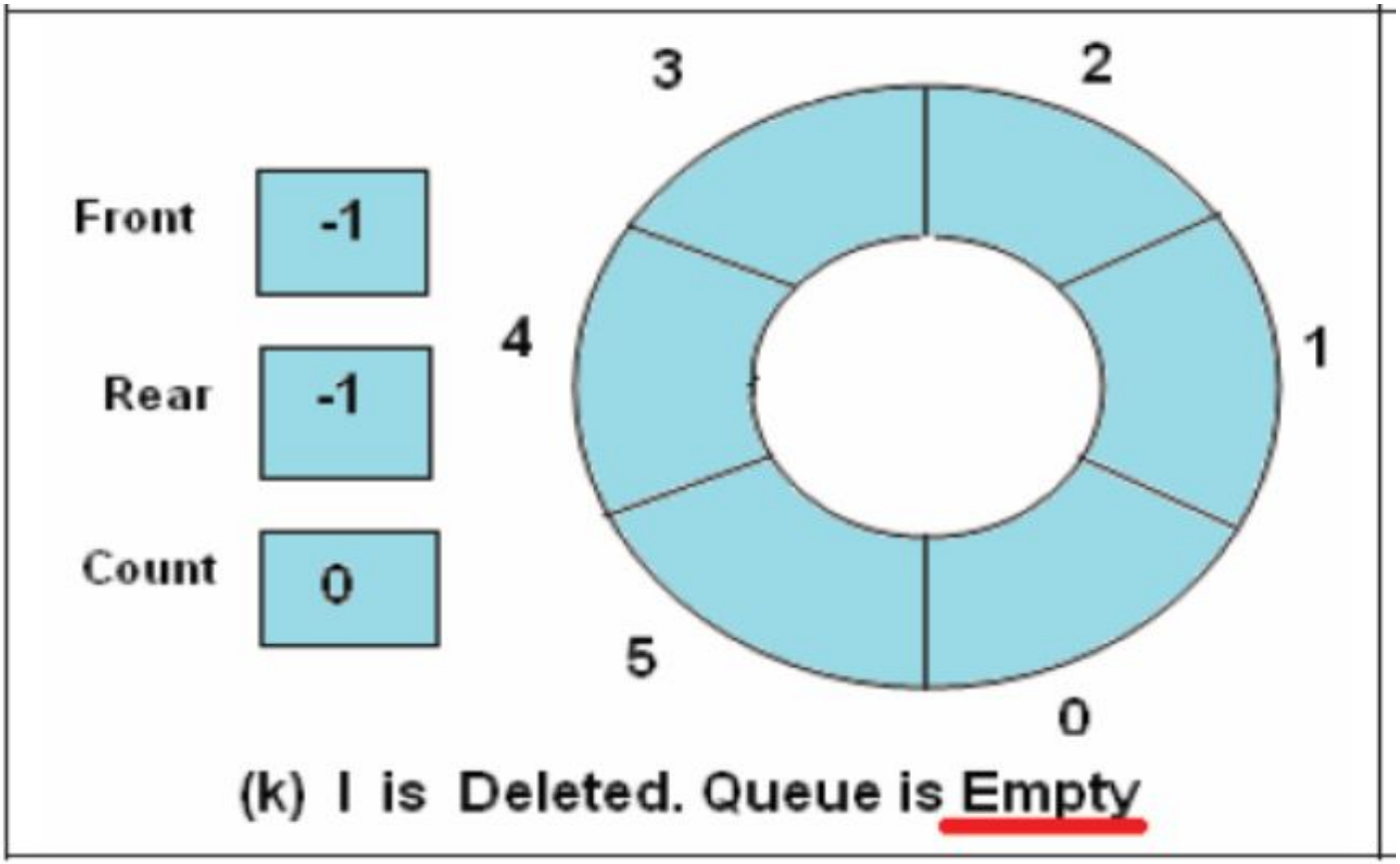
Circular Array



Circular Array



Circular Array



INSERTION TO A QUEUE

```
Algorithm: ENQUEUE(QUEUE, MAXSIZE, FRONT, REAR,COUNT, ITEM)
        This algorithm inserts an element ITEM into a circular queue.
1. [QUEUE already filled?]
        If COUNT = MAXSIZE then: [ COUNT is number of values in the QUEUE]
        Write: OVERFLOW, and Return.
2. [Find new value of REAR.]
        If COUNT= 0, then: [Queue initially empty.]
                Set FRONT= 0 and REAR = 0
        Else: if REAR = MAXSIZE - 1, then:
                Set REAR = 0
        Else:
                Set REAR = REAR+1.
        [End of If Structure.]
3. Set QUEUE[REAR] = ITEM. [This insert new element.]
4. COUNT=COUNT+1 [ Increment to Counter. ]
5. Return.
```

DELETION FROM A QUEUE

Algorithm: DEQUEUE(QUEUE, MAXSIZE, FRONT, REAR,COUNT, ITEM)

This procedure deletes an element from a queue and assigns it to the variable ITEM.

1. [QUEUE already empty?]
   If COUNT= 0, then: Write: UNDERFLOW, and Return.
2. Set ITEM = QUEUE[FRONT].
3. Set COUNT = COUNT -1
4. [Find new value of FRONT.]
   If COUNT = 0, then: [There was one element and has been deleted]
      Set FRONT= -1, and REAR = -1.
   Else if FRONT= MAXSIZE, then: [Circular, so set Front = 0]
      Set FRONT = 0
   Else:
      Set FRONT:=FRONT+1.
   [End of If structure.]
5. Return ITEM

DEQUEUE

- DEQUEUE---□ Double Ended Queue
- Elements can be added or removed at either end but not in the middle
- A DEQUEUE is maintained by a circular array with pointers LEFT and RIGHT which points to the two ends of dequeue
- Variations of dequeue
  - Input restricted dequeue
  - Output restricted dequeue

PRIORITY QUEUE

- A collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules
  - An element of higher priority is processed before any element of lower priority.
  - Two elements with the same priority are processed according to the order in which they were added to the queue

PRIORITY QUEUE

- Priority queue can be maintained in computer memory either
  - As a one way list
    - Each node in the list contains three items of information
      - Information field INFO
      - Priority number PRN
      - Link number LINK
  - Array representations of priority queues
    - Multiple queues are maintain for each level of priority
    - Each such queue appears in its own circular array
    - A two-dimensional array can be used instead of multiple arrays where every row index corresponds to a priority number.