**Q1. a) What is a stack? Explain with examples.**

**Ans:**Stack is a linear data structure in which items are added or removed only at one end, called top of the stack.Thus, there is no way to add or delete elements anywhere else in the stack. A stack is based on Last-In-First-Out (LIFO) principle that means the data item that is inserted last into the stack is the first one to be removed from the stack.

We can relate a stack to certain real-life objects and situations, as shown in Figs.

Fig. 1. Stack of books Fig.2. Stack of plates

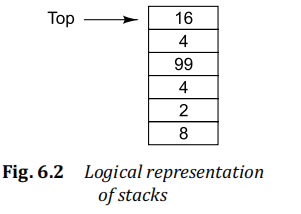
In Fig.1, one can add a new book to an existing stack of books only at its top and nowhere else. Similarly, a plate cannot be added at the middle of the plates stack; one has to first remove all the plates above the insertion point for the new plate to be added there. Another apt example of a stack is a set of bangles worn by Indian women on their arms. A bangle can only be worn from one side of the hand and to remove a bangle from the middle one has to first remove all the prior bangles. The concept of stack in data structures follows the same analogy as the stack of books or the stack of

plates. We may use a stack in data structures to store built-in or user-defined type elements depending upon our programming requirements. Irrespective of the type of elements stored, each stack implementation follows similar representation in memory.

**Stack Representation in Memory**

Just like their real world counterparts, stacks appear as a group of elements stored at contiguous locations in memory. Each successive insert or delete operation adds or removes an item from the group. The top location of the stack or the point of addition or deletion is maintained by a pointer called top. Figure 6.2 shows the logical representation of stacks in memory.

There are six elements in the stack with element 16 being at the top of the



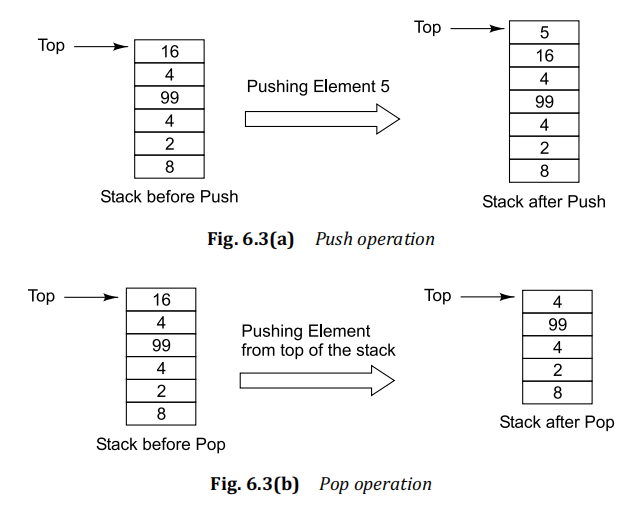
**STACK OPERATIONS**

There are two key operations associated with the stack data structure: push and pop. Adding an element

to the stack is referred as push operation while reading or deleting an element from the stack is referred

as pop operation.

Figures (a) and (b) depict the push and pop operations on a stack.



A stack is a linear list in which elements are added and removed only from one end called top

of the stack.

©

Stacks are based on Last-In-First-Out or LIFO principle that means, the element added last into

the list is the first one to be removed.

©

Inserting an element into a stack is referred as push operation while removing an element from

the stack is referred as pop operation.

©

Stacks can be implemented through arrays or linked lists.

©

The array implementation of stacks reserves a fixed amount of memory space in the form of an

array for storing stack elements.

©

The linked implementation of stacks uses dynamic memory management techniques for allocating

the memory space for storing a new stack element at run time.

©

Since linked implementation of stacks is based on dynamic memory allocation it is more efficient

as compared to array-based implementation.

©

The various application areas of stacks are expression evaluation, program control, recursion

control, etc.

**Q.2. Explain the importance of circular queue.**

**Ans:** Queue is a linear data structure in which items are inserted at one end called ‘Rear’ and deleted from

the other end called ‘Front’. Queues are based on the First-In-First-Out (FIFO) principle that means the

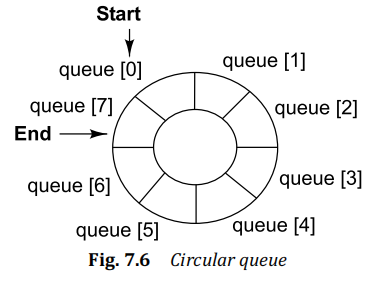
data item that is inserted first in the queue is also the first one to be removed from the queue.

A circular queue is a queue whose start and end locations are logically connected with each

other. That means, the start location comes after the end location. If we continue to add elements in a circular queue till its end location, then after the end location has been filled, the next element will be added at the beginning of the

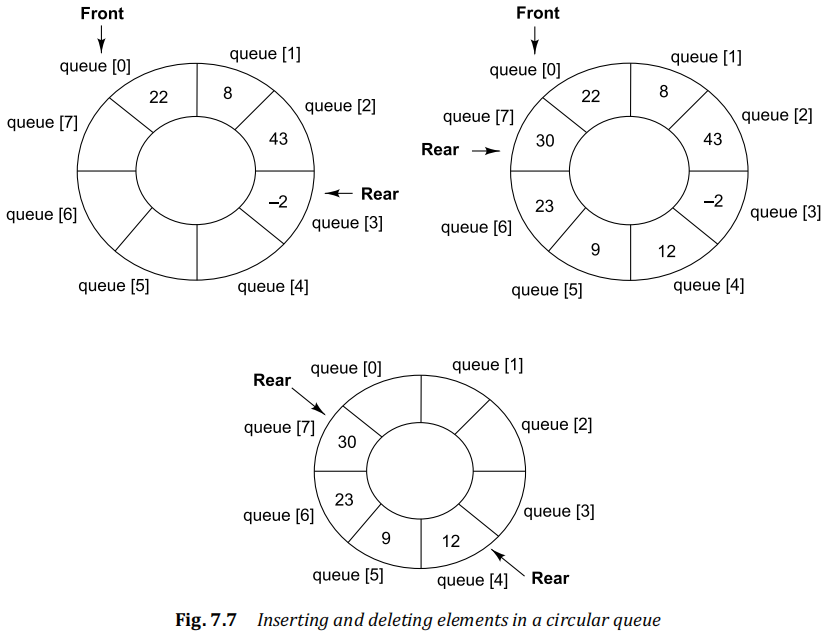
queue. Circular queues remove one of the main disadvantages of array implemented queues in which a lot of memory space is wasted due to inefficient utilization.

Figure 7.6 shows the logical representation of a circular queue.



the start location of the queue comes after its end location. Thus, if the queue is filled till its capacity, i.e., the end location, then the start location will be checked for space, and if it is empty, the new element will be added there. Figure 7.7 shows

the different states of a circular queue during insert and delete operations.



**Insert Operation:** The insert operation for array implemented circular queues involves the following tasks:

1. Checking whether the queue is already full.

2. Updating the rear pointer.

(a) If the queue is empty, set front and rear

(b) If rear is pointing at the last location of the queue, set rear

queue.

(c) If none of the above situations exist, simply increment the rear pointer by 1.

3. Inserting the new element at the rear location.

**Delete Operation:** The delete operation for array implemented circular queues involves the following tasks:

1. Checking whether the queue is already empty.

2. Retrieving the element at the front of the queue.

3. Updating the front pointer.

(a) If the queue has only one element left, set front and rear to point to NULL.

(b) If front is pointing at the last location of the queue, set front

the queue.

(c) If none of the above situations exist, simply increment the front pointer by 1.

4. Returning the element retrieved from the front location.

**Q.3. Define double ended queue with example.**

**Ans:** Queue is a linear data structure in which items are inserted at one end called ‘Rear’ and deleted from

the other end called ‘Front’. Queues are based on the First-In-First-Out (FIFO) principle that means the

data item that is inserted first in the queue is also the first one to be removed from the queue.

A double-ended queue is a special type of queue that allows insertion and deletion of elements at both ends,

i.e., front and rear. In simple words, a double-ended queue can be referred as a linear list of elements in which insertion

and deletion of elements takes place at its two ends but not in the middle.This is the reason why it is termed as double-ended queue or deque.

Based on the type of restrictions imposed on insertion and deletion of elements, a double-ended queue is categorized into two types:

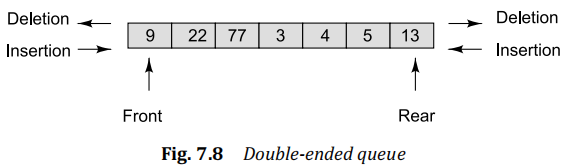
**1. Input-restricted deque** It allows deletion from both the ends but restricts the insertion at only

one end.

**2. Output-restricted deque** It allows insertion at both the ends but restricts the deletion at only

one end.

Figure 7.8 shows the logical representation of a deque.



As shown in Fig. 7.8, insertion and deletion of elements is possible at both front and rear ends of the

queue. As a result, the following four operations are possible for a double-ended queue:

1. i\_front Insertion at front end of the queue.

2. d\_front Deletion from front end of the queue.

3. i\_rear Insertion at rear end of the queue.

4. d\_rear Deletion from rear end of the queue.

1. **4. The contents of a queue Q are as follows:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Queue (Q) | 4 | 5 | -9 | 66 |  |  |  |  |
| Index | 0  ↑ | 1 | 2 | 3  ↑ | 4 | 5 | 6 | 7 |

The queue can store a maximum of eight elements and the front (F) and rear (R) pointers currently

point at index 0 and 3 respectively.

Show the queue contents and indicate the position of the front and rear pointers after each of the

following queue operations:

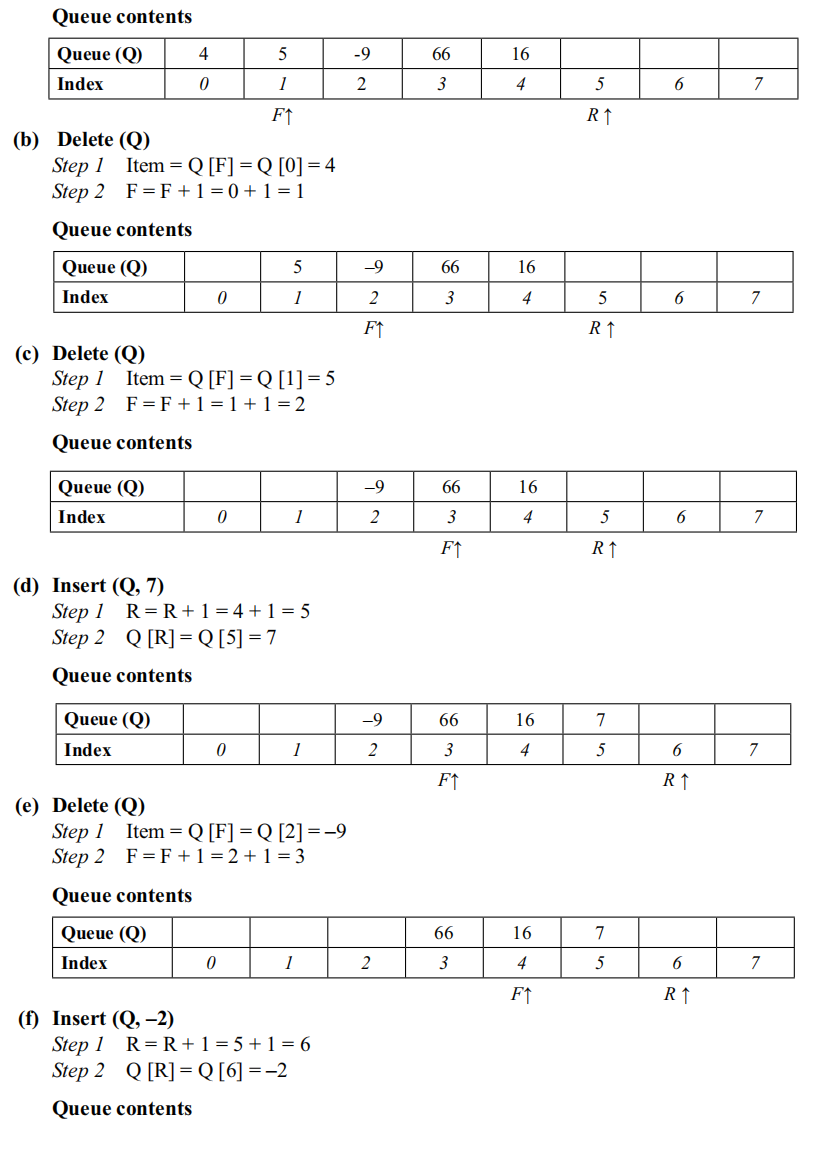
(a) Insert (Q, 16), (b) Delete (Q), (c) Delete (Q), (d) Insert (Q, 7), (e) Delete (Q), (f) Insert (Q, –2)

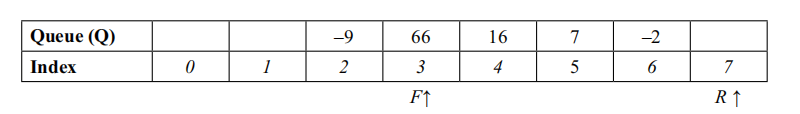
Solution

(a) Insert (Q, 16)

Step 1 R = R + 1 = 3 + 1 = 4

Step 2 Q [R] = Q [4] = 16





**Q.5. Discuss a priority queue?**

**Ans:**Queue is a linear data structure in which items are inserted at one end called ‘Rear’ and deleted from the other end called ‘Front’. Queues are based on the First-In-First-Out (FIFO) principle that means the data item that is inserted first in the queue is also the first one to be removed from the queue.

Priority queue is a type of queue in which each element is assigned certain priority such that the order of deletion of elements is decided by their associated priorities. The order of processing or deletion of elements in a priority queue is decided by the following rules:

1. An element with highest priority is deleted before all other elements of lower priority.

2. If two elements have the same priority then they are deleted as per the order in which they were added into the queue

(i.e., First-In-First-Out).

The implementation of priority queues may follow different approaches. For instance, elements may be added arbitrarily into the queue and deleted as per their priority values or, the elements may be sorted as per their priorities at the time of their insertion itself, and deleted in a sequential fashion.

The structure of a priority queue needs to be defined in such a manner that each queue node is able

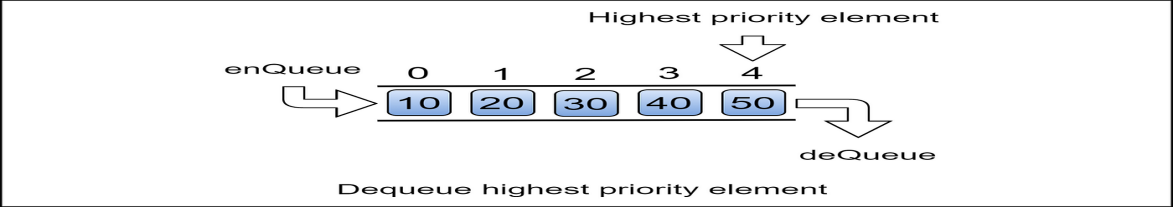
to store both its value as well as its priority information

## ****Advantages of Priority Queue:****

* It helps to access the elements in a faster way. This is because elements in a priority queue are ordered by priority, one can easily retrieve the highest priority element without having to search through the entire queue.
* The ordering of elements in a Priority Queue is done dynamically. Elements in a priority queue can have their priority values updated, which allows the queue to dynamically reorder itself as priorities change.
* Efficient algorithms can be implemented. Priority queues are used in many algorithms to improve their efficiency, such as Dijkstra’s algorithm for finding the shortest path in a graph and the A\* search algorithm for pathfinding.
* Included in real-time systems. This is because priority queues allow you to quickly retrieve the highest priority element, they are often used in real-time systems where time is of the essence.

## ****Disadvantages of Priority Queue:****

* High complexity. Priority queues are more complex than simple data structures like arrays and linked lists, and may be more difficult to implement and maintain.
* High consumption of memory. Storing the priority value for each element in a priority queue can take up additional memory, which may be a concern in systems with limited resources.
* It is not always the most efficient data structure. In some cases, other data structures like heaps or binary search trees may be more efficient for certain operations, such as finding the minimum or maximum element in the queue.
* At times it is less predictable:. This is because the order of elements in a priority queue is determined by their priority values, the order in which elements are retrieved may be less predictable than with other data structures like stacks or queues, which follow a first-in, first-out (FIFO) or last-in, first-out (LIFO) order.



**Q.6. Compare stack and queue.**

**Ans:**

**Difference between Stack and Queue Data Structures are as follows:**

| **Stacks** | **Queues** |
| --- | --- |
| A stack is a data structure that stores a collection of elements, with operations to push (add) and pop (remove) elements from the top of the stack. | A queue is a data structure that stores a collection of elements, with operations to enqueue (add) elements at the back of the queue, and dequeue (remove) elements from the front of the queue. |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Stacks are often used for tasks that require backtracking, such as parsing expressions or implementing undo functionality. | Queues are often used for tasks that involve processing elements in a specific order, such as handling requests or scheduling tasks. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on recursion. | Queue is used in solving problems having sequential processing. |
| Stacks are often used for recursive algorithms or for maintaining a history of function calls. | Queues are often used in multithreaded applications, where tasks are added to a queue and executed by a pool of worker threads. |
| Stack does not have any types. | Queue is of three types – 1. Circular Queue 2. Priority queue 3. double-ended queue. |
| Can be considered as a vertical collection visual. | Can be considered as a horizontal collection visual. |