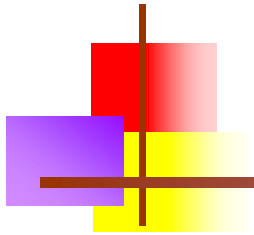


CSE 207

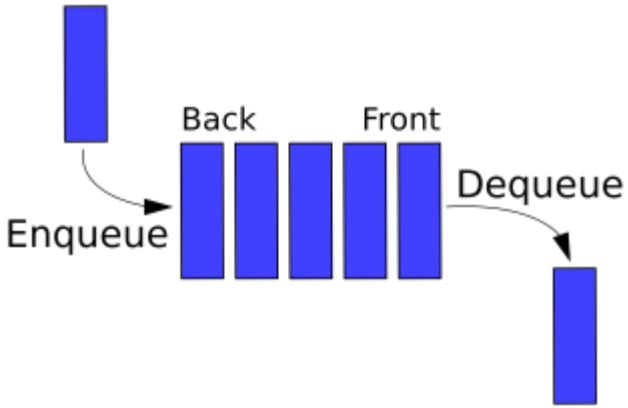
Queue

Queues, and Deques

- A queue is a first in, first out (FIFO) data structure
  - Items are removed from a queue in the same order as they were inserted
- A deque is a double-ended queue—items can be inserted and removed at either end

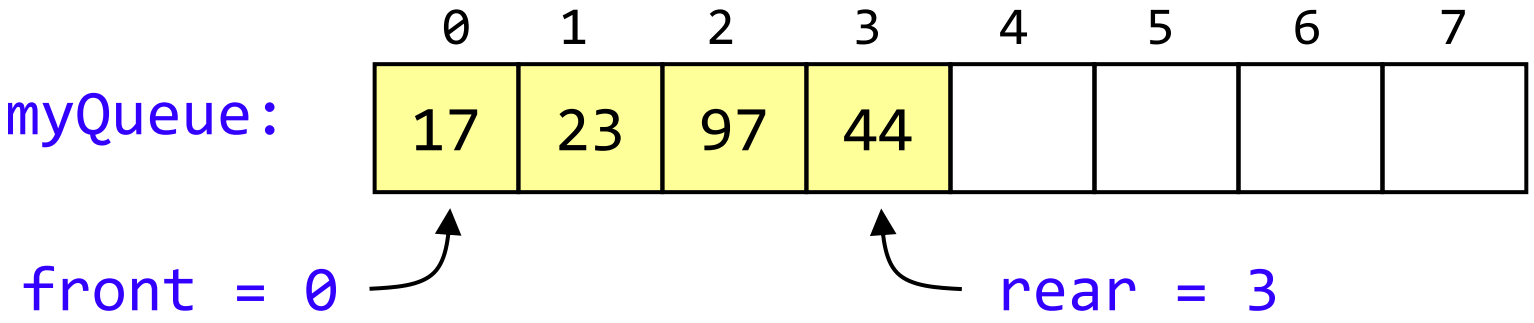
Queue Operations

- Enqueue() : Add an element to the rear of the queue.
- Dequeue() : Remove and returns the element at the front of the queue.
- isEmpty() : returns true if queue is empty.
- isFull() : returns true if queue is full.



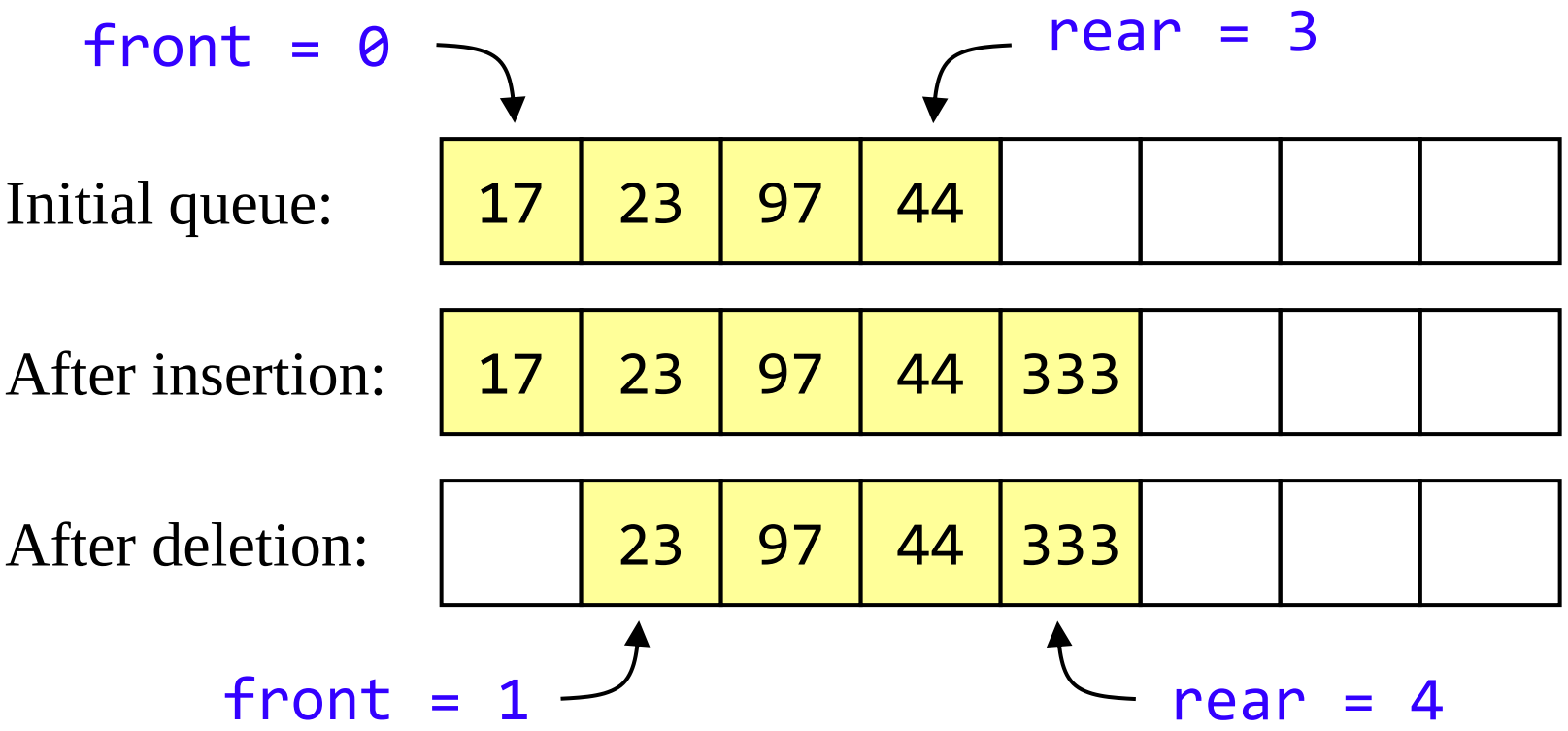
Array implementation of queues

- A queue is a first in, first out (FIFO) data structure
- This is accomplished by inserting at one end (the rear) and deleting from the other (the front)



- **To insert:** put new element in location 4, and set rear to 4
- **To delete:** take element from location 0, and set front to 1

Array implementation of queues

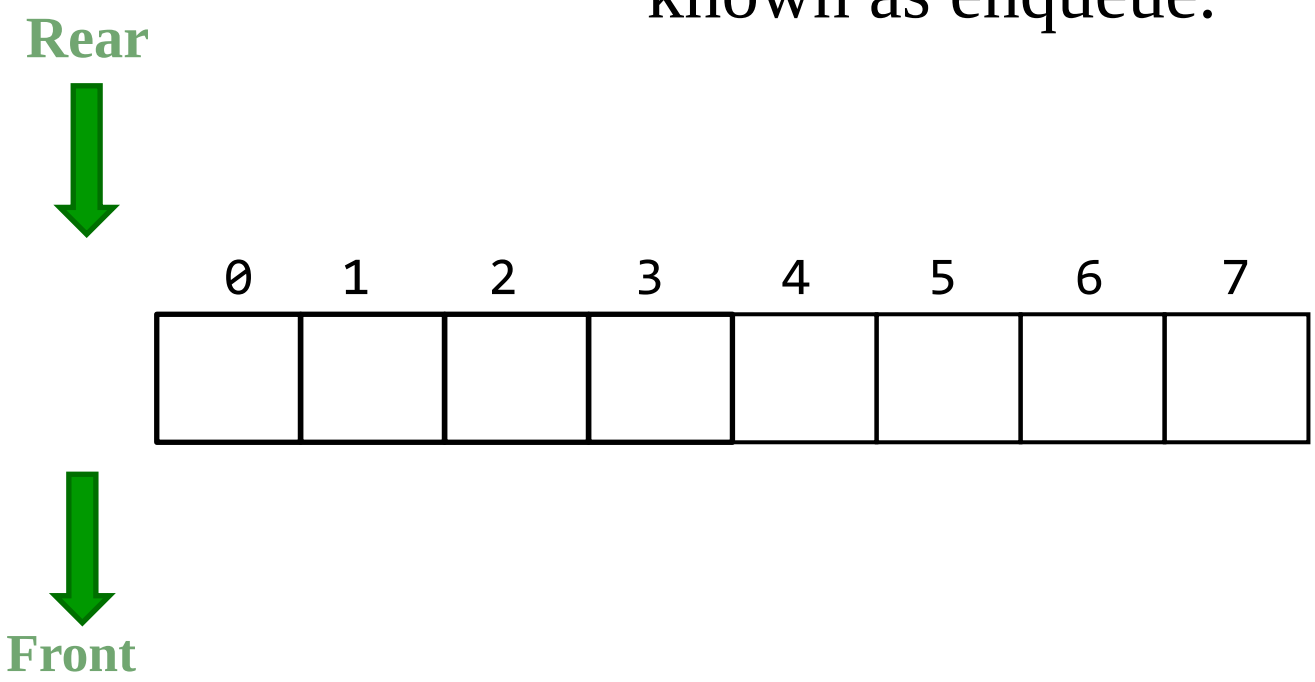


- Notice how the array contents "crawl" to the right as elements are inserted and deleted
- This will be a problem after a while!

Enqueue()

- Inserting 15

- The queue insert operation is known as enqueue.

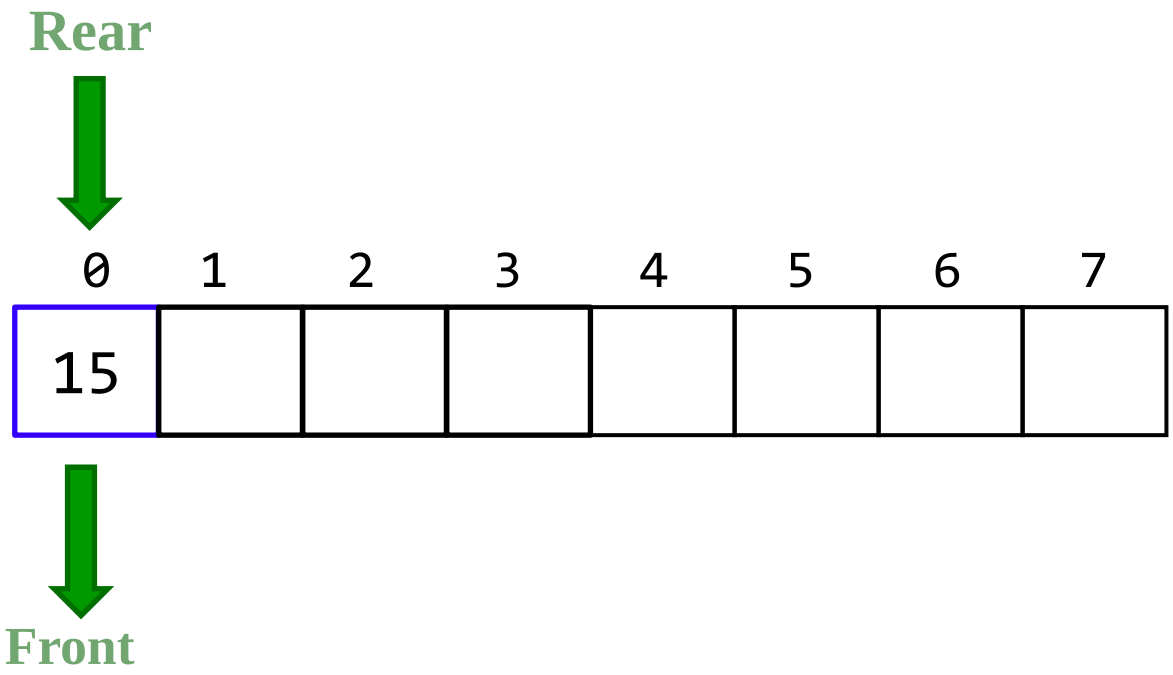


Front=-1

Rear=-1

Enqueue()

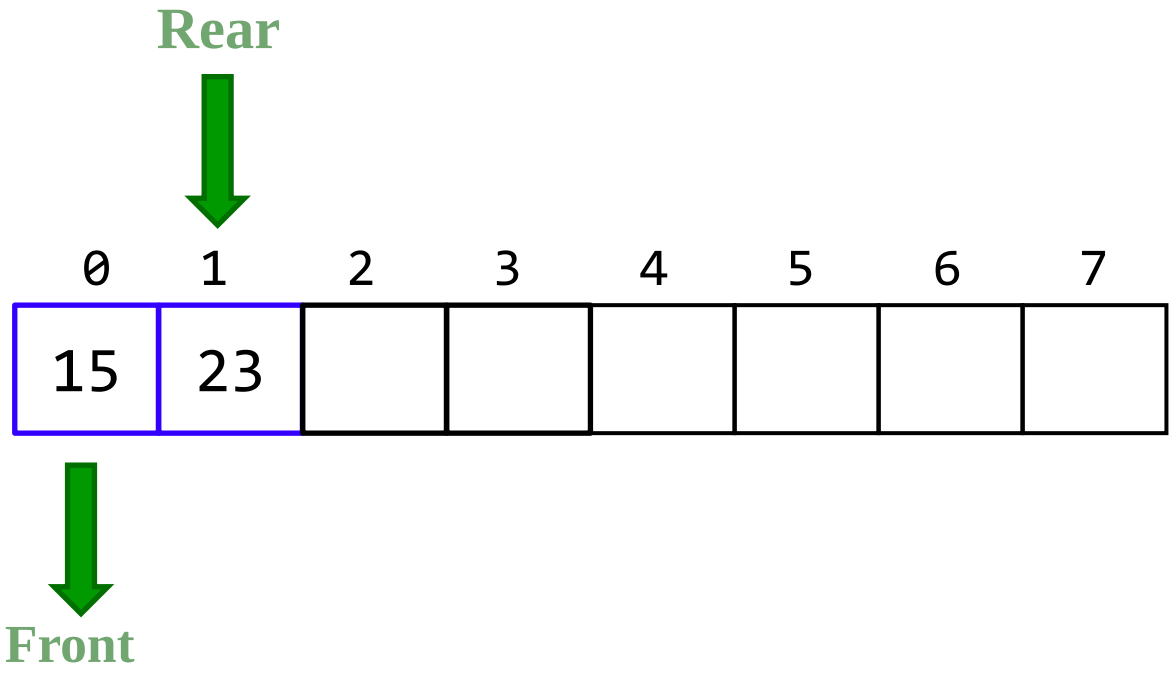
- Inserting 23



Front= 0 **Rear= 0**

Enqueue()

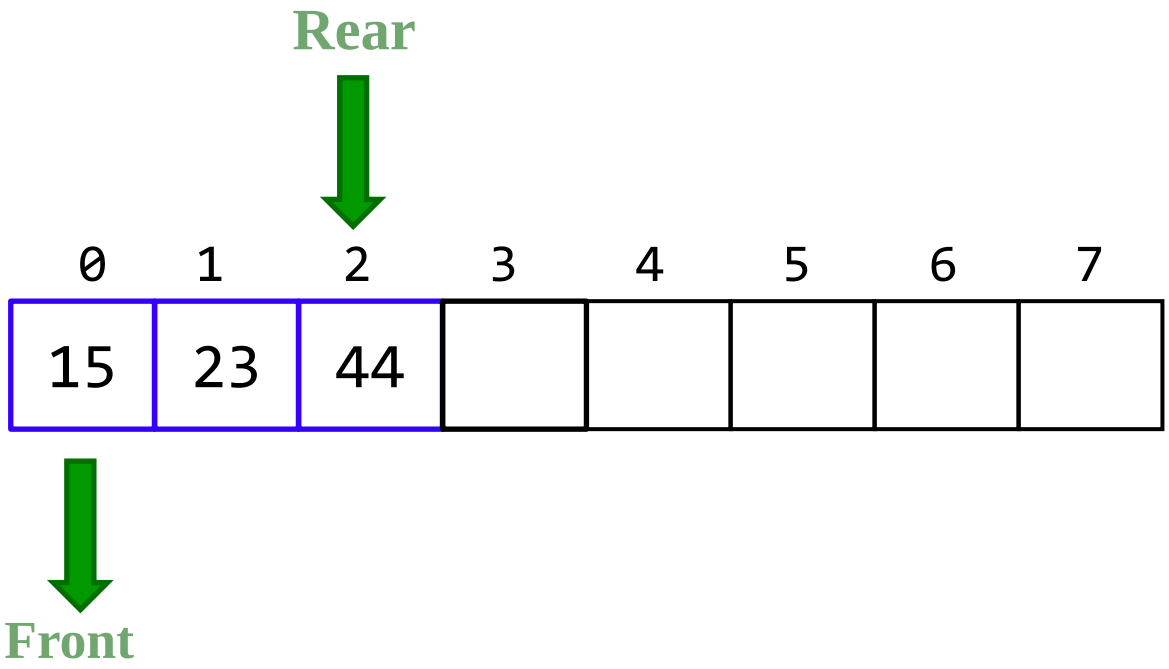
- Inserting 44



Front= 0 **Rear= 1**

Enqueue()

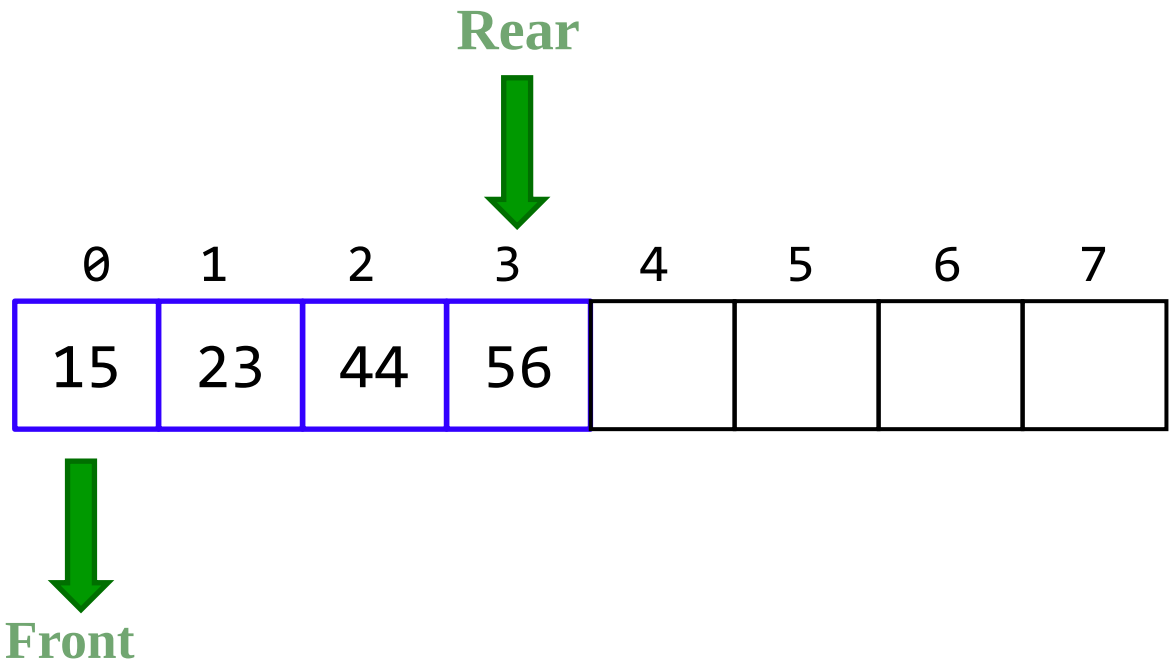
- Inserting 56



Front= 0 **Rear= 2**

Enqueue()

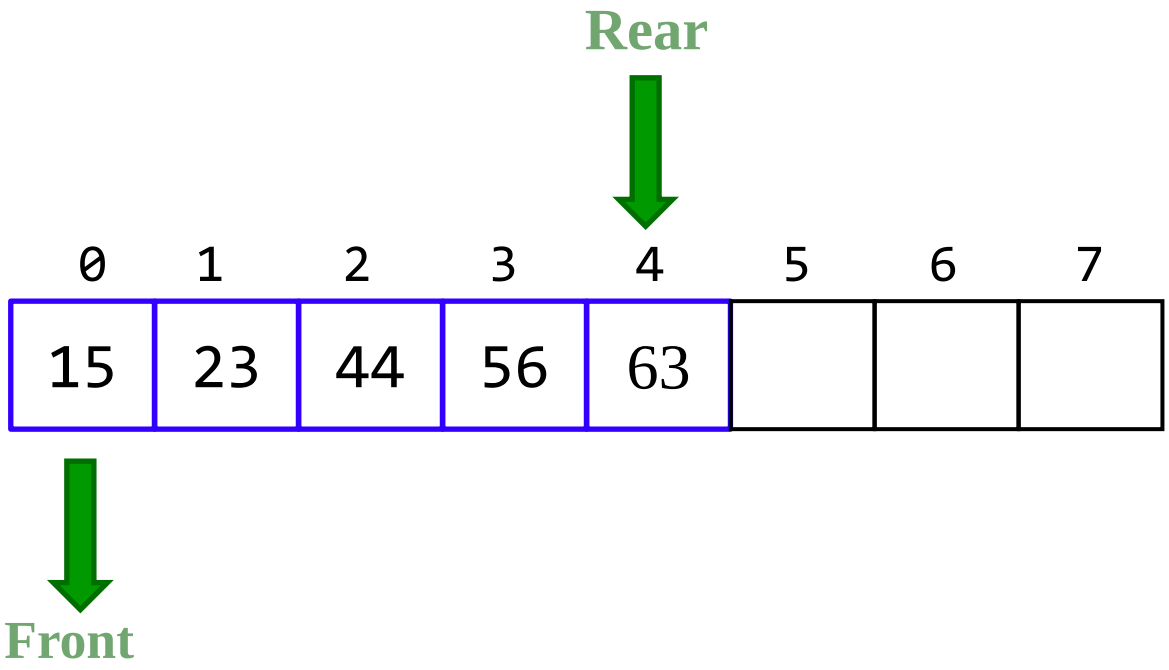
- Inserting 63



Front= 0 **Rear= 3**

Enqueue()

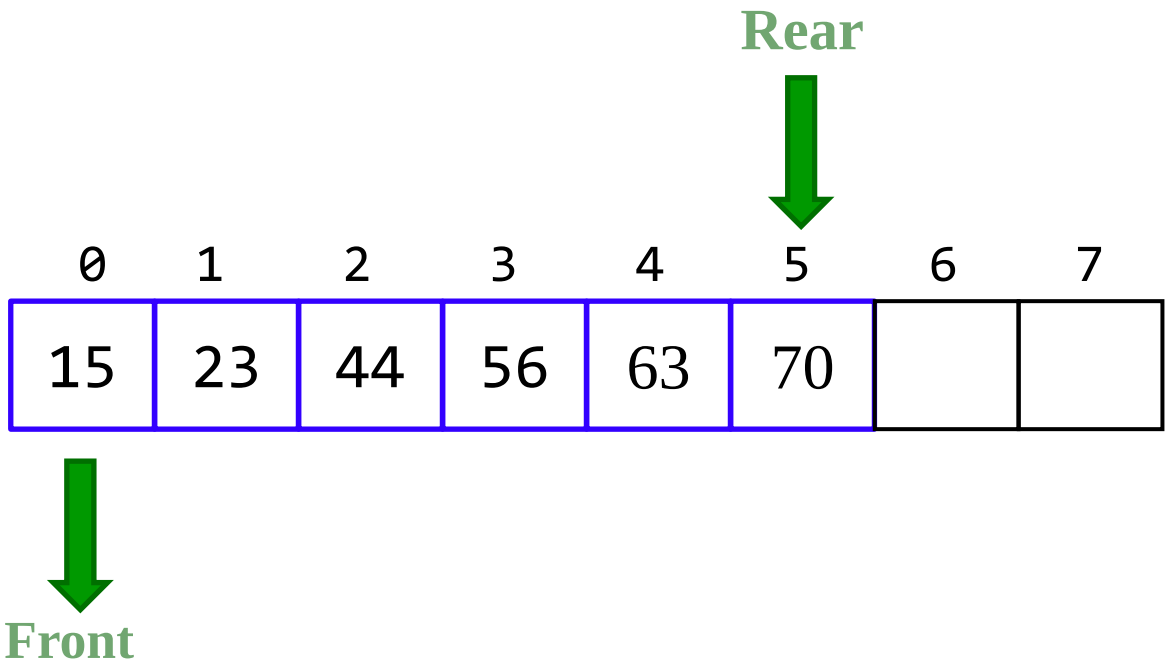
- Inserting 70



Front= 0 **Rear= 4**

Enqueue()

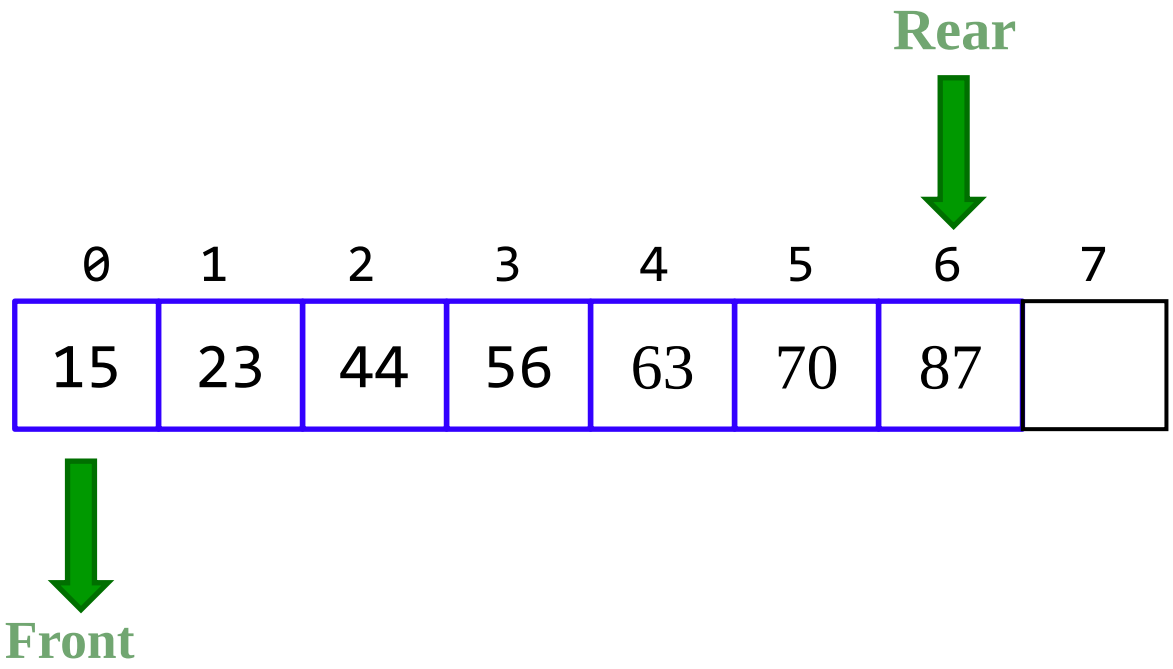
- Inserting 87



Front= 0 **Rear= 5**

Enqueue()

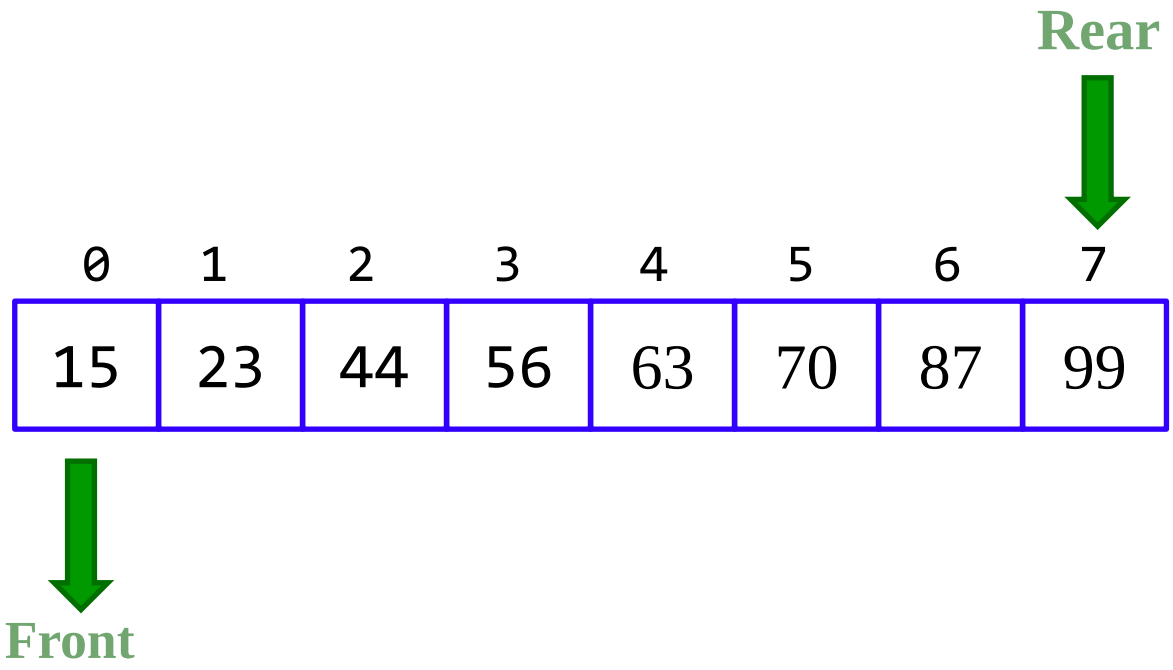
- Inserting 99



Front= 0 **Rear= 6**

Enqueue()

- Inserting 99

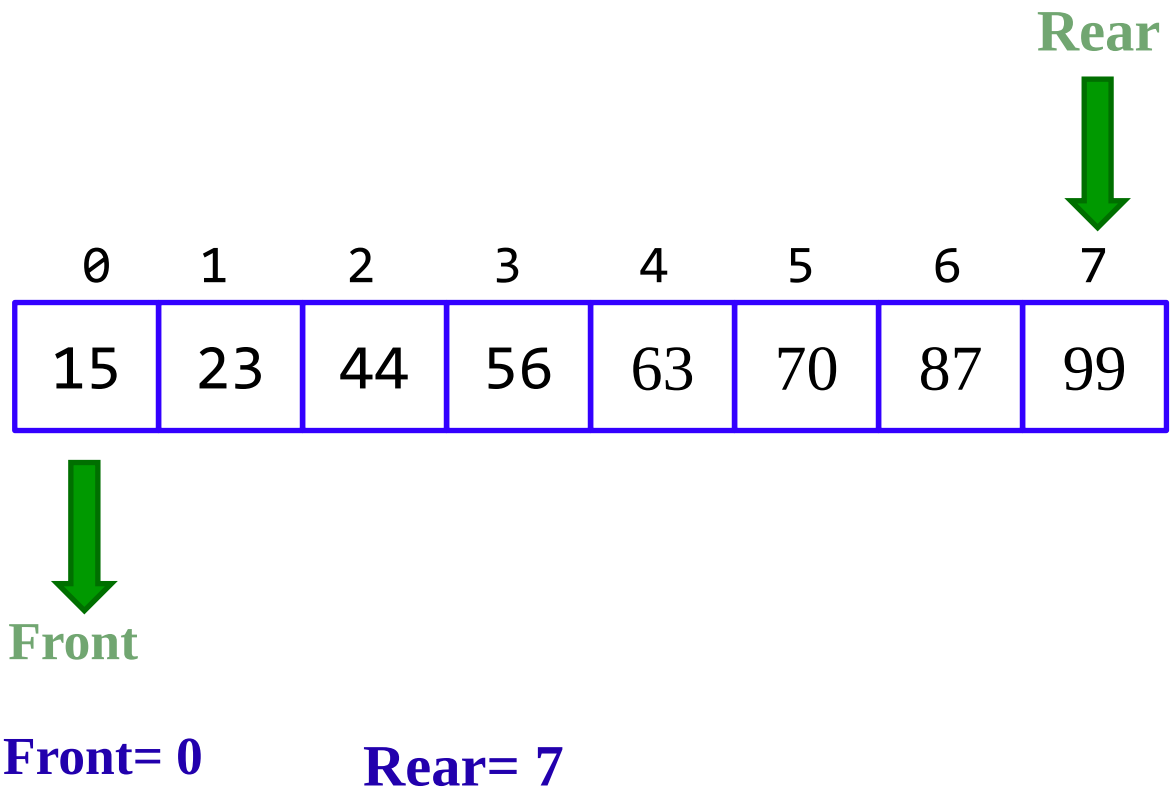


Front= 0 **Rear= 7**

Dequeue()

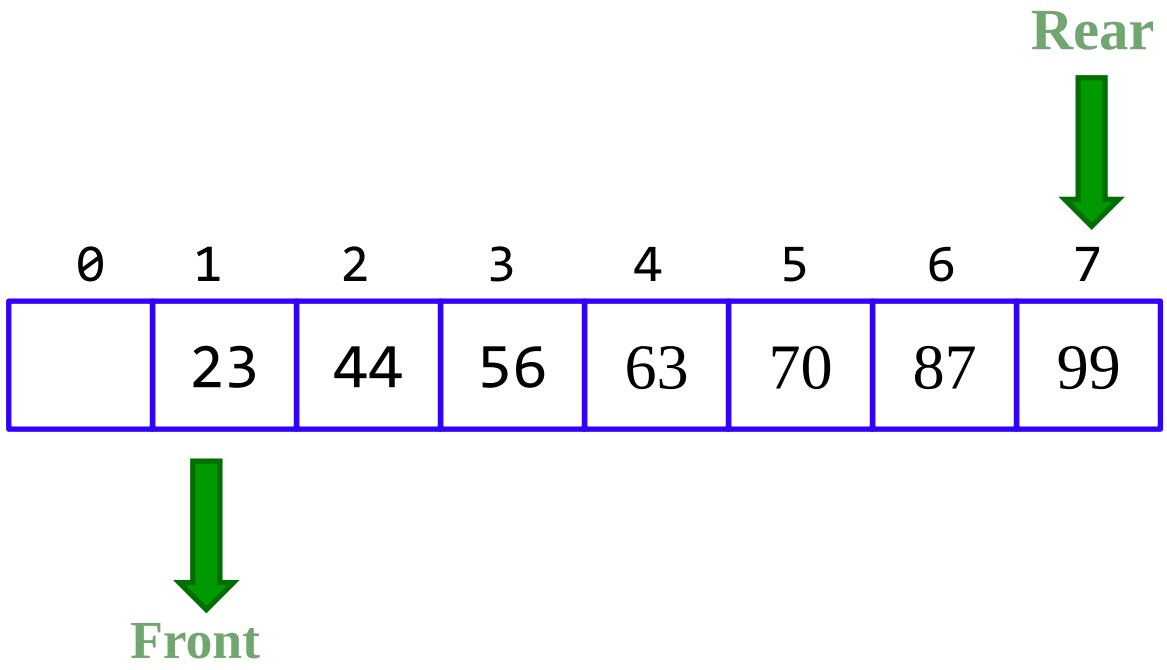
- Deleting 15

- The queue delete operation is known as dequeue.



Dequeue()

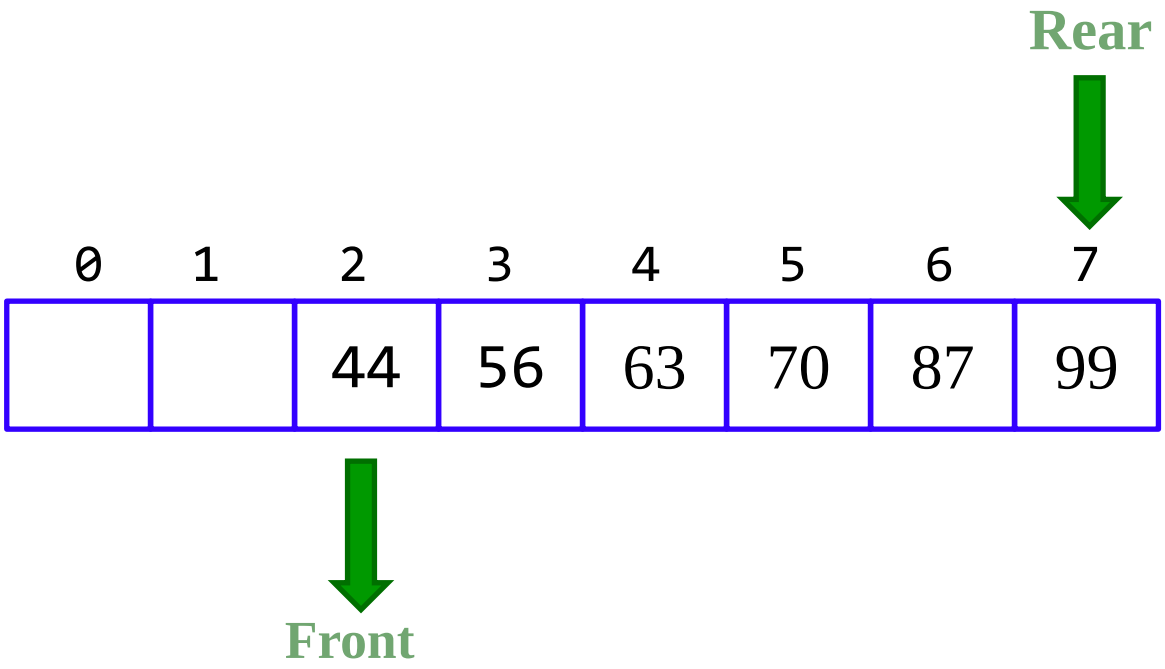
- Deleting 23



Front= 1 **Rear= 7**

Dequeue()

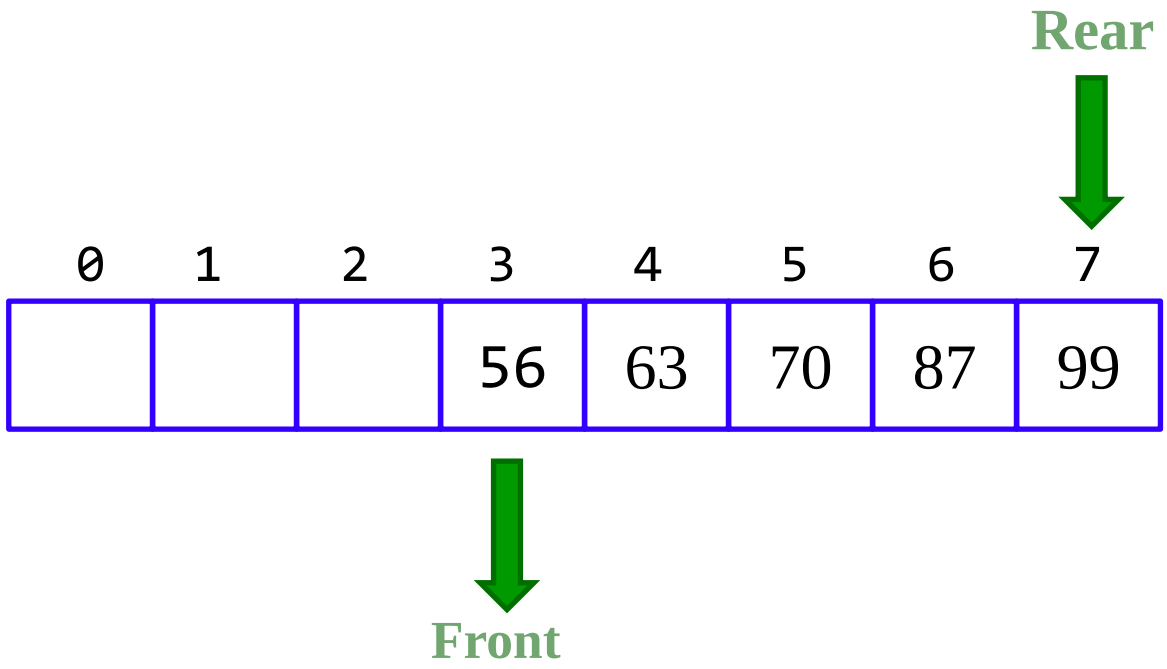
- Deleting 44



Front= 2 **Rear= 7**

Dequeue()

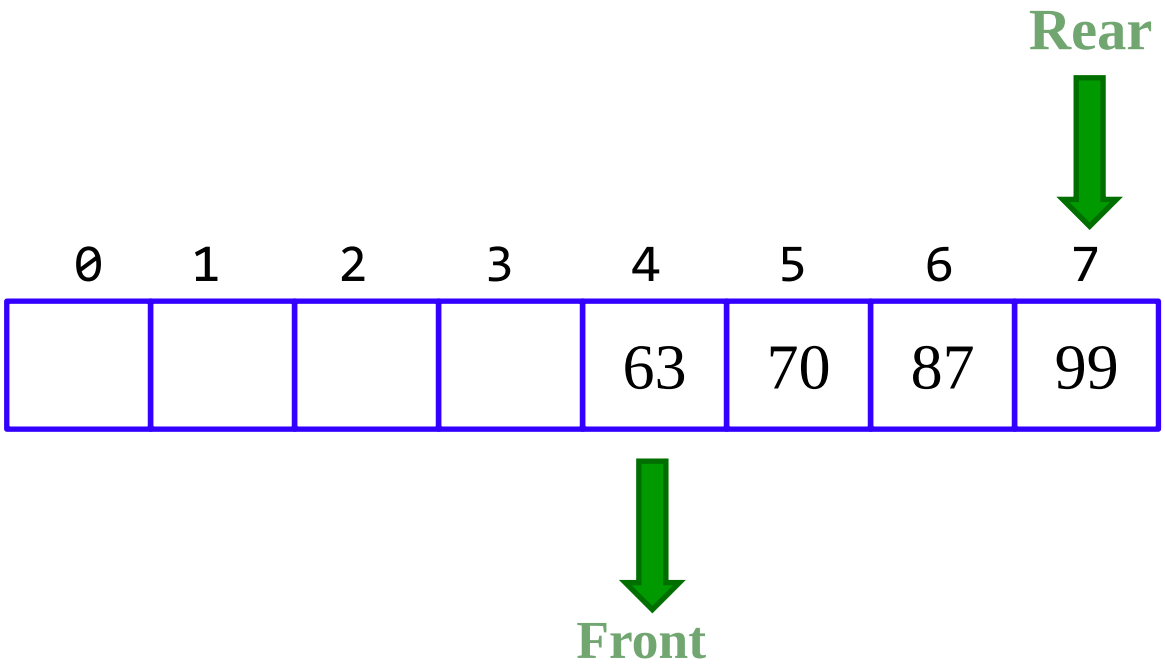
- Deleting 56



Front= 3 **Rear= 7**

Dequeue()

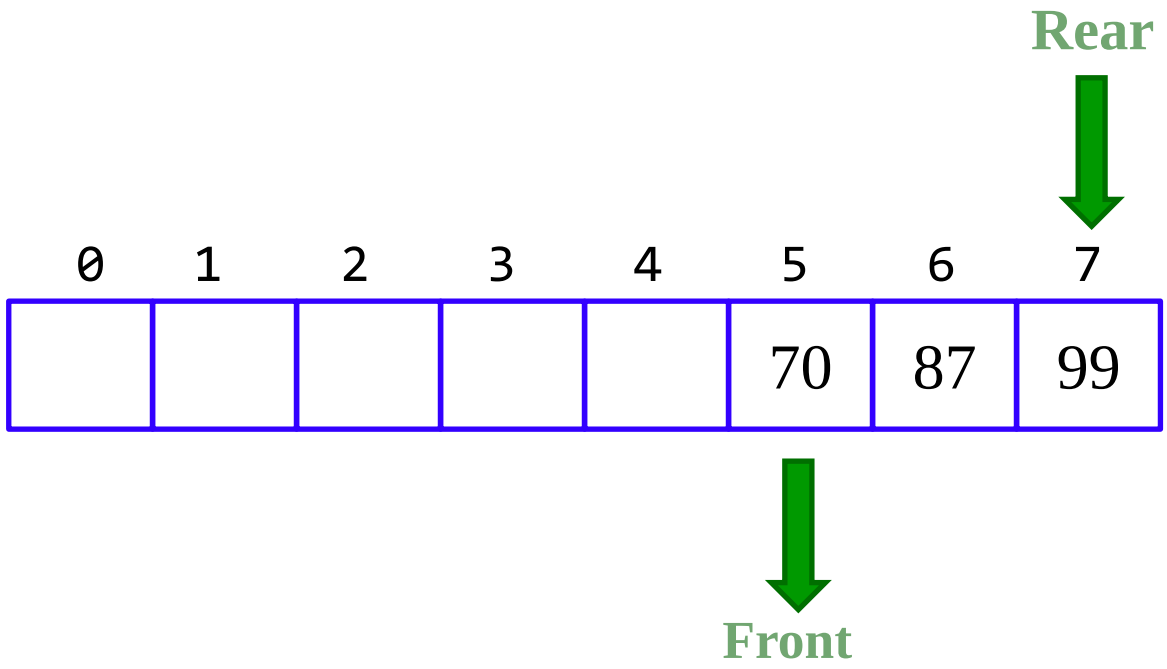
- Deleting 63



Front= 4 **Rear= 7**

Dequeue()

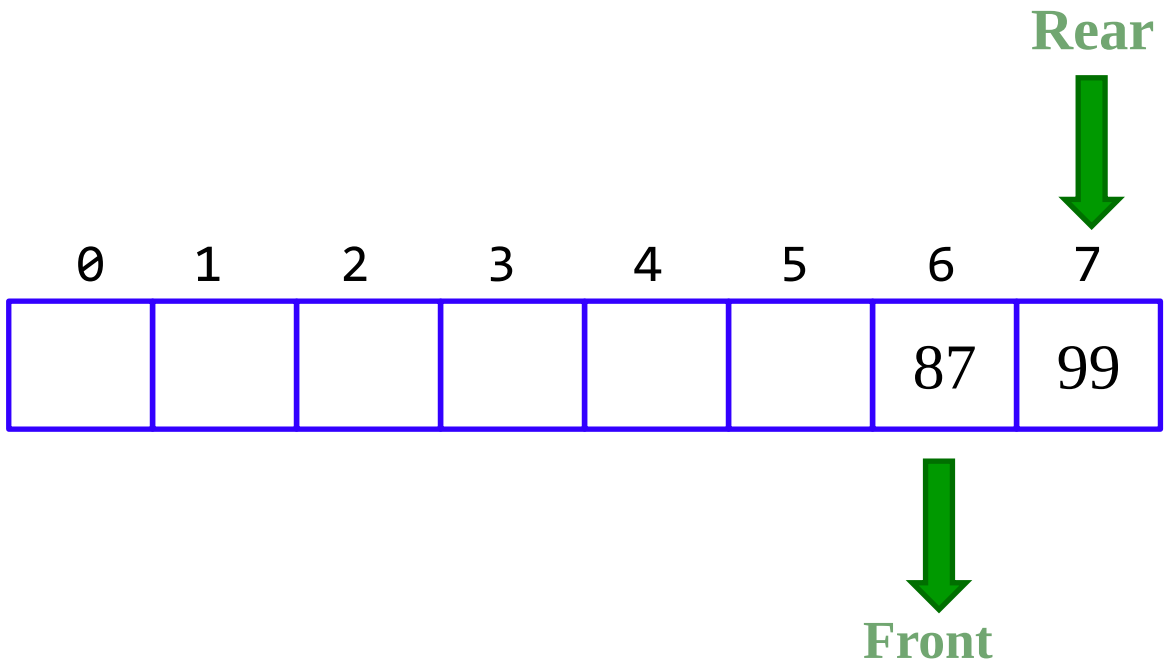
- Deleting 70



Front= 5 **Rear= 7**

Dequeue()

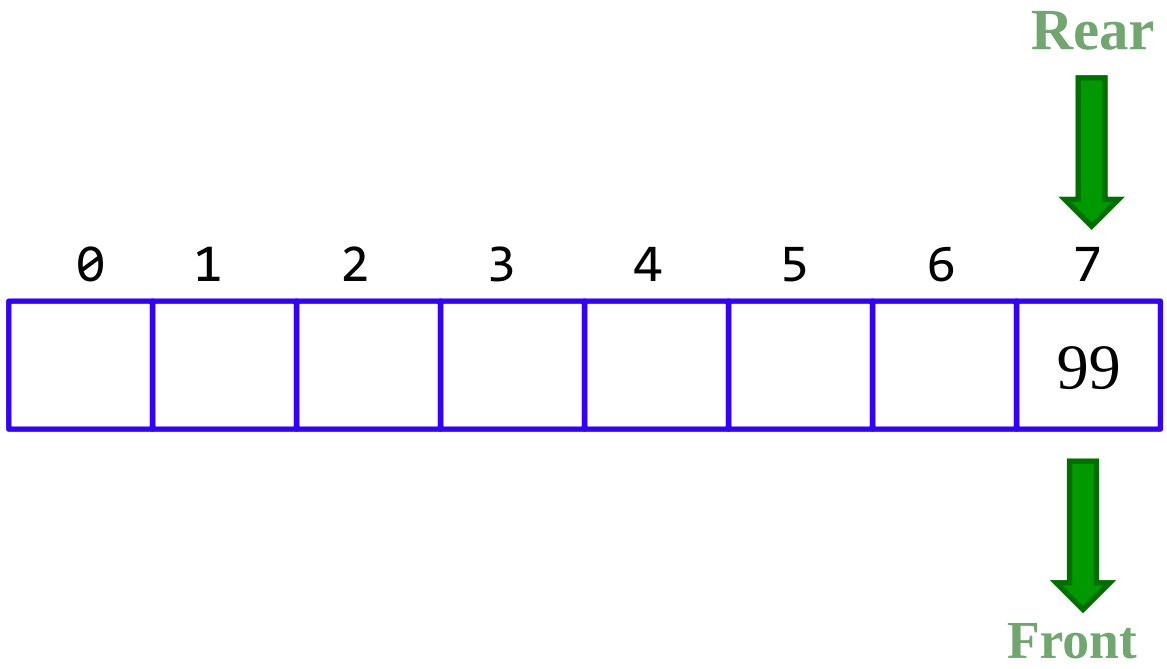
- Deleting 87



Front= 6 **Rear= 7**

Dequeue()

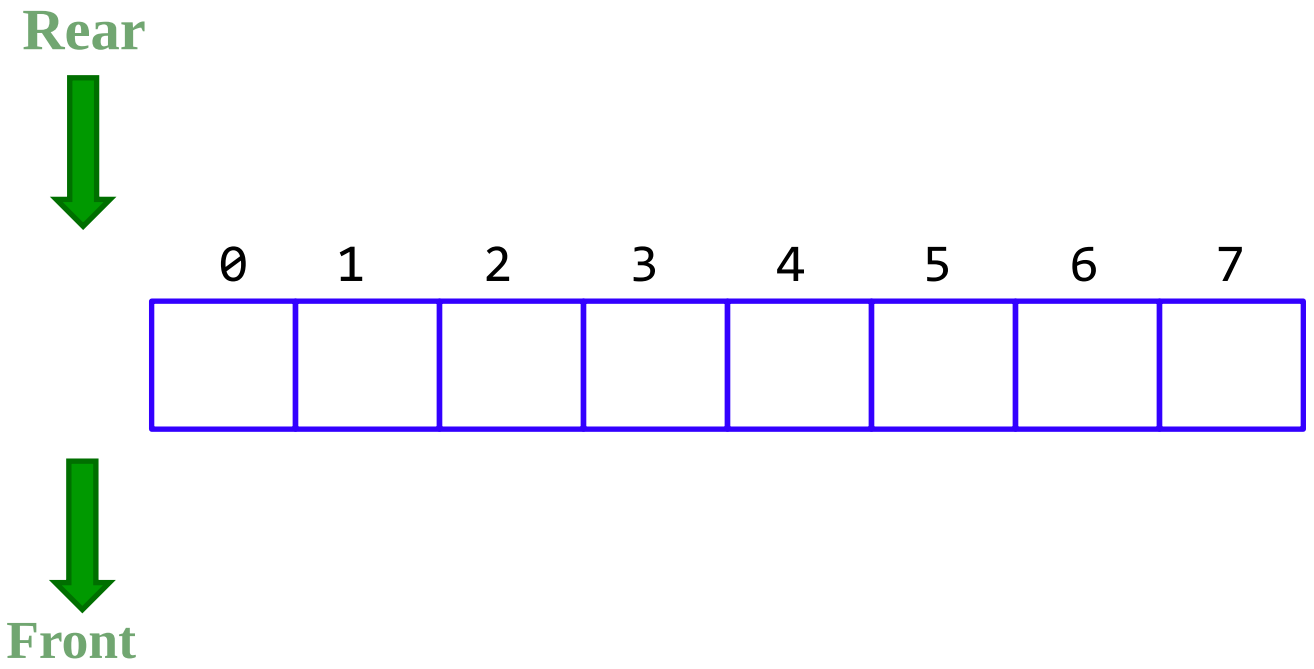
- Deleting 99



Front= 7

Rear= 7

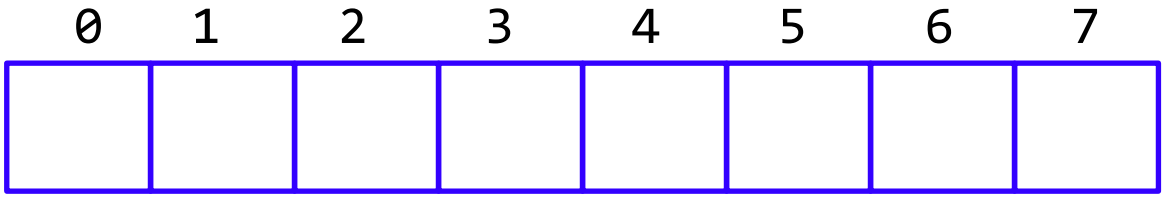
Dequeue()



Front= -1 **Rear= -1**

IsEmpty()

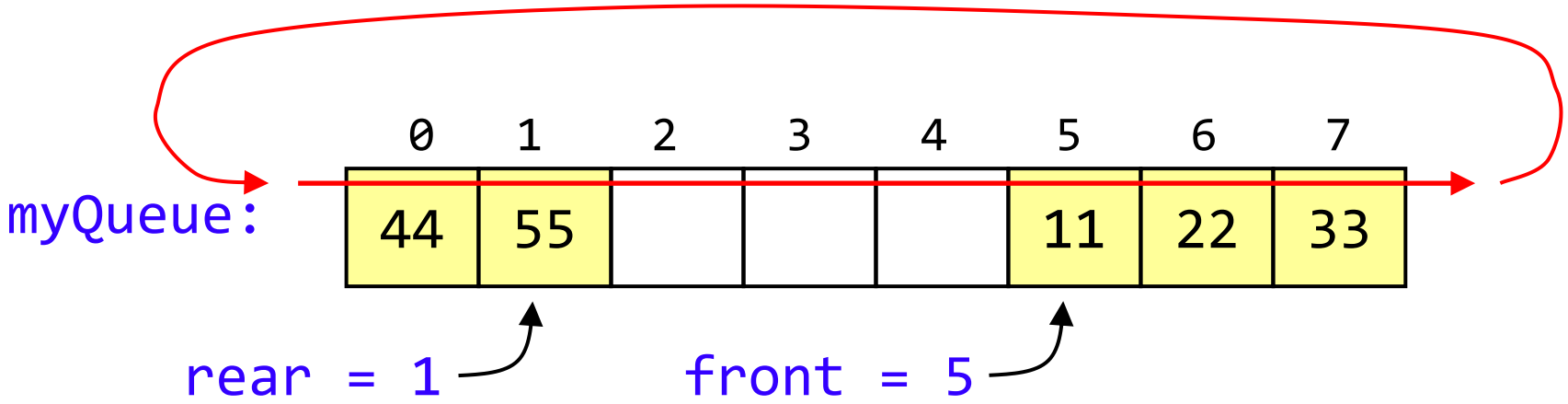
- Empty Queue



- **When, Front = Rear= -1 , it is an empty queue.**

Circular arrays

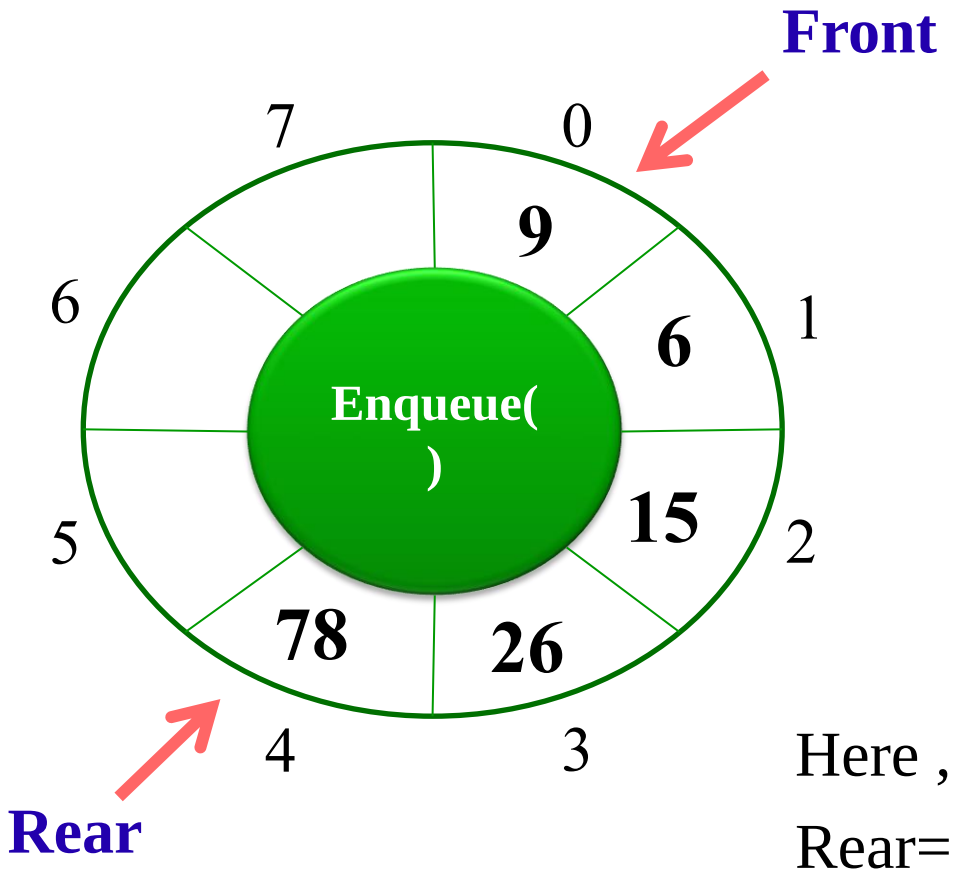
- We can treat the array holding the queue elements as circular (joined at the ends)



- Elements were added to this queue in the order 11, 22, 33, 44, 55, and will be removed in the same order
- Use: `front = (front + 1) % myQueue.length;`
  and: `rear = (rear + 1) % myQueue.length;`

Enqueue()

- Inserting 9 ,6 ,15 ,26 ,78



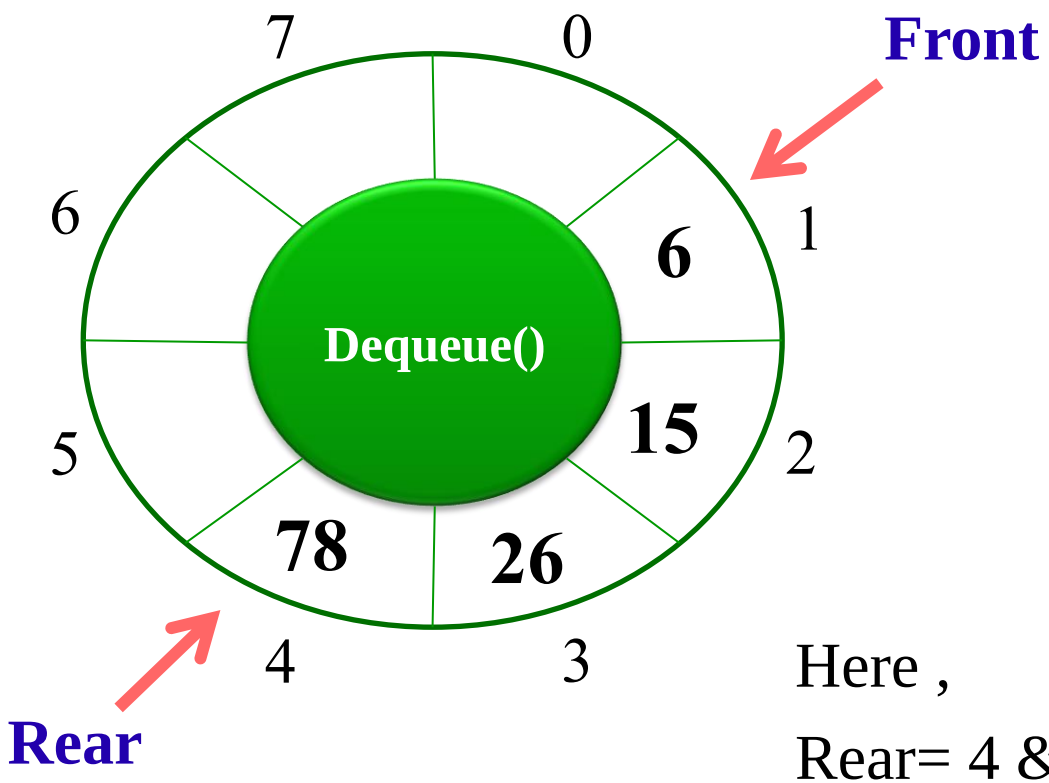
Here ,

Rear= 4 & Front = 0

- If (front = (rear+1)%n) then circular queue **overflow/Full.**
- Otherwise, rear = (rear+1)%n

Dequeue()

- Deleting 9



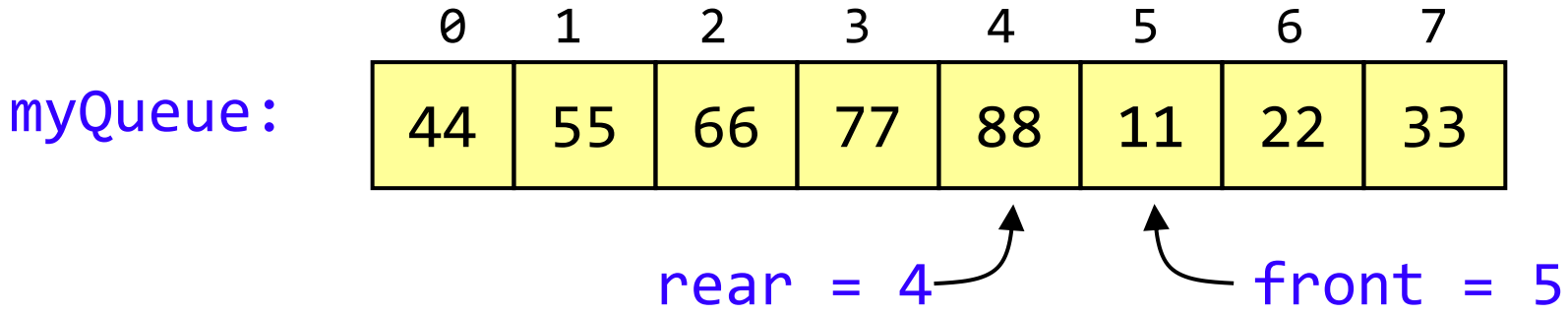
Here ,

Rear= 4 & Front = 0

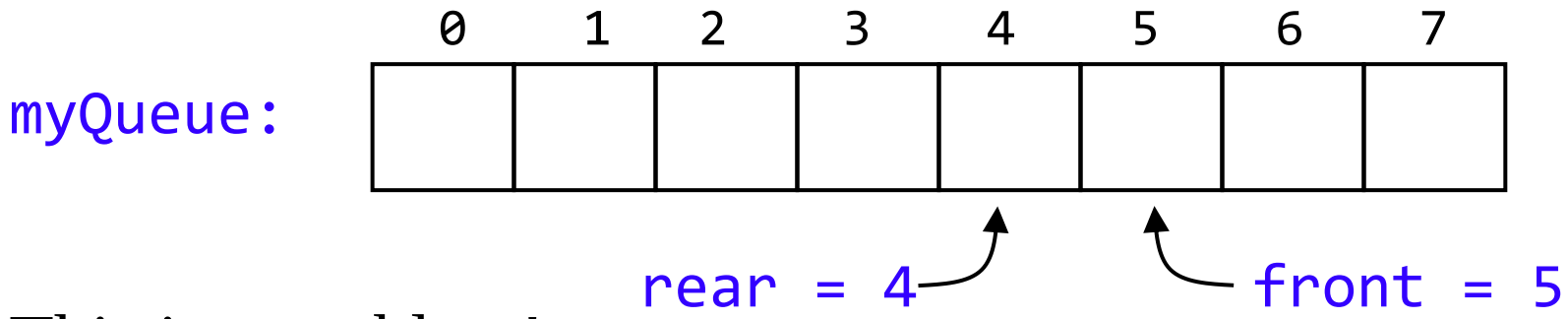
- If ((front = rear) && (rear = -1))then circular queue **underflow/empty**"
- Otherwise ,front = (front + 1) % n

Full and empty queues

- If the queue were to become completely full, it would look like this:



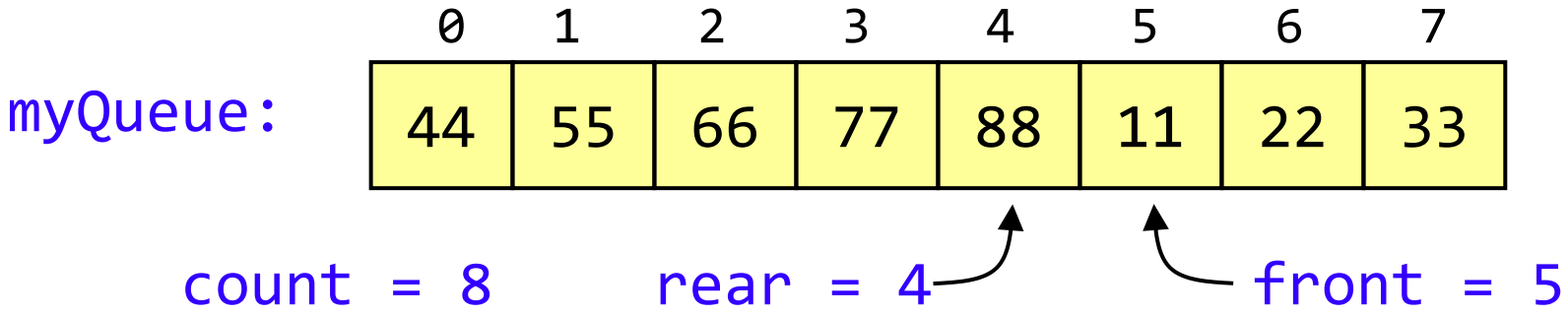
- If we were then to remove all eight elements, making the queue completely empty, it would look like this:



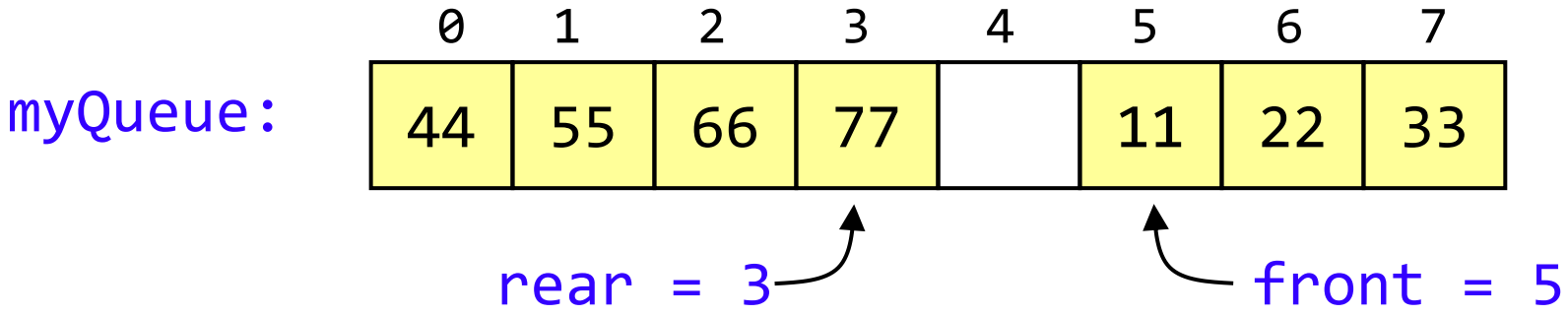
This is a problem!

Full and empty queues: solutions

- **Solution #1:** Keep an additional variable



- **Solution #2:** (Slightly more efficient) Keep a gap between elements: consider the queue full when it has `n-1` elements

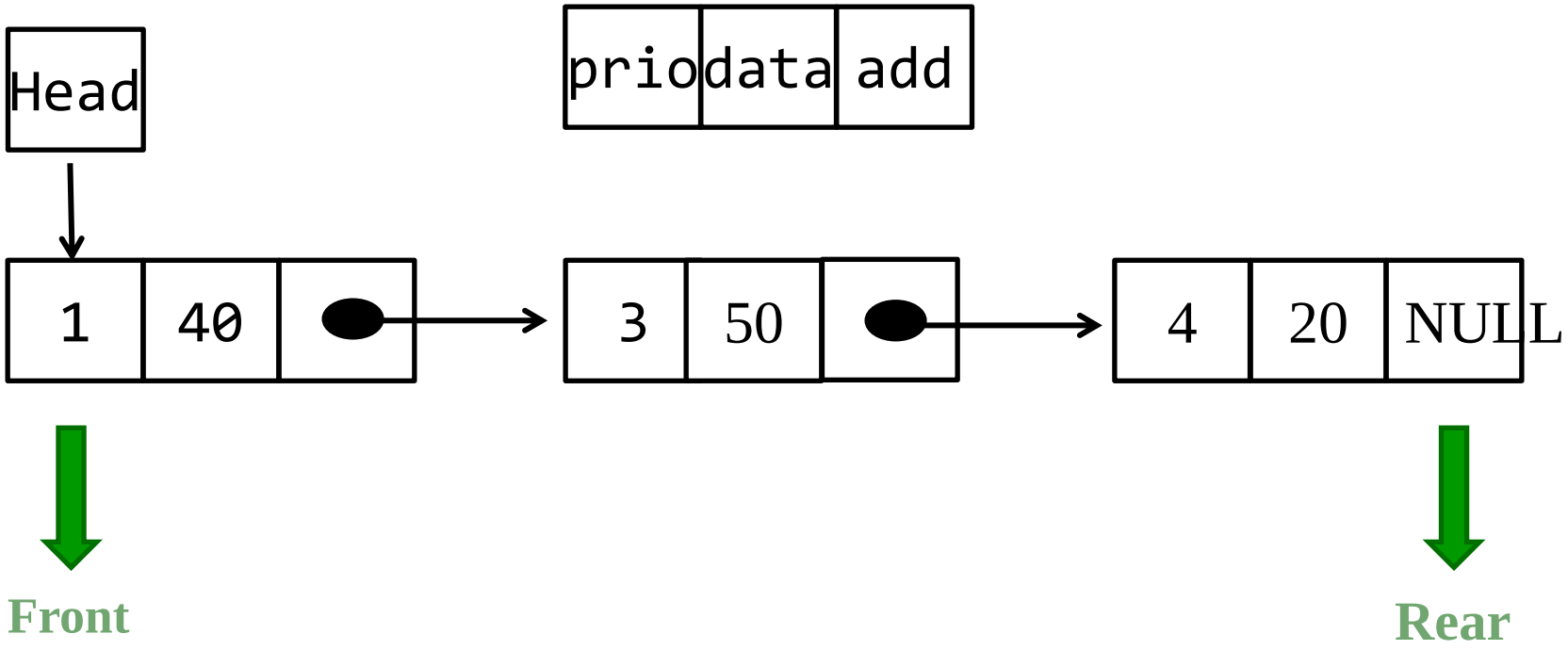


Priority Queue

- a **priority queue** is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it.

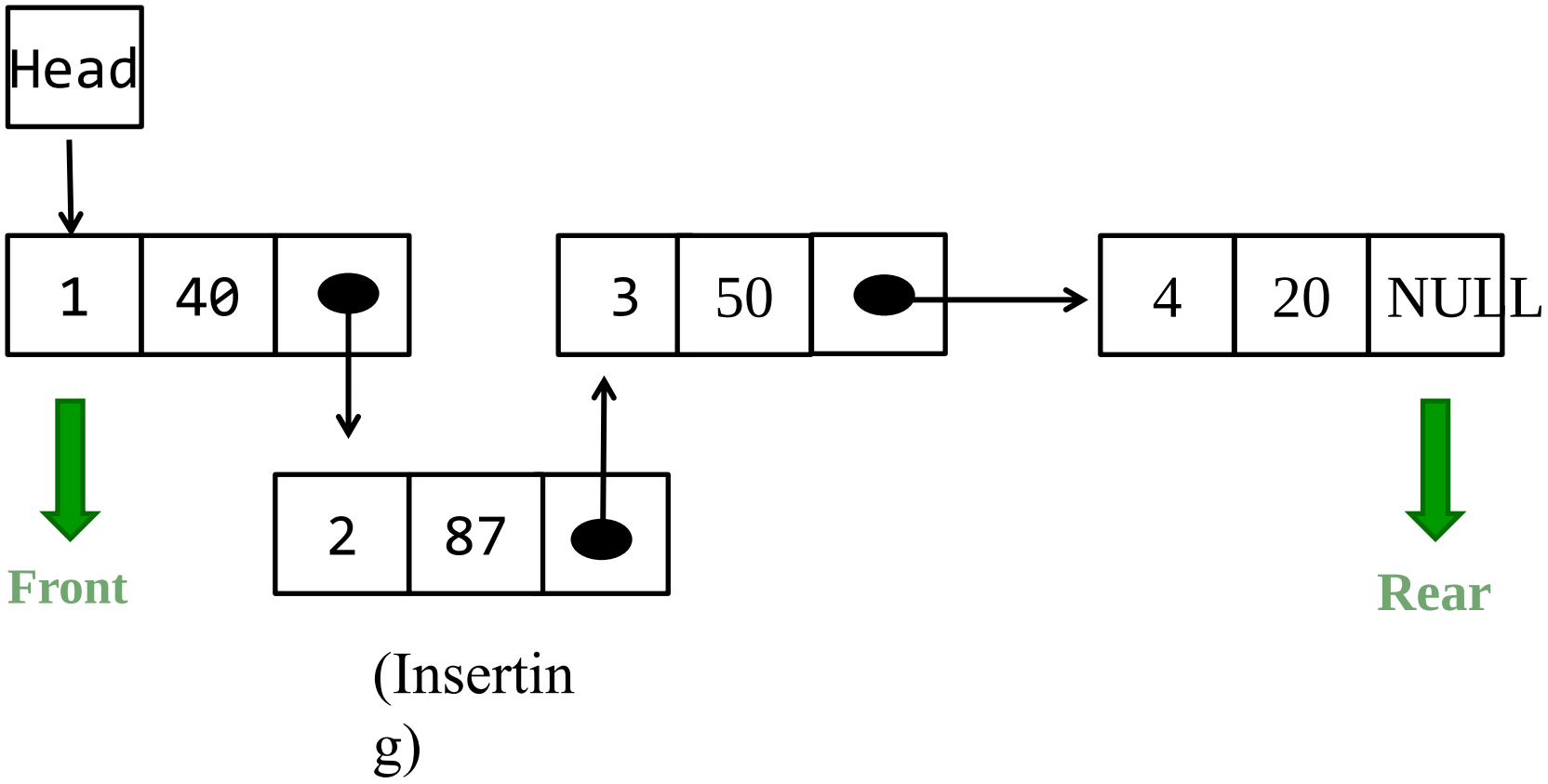
- In a **priority queue**, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

Insertion Operation



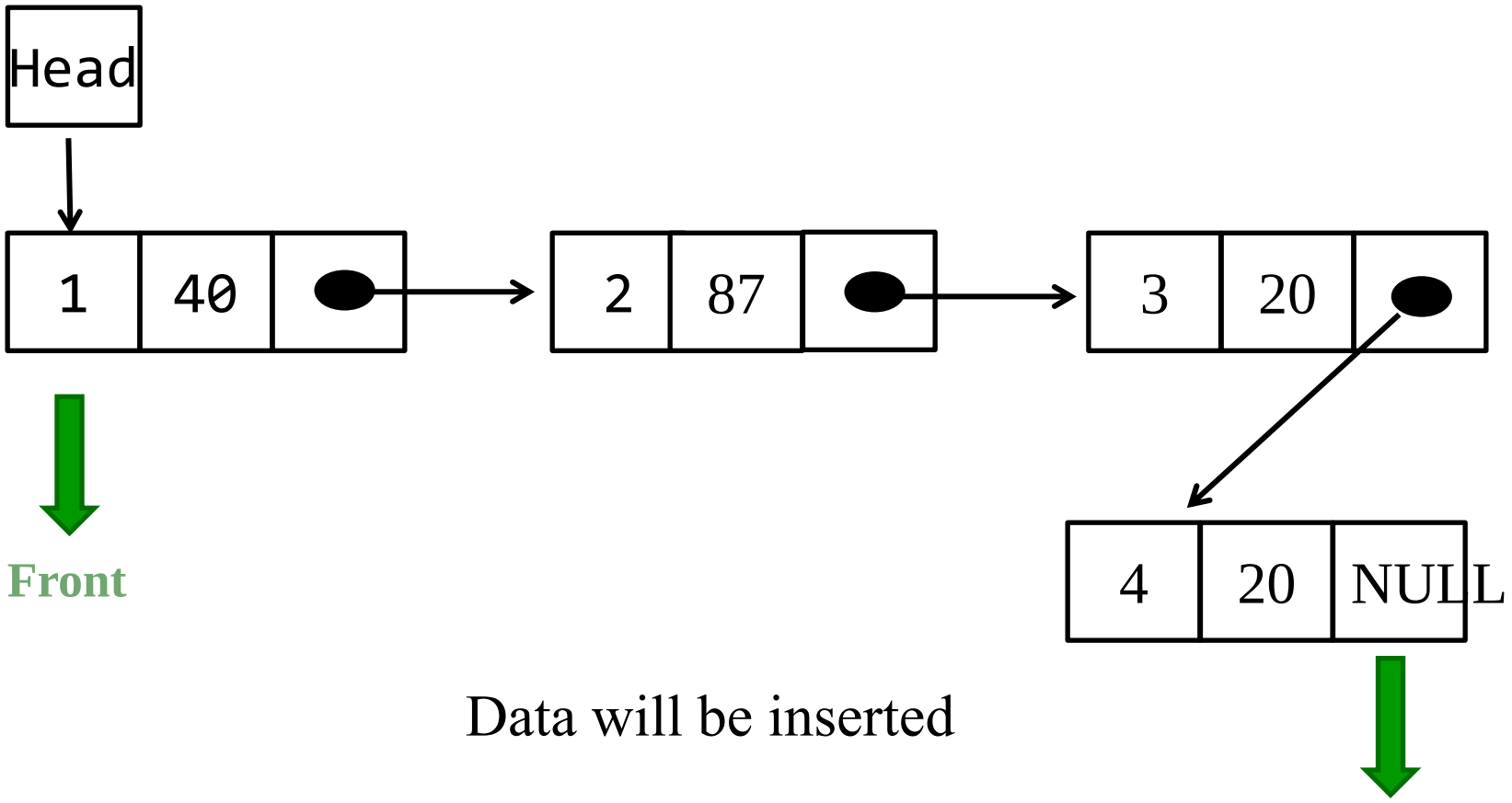
Insertion Operation

- Inserting 87 with priority 2



Insertion Operation

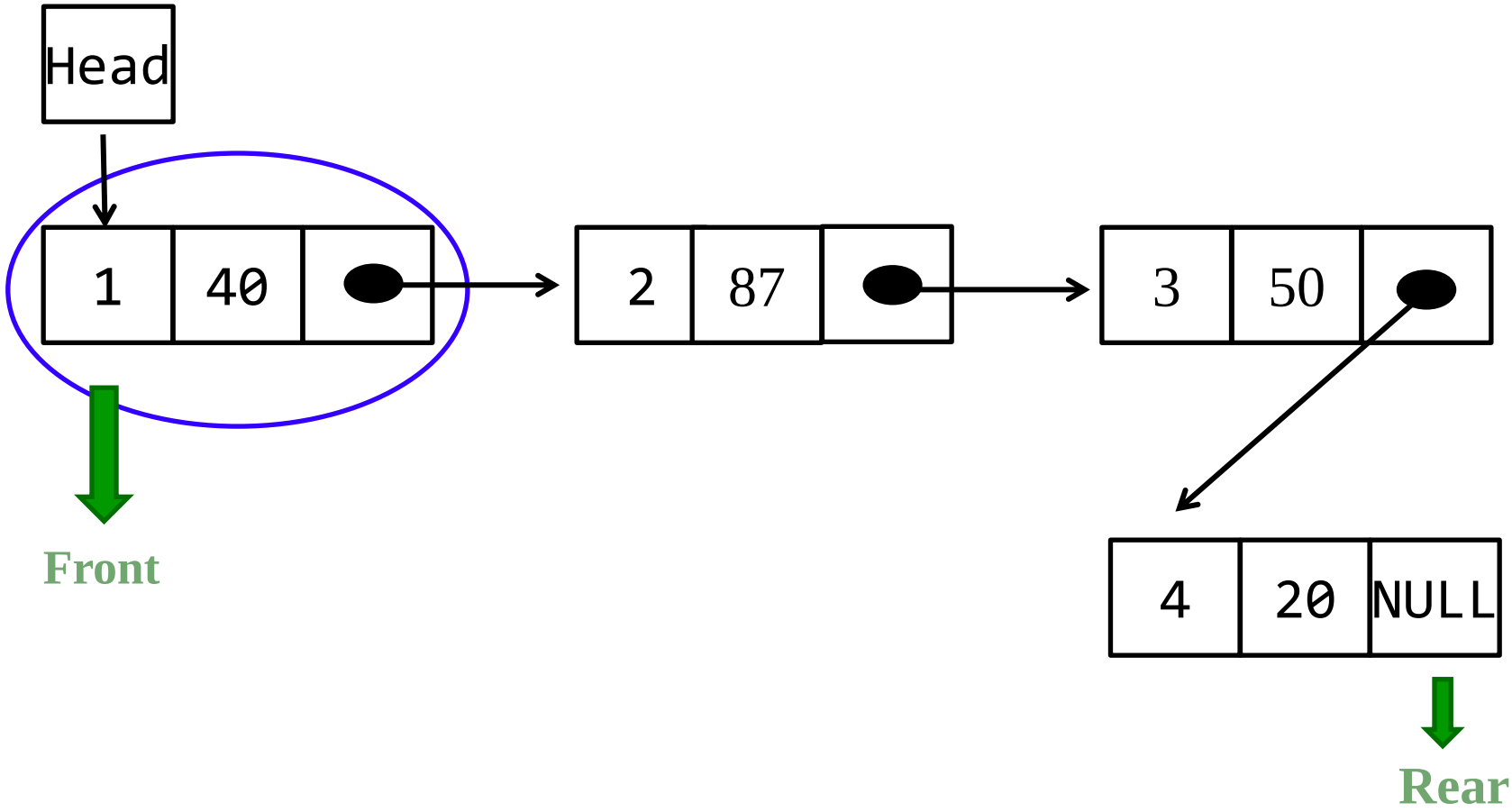
- Inserting 87 with priority 2



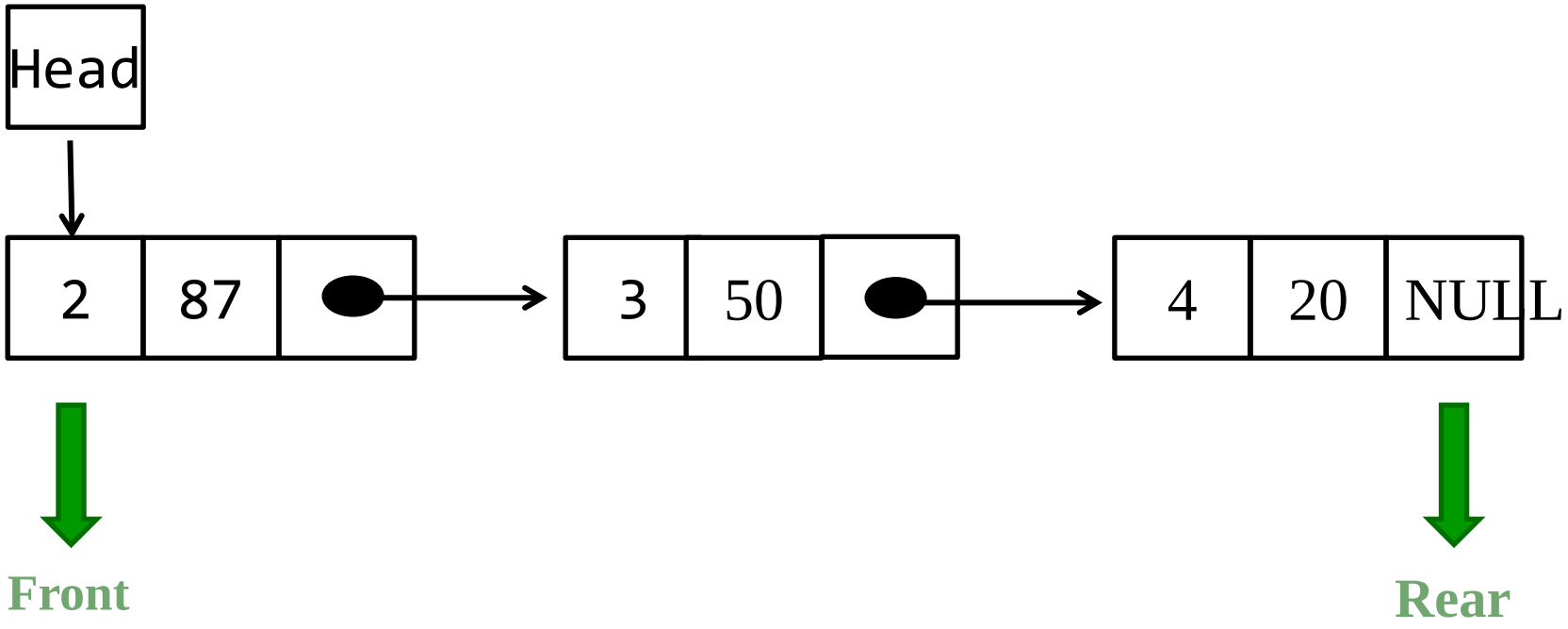
Data will be inserted

Deletion Operation

- Normally Deletion operation occurs at head pointer



Deletion Operation

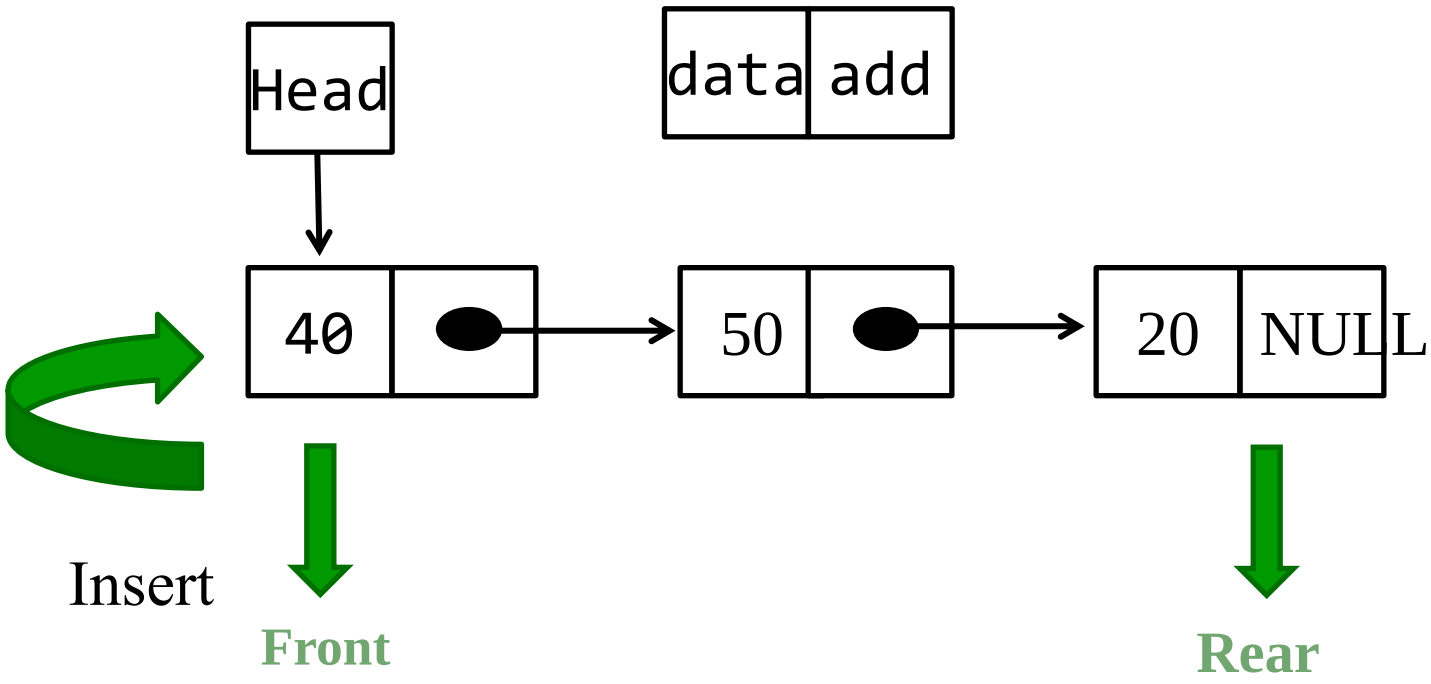


Deques

- A deque is a double-ended queue
- Insertions *and* deletions can occur at *either* end
- Implementation is similar to that for queues
- Deques are not heavily used
- You should know what a deque is, but we won't explore them much further

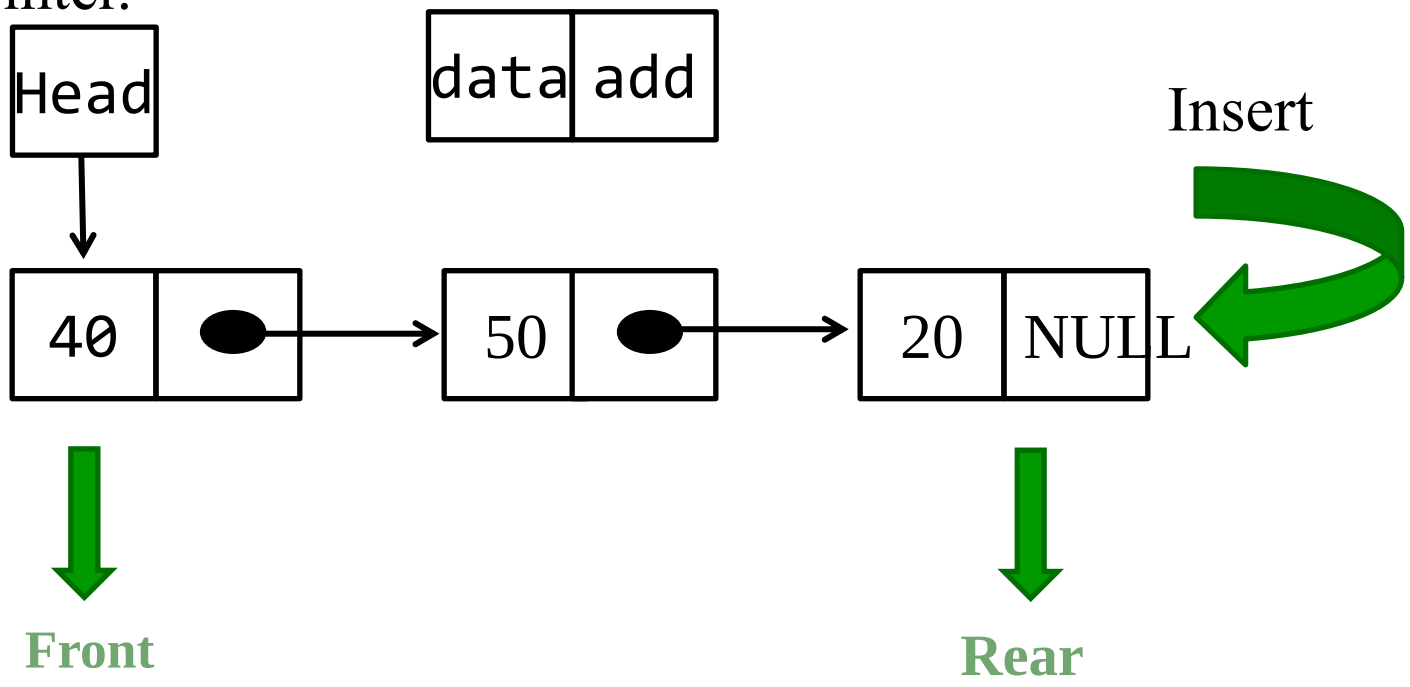
Insertion at Front

- Create a Node containing data and then point it to the node which is pointed to the head pointer.
- Then head pointer will point the created node.



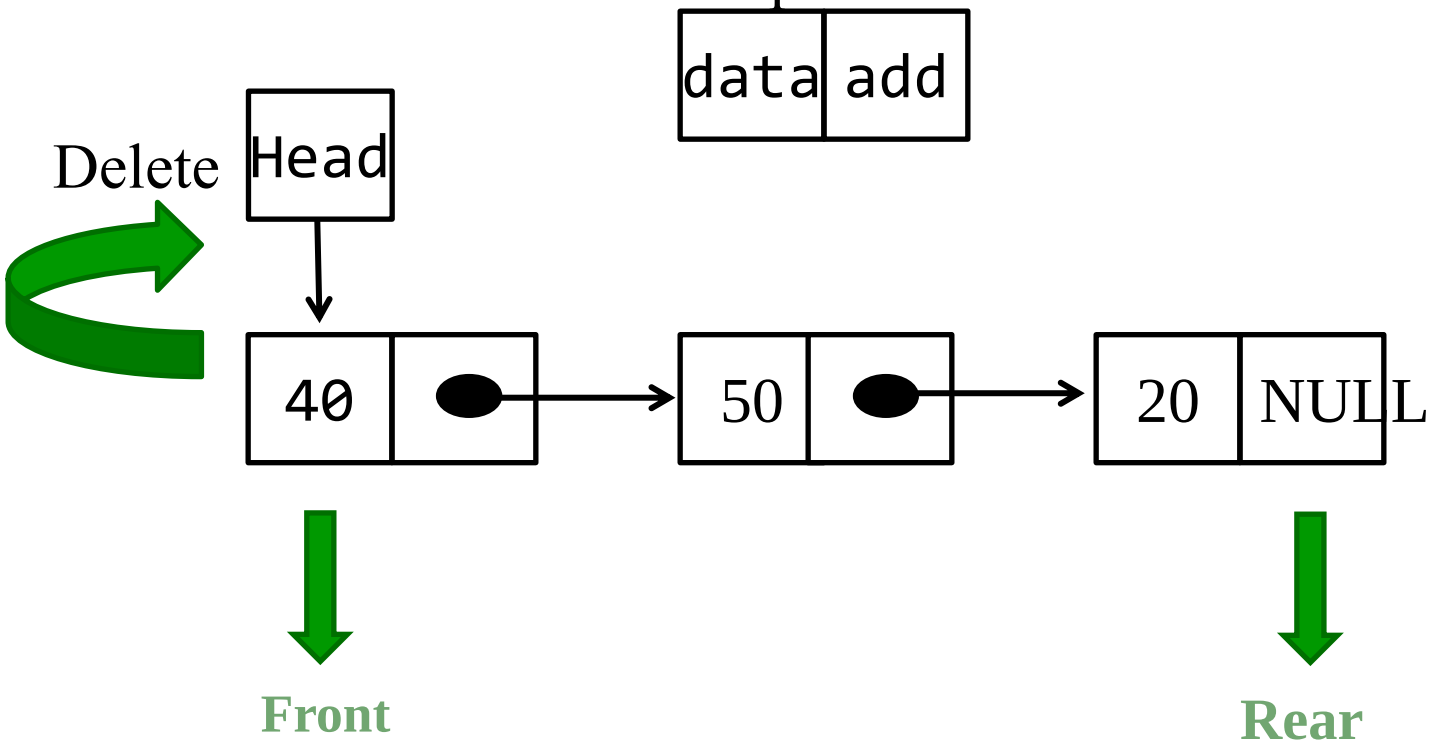
Insertion at Rear

- Create a Node containing data and then point it to the node which is pointed to the head pointer if head is NULL otherwise , check till the NULL pointer and then insert at that pointer.



Deletion Operation at Front

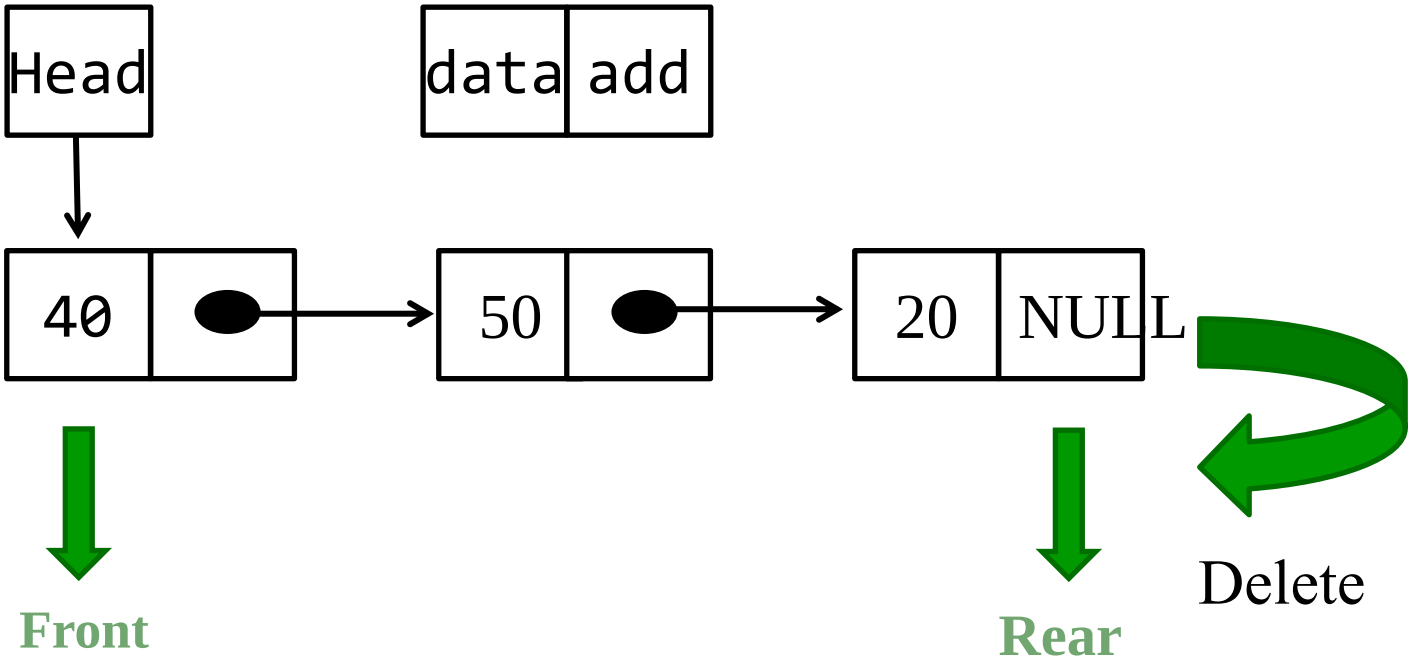
- Create a node temp then point it to head .Then head will point to head->next .
  Then we have to free temp.



Deletion Operation at Rear

- If head is NULL then queue is empty.
- If head->next is NULL then head=NULL.
- Otherwise, create a node temp.Then, we will check temp->next->next .If it is not

NULL then we will set temp= temp->next. Then if we find NULL then we free(temp->next).



QUEUING THEORY

Queuing Theory is a field of applied mathematics and computer science that is used to predict the performance of queues

Queuing Theory

- A single-server queue can provide service to only one customer at a time (hot food vendor on street corner).
- A multi-server queue can provide service to many customers at a time (bank, post-office).
- Multiqueues – multiple single-server queues (a grocery store).
- A customer is any person or thing needing the service.
- The service is any activity needed to accomplish the required result.

Queuing Theory

- The rate at which customers arrive in the queue for service is known as the arrival rate. It may be random or regular.
- Service time is the average time required to complete the processing of a customer request.

Queuing Theory

- In an ideal situation, customers would arrive at a rate that matches the service time.
- However, things are seldom ideal. Sometimes the server can be idle because there are no customers to be served. At other times there will be many customers to be served.
- If we can predict the patterns, we may be able to minimize idle servers and waiting customers.

Queuing Theory

- One of the main tasks in Queuing Theory is to predict such patterns.
- Specifically, it attempts to predict queue time (which is defined as the average length of time customers wait in the queue), the average size of the queue, and the maximum queue size.
- These predictions are based on the two factors: the arrival rate and the average service time (the average of the total service time between idle periods)

Queuing Theory

- Given queue time and service time, we know response time, - a measure of the average time from the point at which customers enter the queue until the moment they leave the server.

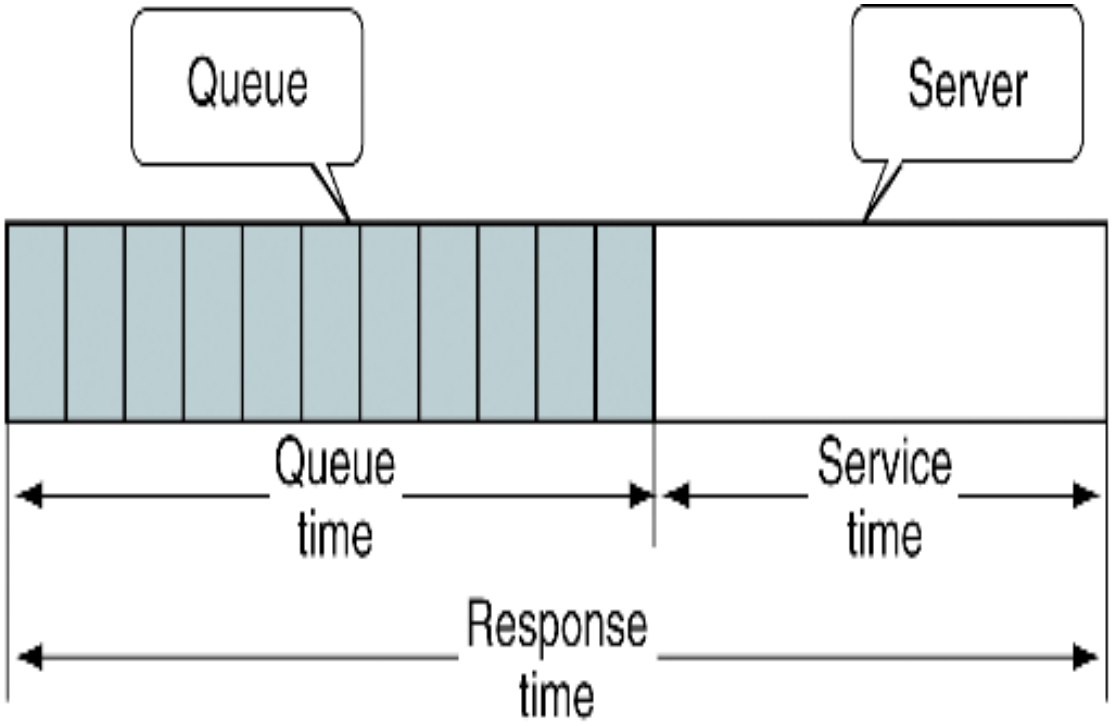


FIGURE 4-12 Queuing Theory Model

Queuing Theory

- The two factors that most affect the performance of queues are the **arrival rate** and the **service time**.

4-5 Queue Applications

We develop two queue applications. The first shows how to use a queue to categorize data. The second is a queue simulator, which is an excellent tool to simulate the performance and to increase our understanding of its operation.

- **Categorizing Data**
- **Queue Simulation**

Categorizing Data

- Categorizing Data is the rearrangement of data without destroying their basic sequence.
- As an example: suppose, it is necessary to rearrange a list of numbers grouping them, while maintaining original order in each group (multiple-queue application).

Categorizing Data

Initial list of numbers:

3 22 12 6 10 34 65 29 9 30 81 4 5 19 20 57 44 99

We need to categorize them into four different groups:

Group 1: less than 10

Group 2: between 10 and 19

Group 3: between 20 and 29

Group4: 30 and greater

| 3 6 9 4 5 | 12 10 19 | 22 29 20 | 34 65 30 81 57 44 99

Categorizing Data

- The solution: we build a queue for each of the four categories. We than store the numbers in the appropriate queue as we read them.
- After all the data have been processed, we print each queue to demonstrate that we categorized data correctly.

ALGORITHM 4-9 Category Queues

```
Algorithm categorize
   Group a list of numbers into four groups using four queues.
      Written by:
      Date:
 1 createQueue (q0to9)
 2 createQueue (q10to19)
 3 createQueue (q20to29)
 4 createQueue (qOver29)
 5 fillQueues  (q0to9, q10to19, q20to29, qOver29)
 6 printQueues (q0to9, q10to19, q20to29, qOver29)
end categorize
```

ALGORITHM 4-10 Fill Category Queues

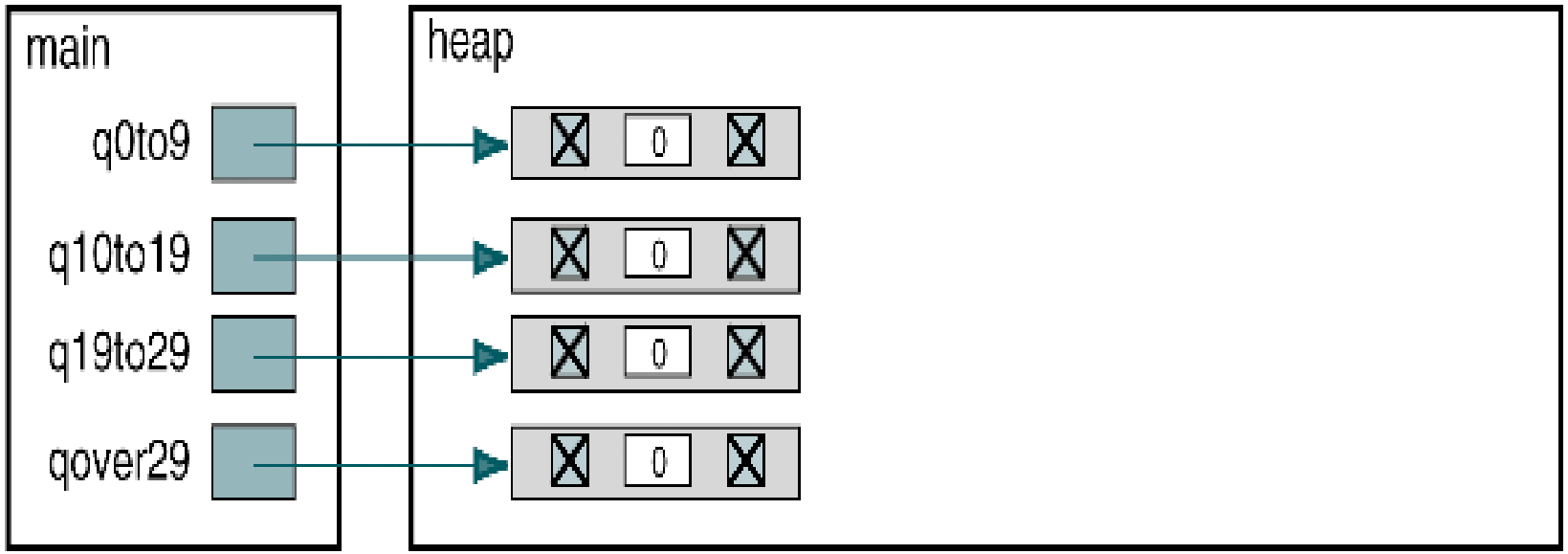
```
Algorithm fillQueues (q0to9, q10to19, q20to29, qOver29)
This algorithm reads data from the keyboard and places them in
one of four queues.
  Pre  all four queues have been created
  Post queues filled with data
1 loop (not end of data)
```

continued

ALGORITHM 4-10 Fill Category Queues *(continued)*

```
   1  read (number)
   2  if (number < 10)
      1  enqueue (q0to9, number)
   3  elseif (number < 20)
      1  enqueue (q10to19, number)
   4  elseif (number < 30)
      1  enqueue (q20to29, number)
   5  else
      1  enqueue (qOver29, number)
   6  end if
 2 end loop
end fillQueues
```

FIGURE 4-13 Structures for Categorizing Data



(a) Before calling fillQueues

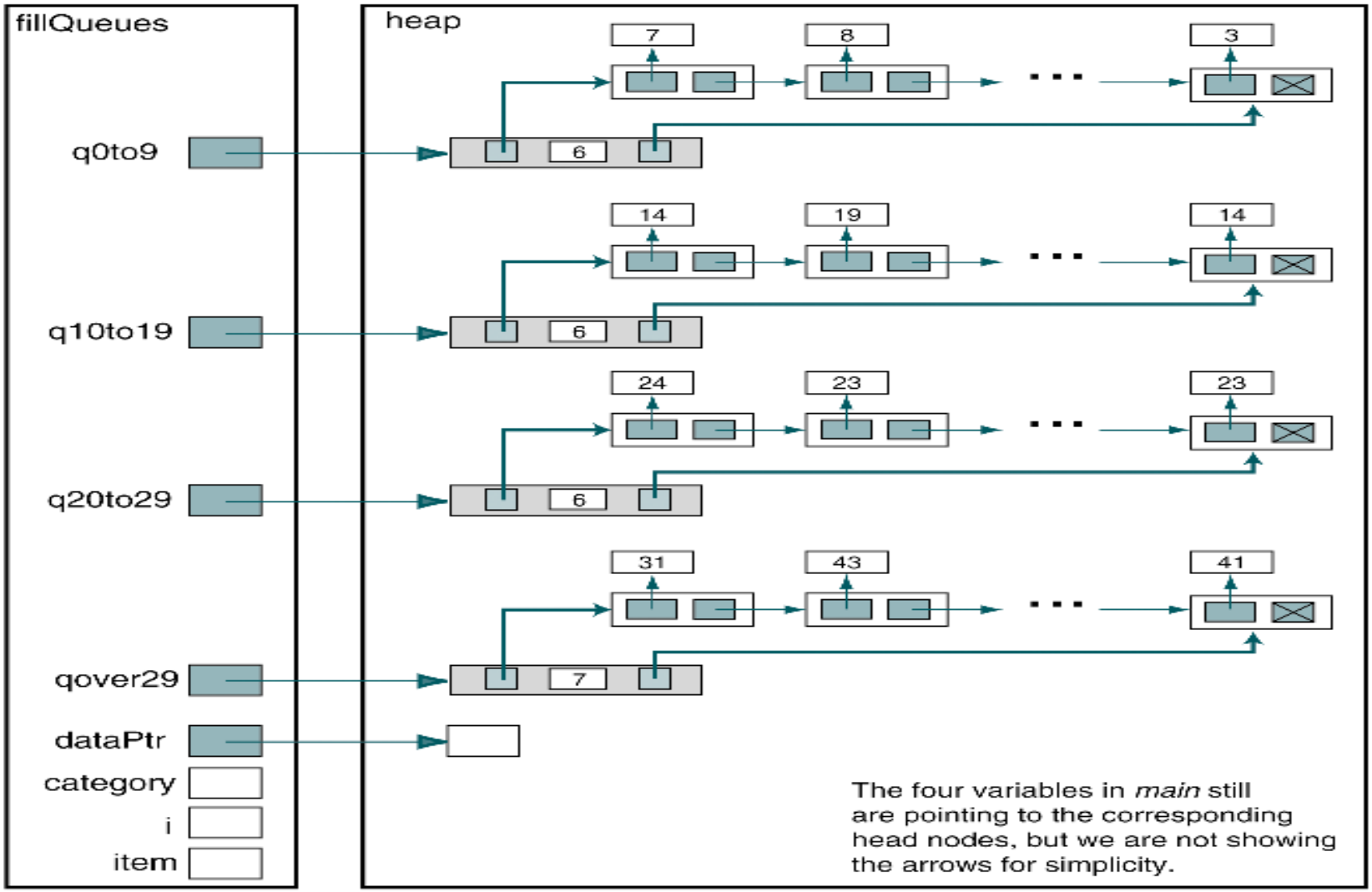
PROGRAM 4-14 Print One Queue *(continued)*

```
Results:
Welcome to a demonstration of categorizing
data. We generate 25 random numbers and then
group them into categories using queues.

Categorizing data:
 24  7 31 23 26 14 19  8  9  6 43
 16 22  0 39 46 22 38 41 23 19 18
 14  3 41
End of data categorization

Data   0.. 9:  7   8   9   6   0   3
Data  10..19: 14  19  16  19  18  14
Data  20..29: 24  23  26  22  22  23
Data over 29: 31  43  39  46  38  41  41
```

FIGURE 4-13 Structures for Categorizing Data **(Continued)**



(b) After calling fillQueues