Algorithm of Circular queue: FIFO

1. Declare necessary variable:
   i.e size=5, front=0, rear=-1, queue[size], count=0
2. For **Enqueue** operation:
   Check queue full condition **i.e count==size**
   If queue is full

   Display "**The queue is full**" message.
   Stop.
   else
   Read the data to be inserted [DTBI]
   Increment the rear by 1 **i.e rear=(rear+1)%size;**
   Store DTBI in queue[rear] **i.e queue[rear]=newdata**
   Increment the count by 1 **i.e count++;**

3. To enqueue next data, repeat step 2.

4. For **Dequeue** operation:
   Check queue empty condition **i.e count==0**
   If queue is empty

   Display "**The queue is empty**" message.
   Stop.
   else
   Display the value of **queue[front]** i.e Display "**The data deleted from the queue is %d**" message.
   Increment the front by 1 **i.e front=(front+1)%size;**
   Decrement the count by 1 **i.e count--;**

5. To dequeue next data, repeat step 4.

```
/* Circular queue lab number 3*/
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define size 5
int front=0,rear=-1,count=0;
int queue[size];

void enqueue()
    {
        if(count==size)
                {
                        printf("Queue is full\n");
                }
        else
                {
                        int data;
                        printf("Insert a data into a circular queue:\n");
                        scanf("%d",&data);
                        rear=(rear+1)%size;
                        queue[rear]=data;
                        count++;
                }
    }
void dequeue()
     {
        if(count==0)
                {
                        printf("Queue is empty\n");
                }
        else
                {
                        int data;
                        data=queue[front];
                        printf("The dequeue data is: %d",data);
                        front=(front+1)%size;
                        count--;
                }
     }
int main()
{
        int choice;
        clrscr();
        do
        {
        printf("\nEnter your choice\n1.Enqueue\n2.Dequeue\n3.Exit\n");
                scanf("%d",&choice);
                switch(choice)
                {
```

```
                case 1:
                        enqueue();
                        break;

                case 2:
                        dequeue();
                        break;

                case 3:
                        exit(0);
                }
        }while(choice<=3);
        getch();
        return 0;
}
```

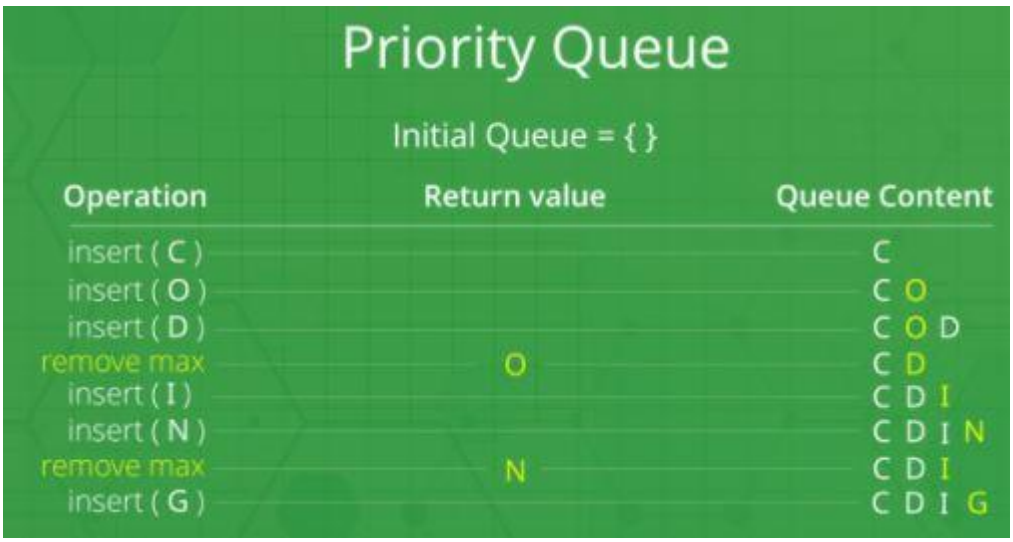
Basis of comparison	Linear Queue	Circular Queue
Meaning	The linear queue is a type of linear data structure that contains the elements in a sequential manner.	The circular queue is also a linear data structure in which the last element of the Queue is connected to the first element, thus creating a circle.
Insertion and Deletion	In linear queue, insertion is done from the rear end, and deletion is done from the front end.	In circular queue, the insertion and deletion can take place from any end.
Memory space	The memory space occupied by the linear queue is more than the circular queue.	It requires less memory as compared to linear queue.
Memory utilization	The usage of memory is inefficient.	The memory can be more efficiently utilized.
Order of execution	It follows the FIFO principle in order to perform the tasks.	It has no specific order for execution.

1. **Priority queue:**
   Priority Queue is an extension of queue with following properties.
   a. Every item has a priority associated with it.
   b. An element with high priority is dequeue before an element with low priority.
   c. If two elements have the same priority, they are served according to their order in the queue i.e first come first serve.

So it is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes from the rules:

i. An element of higher priority is processed before any element of lower priority.
ii. Two elements with same priority are processed according to the order they were added to the queue.

In the below priority queue, element with maximum ASCII value will have the highest priority.



In a queue, the **first-in-first-out rule** is implemented whereas, in a priority queue, the values are removed **on the basis of priority**. The element with the highest priority is removed first.