Data Structures & Algorithms

Queue

Queue

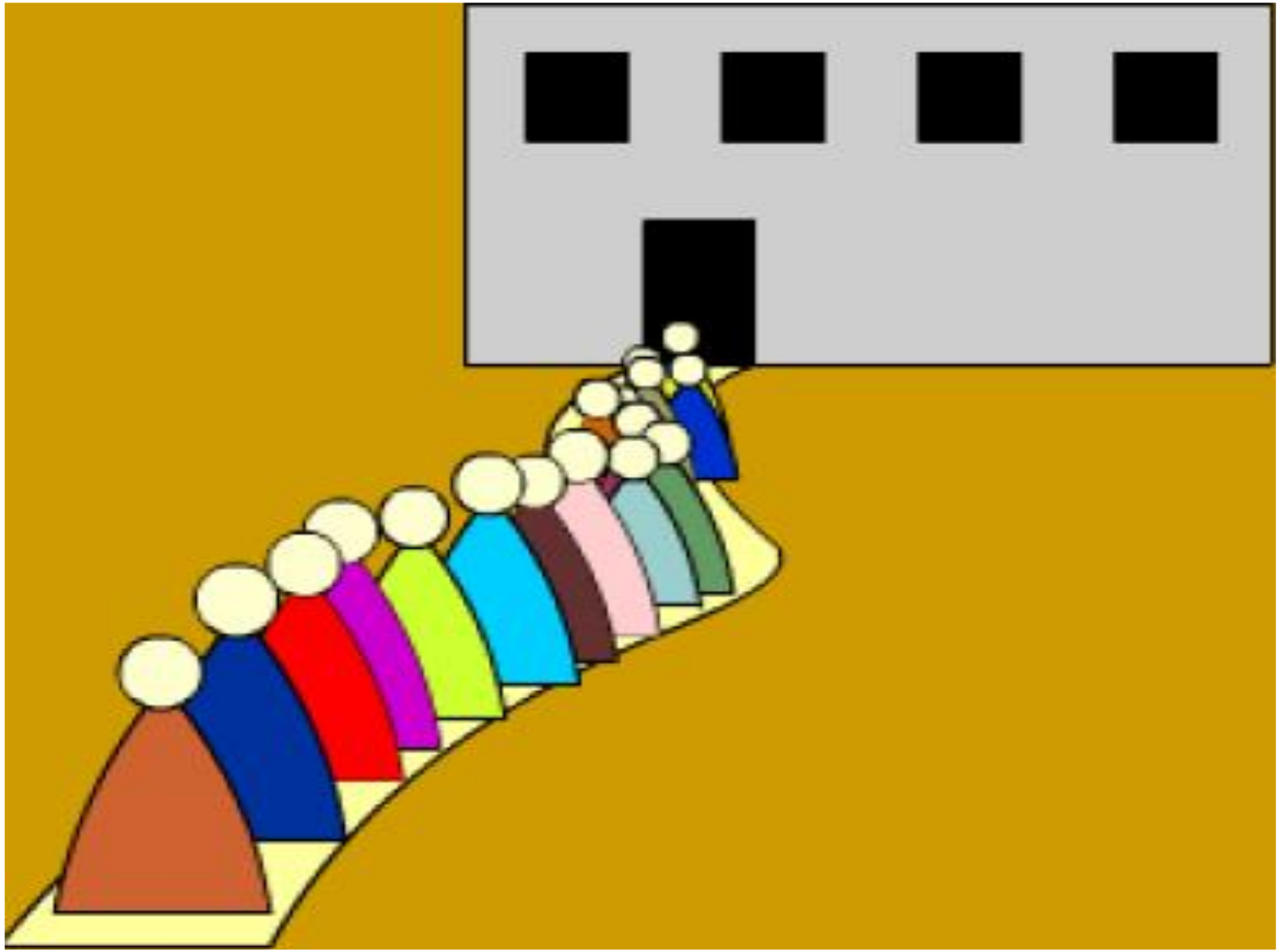
A queue is an ordered collection of similar items, where deletion take place only at one end, called the FRONT, and insertions take place only at the other end, called the REAR.

Items are removed from a queue in the same order as they were inserted.

A queue is a data container where data elements are inserted and removed according to the First-In-First-Out (FIFO) principle.

Real life example
Student standing in a line/queue at plate counter in cafeteria
- the person who goes in first is the person who served first, i.e. comes out first too.

Queue



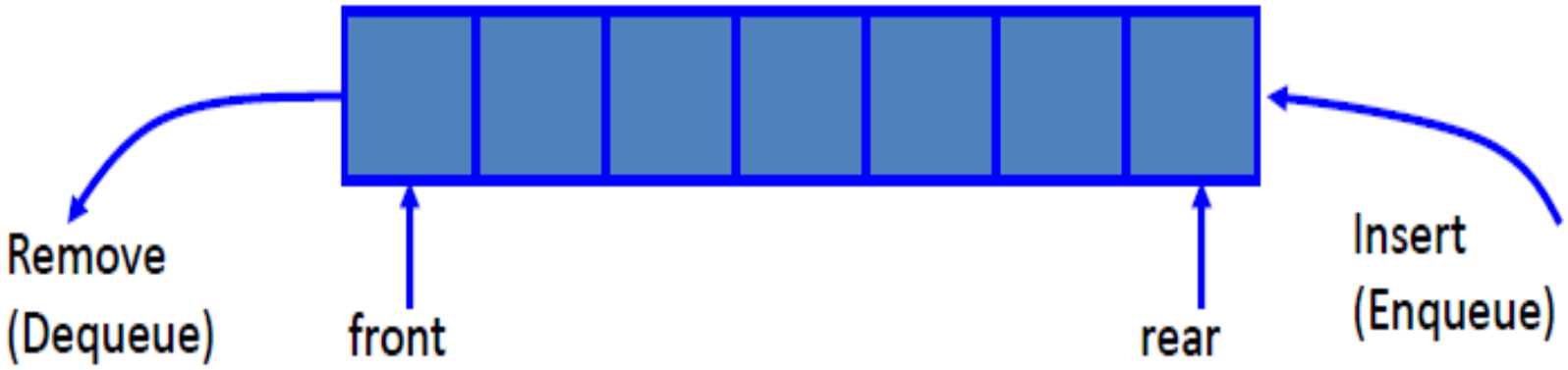
Queue

Operations

Two main operations are

enqueue - The operation of adding an element to the rear of the queue

dequeue - The operation of removing an element from the front of the queue

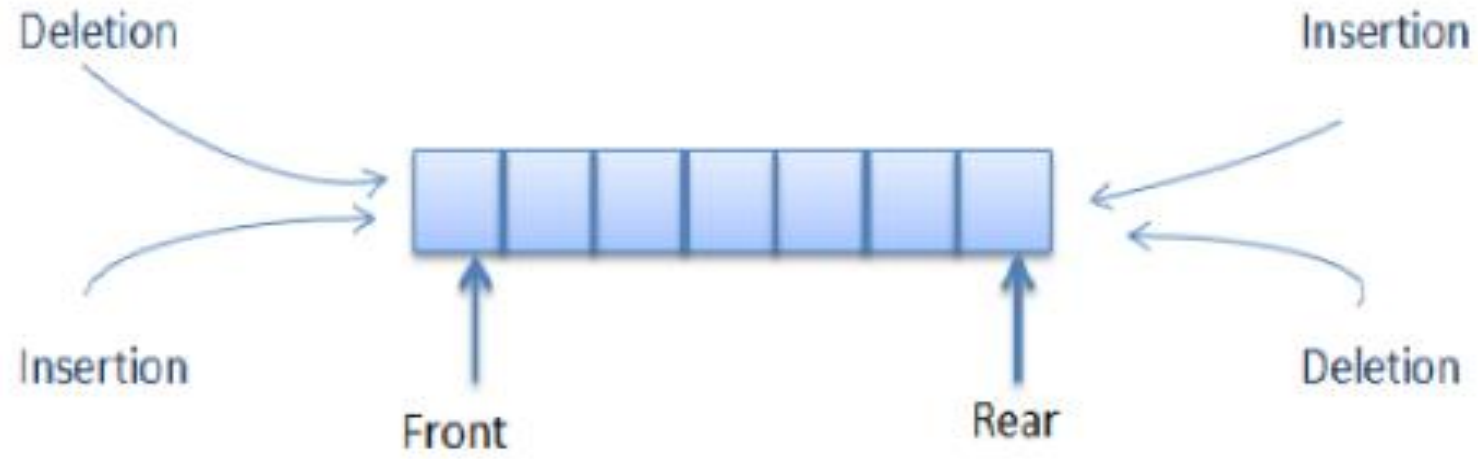


Various Queues

- Normal queue (FIFO)
- Circular Queue
- Double-ended Queue (Deque)
- Priority Queue

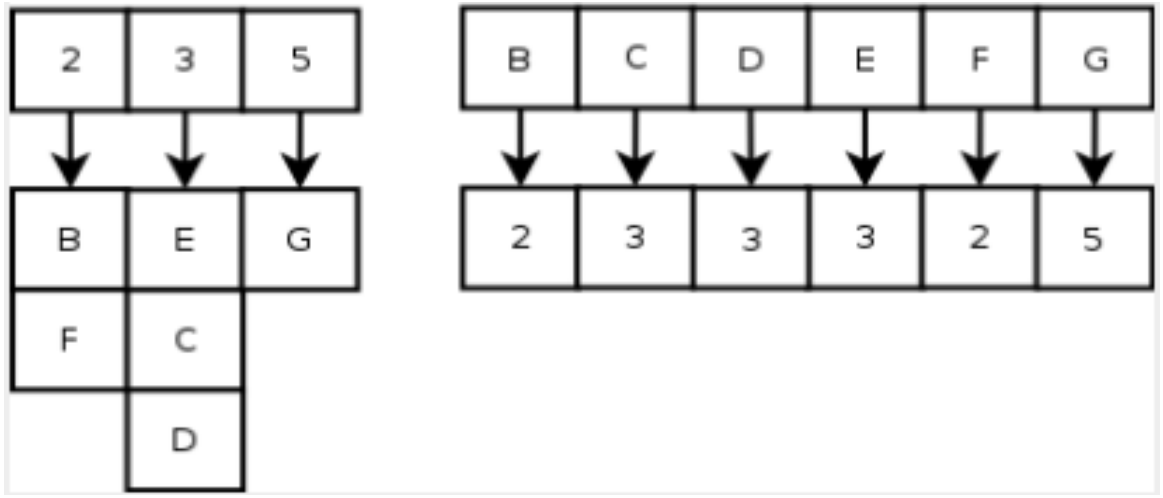
Various Queues

Double ended queue **(deque) -** A queue for which elements can be added to or removed from either ends, i.e. the front or rear. This differs from the normal queue or first in first out list (FIFO), where elements can only be added to one end and removed from the other.



Various Queues

Priority queue - The priority queue is similar to a queue, except each element additionally has a numerical priority associated with it. In a priority queue, an element with high priority is served before an element with low priority. Two element of same priority are processed on first-come-first-served basis.



Queue

Implementations

Array based implementation
Circular array-based implementation
Singly Linked List based implementation

Array Implementation of Queue

0	1	2	3	4	5

enqueue(10)

0	1	2	3	4	5
10					

front=-1 rear=0

enqueue(20)

0	1	2	3	4	5
10	20				

front=-1 rear=1

dequeue()

0	1	2	3	4	5
	20				

front=0 rear=1

Array Implementation of Queue

Initially, front=rear=-1

```
Enqueue(Q[], item, size, rear)
{  //insert item into the queue represented in Q[0:n-1]
   if (rear == size-1) {
       print (“QUEUE is FULL”);
       return;
   }
   rear = rear + 1;
   Q[rear] = item;
}
```

Time Complexity : O(1)

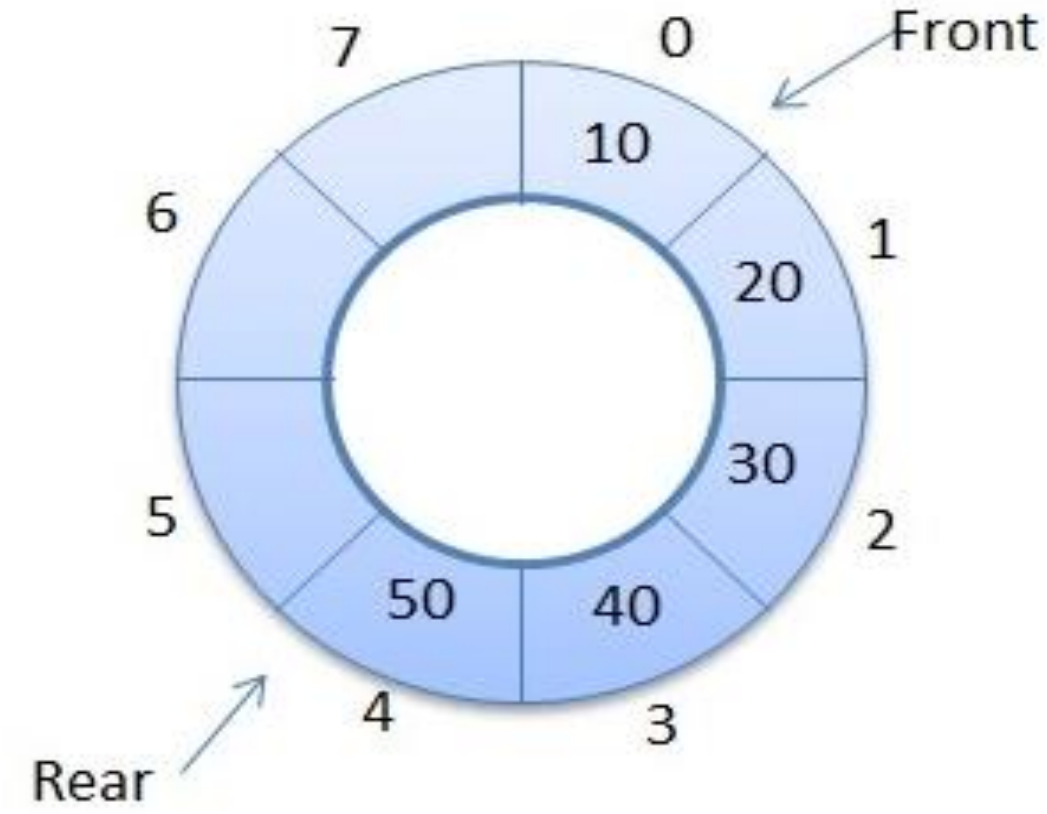
Array Implementation of Queue

```
Dequeue(Q[], front, rear)
{
   if (front == rear) {
        print("QUEUE is EMPTY");
        return;
    }
    front = front + 1;
    return Q[front];
}
```

Time Complexity : O(1)

Problem with Array Implementation of Queue

The main disadvantage of linear queue using array is that even after deletion of elements from the queue, new elements cannot be added in their place in the queue, i.e. the position cannot be reused, as a result space wastage.



Circular Array Implementation of Queue

```
CEnqueue(Q[], item, size, rear, front)
{
   rear = (rear + 1)%size;
   if (rear == front+1) {
       print (“QUEUE is FULL”);
       return;
   }
   Q[rear] = item;
}
```

Time Complexity : O(1)

Circular Array Implementation of Queue

```
CDequeue(Q[], front, rear, size)
{
    if (front == rear) {
        print(“QUEUE is EMPTY”);
        return;
    }
    front = (front + 1)%size;
    return Q[front];
}
```

Time Complexity : O(1)

Linked list Implementation of Queue

Enqueue
Insert at the end of a singly linked list

Dequeue
Deletion from the beginning of a singly linked list

Queue

Applications
Networked/shared printers
Mail servers
CPU process scheduling
Client-server models