1.What is a circular queue?

The front and back of the queue are connected in a circular queue, which is a variation on the standard queue data structure. The last element is connected to the first element, forming a circular pattern. When the queue is full, new elements can cyclically replace the older ones, making efficient use of the available memory.

2. What are the characteristics of the circular queue?

The characteristics of a circular queue are as follows:
- It follows the First-In-First-Out (FIFO) principle, like a linear queue.
- The rear and front of the queue are connected in a circular manner.
- When the queue is full, new elements overwrite the oldest elements in a cyclic fashion.
- It can be implemented using an array or a linked list.
- Circular queues have a fixed size, and their capacity is determined during initialization.

3. Give applications of the circular queue.

The circular queue finds applications in various scenarios, some of which include:

- Bounded Buffer: In computer programming, a circular queue is used as a bounded buffer to efficiently manage data between producers and consumers.

- Process Scheduling: In operating systems, a circular queue is employed for process scheduling, where the next process to be executed is selected in a circular fashion.

- Keyboard Input Buffer: To manage keyboard input, circular queues can be used to store keystrokes until they are processed by the computer.

- Traffic Management: In traffic signal control systems, circular queues help manage vehicle movements efficiently.

-Resource Management: In systems with limited resources, circular queues can be used to allocate and deallocate resources in a cyclic manner.

4. What is the algorithm of the circular queue?

The circular queue algorithm involves the following operations:

- Enqueue: Adds an element to the rear of the queue. If the queue is full, it overwrites the oldest element.
- Dequeue: Removes the element from the front of the queue. If the queue is empty, no removal takes place.
- isEmpty: Checks if the queue is empty.
- isFull: Checks if the queue is full.
- Front: Retrieves the element at the front of the queue without removing it.

5. Write a simple program of a circular queue.

Below is a simple implementation of a circular queue using an array in Python:

PYTHON

```
class CircularQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = self.rear = -1

    def is_empty(self):
        return self.front == -1

    def is_full(self):
        return (self.rear + 1) % self.capacity == self.front

    def enqueue(self, item):
        if self.is_full():
            print("Queue is full. Cannot enqueue.")
        elif self.is_empty():
            self.front = self.rear = 0
            self.queue[self.rear] = item
        else:
            self.rear = (self.rear + 1) % self.capacity
            self.queue[self.rear] = item

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty. Cannot dequeue.")
        elif self.front == self.rear:
            data = self.queue[self.front]
```

```
            self.front = self.rear = -1
            return data
        else:
            data = self.queue[self.front]
            self.front = (self.front + 1) % self.capacity
            return data

    def display(self):
        if self.is_empty():
            print("Queue is empty.")
        else:
            index = self.front
            while True:
                print(self.queue[index], end=" ")
                if index == self.rear:
                    break
                index = (index + 1) % self.capacity
            print()
```

6. Compare and contrast linear queues and circular queues.

Both circular queues and linear queues are types of data structures that adhere to the FIFO principle.
- In both kinds of queues, enqueue and dequeue actions are carried out equally.
- They can be implemented using linked lists or arrays.

Contrast:

Linear queues have a fixed size and can become inefficient when dequeued frequently. Circular queues efficiently reuse empty spaces due to their circular structure. Linear queues have unused spaces at the front, while circular queues wrap around to allow enqueueing. Circular queues require extra information (front and rear pointers) to track their position within the circular array, while linear queues only need front and rear pointers without wrap-around consideration.

Overall, circular queues are more efficient when frequent enqueue and dequeue operations are required, as they make better use of the available memory.