Types of Queues

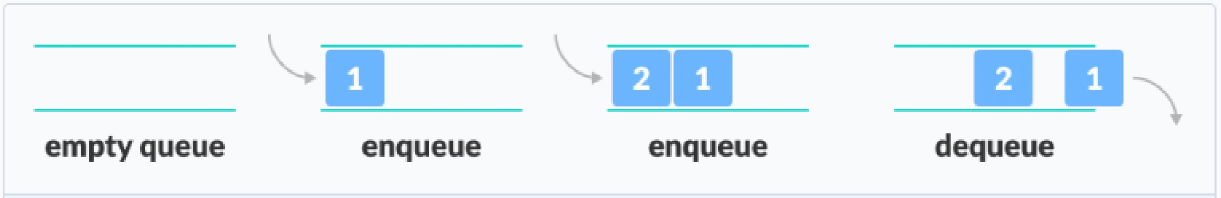
- Queue can be of four types:
  - Simple Queue
  - Circular Queue
  - Priority Queue
  - De-queue (Double Ended Queue)

A queue is a container of elements that are inserted and removed according to the ***first-in first-out*** (***FIFO***) principle. Elements can be inserted in a queue at any time, but only the element that has been in the queue the longest can be removed at any time. We usually say that elements enter the queue at the ***rear*** and are removed from the ***front***. The metaphor for this terminology is a line of people waiting to get on an amusement park ride. People enter at the rear of the line and get on the ride from the front of the line.

Formally, the queue abstract data type defines a container that keeps elements in a sequence, where element access and deletion are restricted to the first element in the sequence, which is called the ***front*** of the queue, and element insertion is restricted to the end of the sequence, which is called the ***rear*** of the queue. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in first-out (FIFO) principle.

The ***queue*** abstract data type (ADT) supports the following operations:

enqueue(e):	Insert element e at the rear of the queue.
dequeue():	Remove element at the front of the queue; an error occurs if the queue is empty.



FIFO Representation of Queue

front(): Return, but do not remove, a reference to the front element in the queue; an error occurs if the queue is empty.

The queue ADT also includes the following supporting member functions:

size(): Return the number of elements in the queue.

empty(): Return true if the queue is empty and false otherwise.

We illustrate the operations in the queue ADT in the following example.

Example 5.4: *The following table shows a series of queue operations and their effects on an initially empty queue, Q, of integers.*

Operation	Output	front $\leftarrow Q \leftarrow$ rear
enqueue(5)	–	(5)
enqueue(3)	–	(5,3)
front()	5	(5,3)
size()	2	(5,3)
dequeue()	–	(3)
enqueue(7)	–	(3,7)
dequeue()	–	(7)
front()	7	(7)
dequeue()	–	()
dequeue()	"error"	()
empty()	true	()

Double-Ended Queues

Consider now a queue-like data structure that supports insertion and deletion at both the front and the rear of the queue. Such an extension of a queue is called a ***double-ended queue***, or ***deque***, which is usually pronounced "deck" to avoid confusion with the dequeue function of the regular queue ADT, which is pronounced like the abbreviation "D.Q." An easy way to remember the "deck" pronunciation is to observe that a deque is like a deck of cards in the hands of a crooked card dealer—it is possible to deal off both the top and the bottom.

5.3.1 The Deque Abstract Data Type

The functions of the deque ADT are as follows, where D denotes the deque:

insertFront(e):	Insert a new element e at the beginning of the deque.
insertBack(e):	Insert a new element e at the end of the deque.
eraseFront():	Remove the first element of the deque; an error occurs if the deque is empty.
eraseBack():	Remove the last element of the deque; an error occurs if the deque is empty.

Additionally, the deque includes the following support functions:

front():	Return the first element of the deque; an error occurs if the deque is empty.
back():	Return the last element of the deque; an error occurs if the deque is empty.
size():	Return the number of elements of the deque.
empty():	Return true if the deque is empty and false otherwise.

Example 5.5: *The following example shows a series of operations and their effects on an initially empty deque, D, of integers.*

Operation	*Output*	D
insertFront(3)	–	(3)
insertFront(5)	–	(5,3)
front()	5	(5,3)
eraseFront()	–	(3)
insertBack(7)	–	(3,7)
back()	7	(3,7)
eraseFront()	–	(7)
eraseBack()	–	()

Stack Method	***Deque Implementation***
size()	size()
empty()	empty()
top()	front()
push(o)	insertFront(o)
pop()	eraseFront()

Queue Method	***Deque Implementation***
size()	size()
empty()	empty()
front()	front()
enqueue(e)	insertBack(e)
dequeue()	eraseFront()

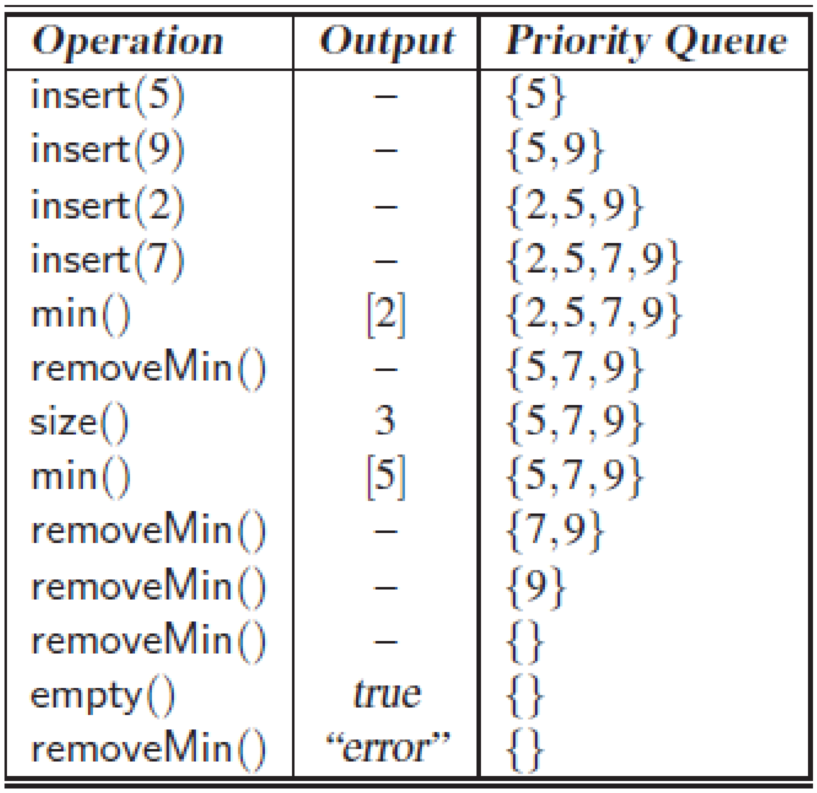
A data structure or software that contains ("wraps around") other data or software, so that the contained elements can exist in the newer system. The term is often used with component software, where a wrapper is placed around a legacy routine to make it behave like an object.

The Priority Queue ADT

Having described the priority queue abstract data type at an intuitive level, we now describe it in more detail. As an ADT, a priority queue P supports the following functions:

size():	Return the number of elements in P.
empty():	Return true if P is empty and false otherwise.
insert(e):	Insert a new element e into P.
min():	Return a reference to an element of P with the smallest associated key value (but do not remove it); an error condition occurs if the priority queue is empty.
removeMin():	Remove from P the element referenced by min(); an error condition occurs if the priority queue is empty.

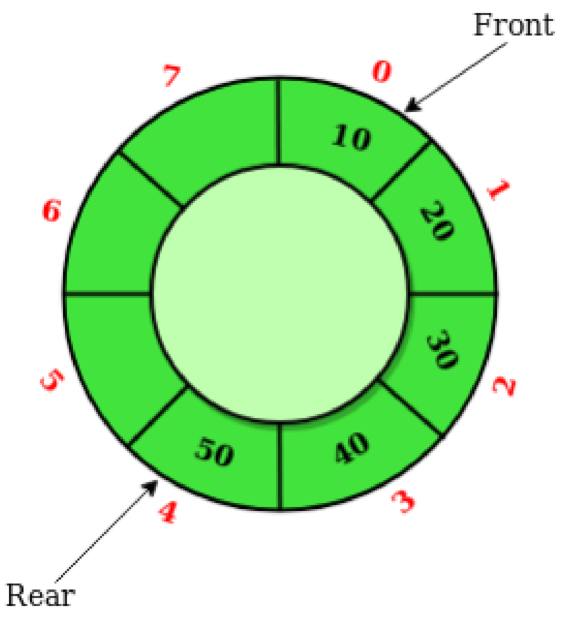
- The priority queue is a special type of data structure in which items can be inserted or deleted based on the priority.
- If the elements in the queue are of same priority, then the element, which is inserted first into the queue, is processed.



Operation	Output	Priority Queue
insert(5)	–	{5}
insert(9)	–	{5,9}
insert(2)	–	{2,5,9}
insert(7)	–	{2,5,7,9}
min()	[2]	{2,5,7,9}
removeMin()	–	{5,7,9}
size()	3	{5,7,9}
min()	[5]	{5,7,9}
removeMin()	–	{7,9}
removeMin()	–	{9}
removeMin()	–	{}
empty()	*true*	{}
removeMin()	"error"	{}

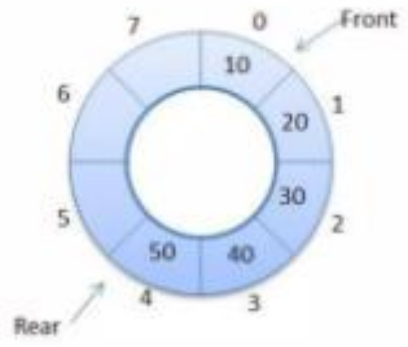
> *A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle.*

The operations are performed based on FIFO (First In First Out) principle. It is also called **'Ring Buffer'**.

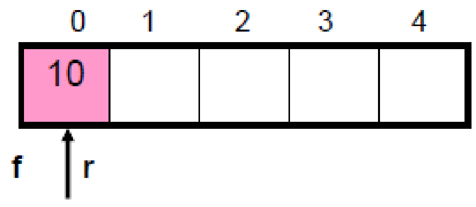


Circular Queue

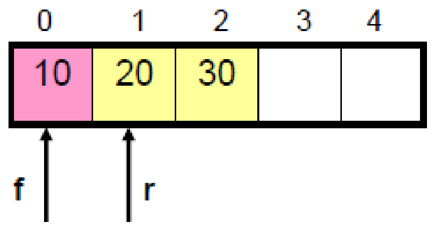
- Circular Queue: A circular queue is a queue in which all nodes are treated as circular such that the last node follows the first node.



- In *circular queue*, the elements of a given queue can be stored efficiently in an array so as to "**wrap around**" so that end of the queue is followed by the front of the queue.

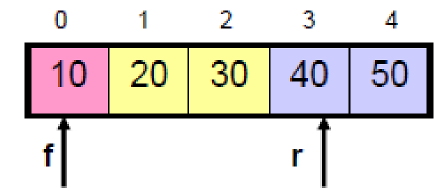


Initial queue

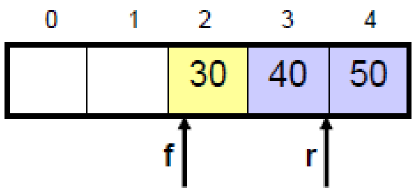


After inserting 20 & 30

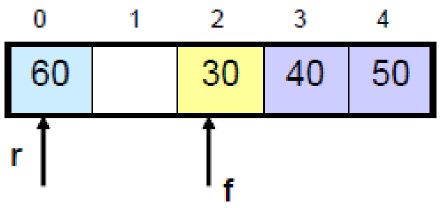
ABDULLAH: 03343127215



After inserting 40 and 50



After deleting 10 and 20



After inserting 60

A priority queue is a **special type of queue** in which each element is associated with a **priority value.** And, elements are served on the basis of their priority. That is, higher priority elements are served first.

However, if elements with the same priority occur, they are served according to their order in the queue.

Difference between Priority Queue and Normal Queue?

There is no priority attached to elements in a queue, the rule of first-in-first-out(FIFO) is implemented whereas, in a priority queue, the elements have a priority. The elements with higher priority are served first.