NAME: AYUSH PRASHANT KALASKAR.

CLASS: S.E. COMP -I.

ROLL NO. : 69 PRN.: F22111074

SEMESTER: SEM-III (2023-2024)

SUBJECT: DATA STRUCTURE LABORATORY: (DSL):-

ASSIGNMENT: NO: E-32 :-

D ALGORITHM FOR THE PROGRAM CODE :-

⇒ STEP: 1:- START

STEP:2:- Create a class with class_name 'pizza' which defines a pizza order system with a circular queue to manage orders.

STEP:3:- Declare the three private data members porder[] , front and rear.

STEP:4:- In the default constructor pizza(), intialize 'front' and 'rear' to -1 to indicate an empty order queue.

STEP:5:- In Function 'qfull()', check if the queue order is full by comparing 'front' and (rear+1) % size. Return 1 if the queue is full or zero otherwise.

STEP:6:- In Function 'qempty()', check if the order queue is empty by checking if 'front' is -1 and Return 1 if the queue is empty or zero otherwise

STEP:7:- Function 'accept-order (int item):'

- Accept an integer item 'n' representing the pizza order and check if the queue is empty using 'qempty()'. If Empty, display an error message.
- If not empty, display the delivered orders, calculate the total payment, and update the queue accordingly

STEP:8:- Function 'order_in_queue()'

Display the pending orders in the queue if any.

STEP:9.: MAIN FUNCTION:-

Create an instance (object) of class' pizza' named 'p1'

STEP:10:- Use a Do-While loop to display a menu and perform pizza parlour operations based on the user input:

- 1: Accept a Pizza Order.
- 2: Make a payment for delivered Pizzas.
- 3: Viewing pending orders
- 4: Exit the program.

STEP:11:- Based on the user choice, perform the following Operations.

- For choice 1, Accept a pizza order, and add it to the queue using 'accept-order()'
- For choice 2, specify the number of pizzas to be delivered, calculate the total payment, and display the delivered orders using 'make-payment()'
- For choice 3, view the pending orders in the queue using 'order-in-queue()';

Repeat the Loop until the user chooses to Exit.

□ PSEUDOCODE:-

```
⇒ class Pizza
    Integer Array porder[size]
    Integer front, rear.

    Pizza()    // Default Constructor.
      Set front to -1
      set rear to -1

    Function qFull() returns Integer
      IF front is equal to (rear + 1) % size then
         Return 1
      Else
         Return 0

    Function qEmpty() returns Integer
      IF Front is equal to -1 then
         Return 1
      Else
         Return 0

    Function accept_order(item: integer)
      IF qfull() equals 1 then
         Display "Very Sorry !!!! No more Orders....."
      Else
         IF Front is equal to -1 then
            set front to 0
            set rear to 0
         Else
            set rear to (rear+1) % size
      (End if)
      set porder[rear] = item
```

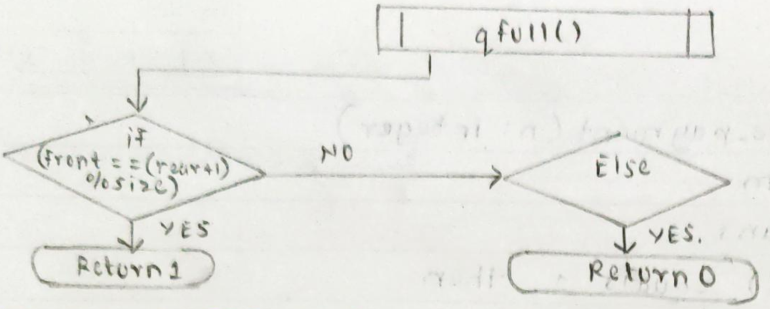
```
Function make_payment (n: integer)
   Integer item
   character ans
   IF qempty () equals 1 then
     Display "Sorry !!!! Order is not there....."
   Else
     Display " Delivered orders as follows
     For i from 0 to n-1
       Set item to porder[front]
       IF front is equal to rear then
          Set front to -1
          Set rear to -1
       Else
          Set front to (front + 1) % size
    (End if)
      Display item
   (End for)
   Display " Total Amount to pay :" + n*100
   Display " Thank You. Visit Again.... "

Function order_in_queue()
   Integer temp
   If qempty () equals 1 then
     Display " Sorry !! There is no pending order.... "
   Else
     Set temp to front
     Display " Pending Orders as follows.... "
     While temp is not equal to rear
       Display porder[temp]
       set temp to (temp + 1) % size
     End while
     Display porder[temp]
  (End if)
(End class)
```

```
Function main()
  Pizza p1
  Integer ch, k, n
  Do
     Display " ***** Welcome to Pizza Parlour ***** "
     Display " 1. Accept Order. "
     Display " 2. Make Payment. "
     Display " 3. Pending Orders. "
    Display " Enter your choice."
    Read ch
    Switch ch
        Case 1:
            Display " Which pizza would you ~~hav~~ like to have today"
            Display " 1. Veg Soya Pizza "
            Display " 2. Veg Butter Pizza"
            Display " 3. Egg Pizza. "
            Display " Please Enter your Order. "
            Read k
            p1. accept_order (k)
        Case 2:
            Display " How many Pizza ? "
            Read n
            p1. make_payment (n)
        Case 3:
            Display " Following orders are in queue to deliver.... "
            p1. order_in_queue ()
    End Switch
  While ch is not equal to 4
(End Function)
```

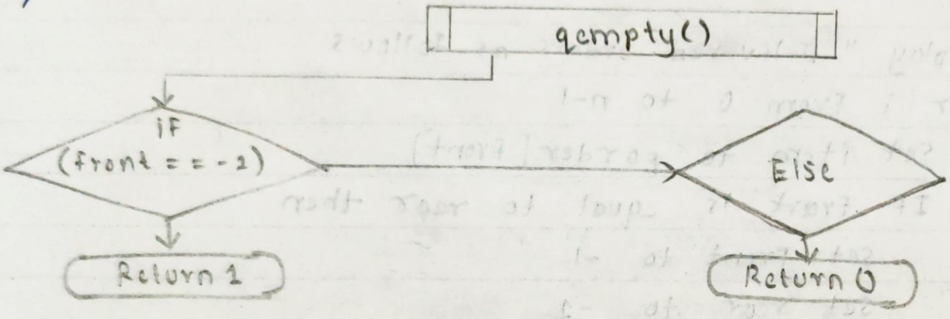
(1.) FLOWCHART FOR THE FUNCTION:- int qfull()

→



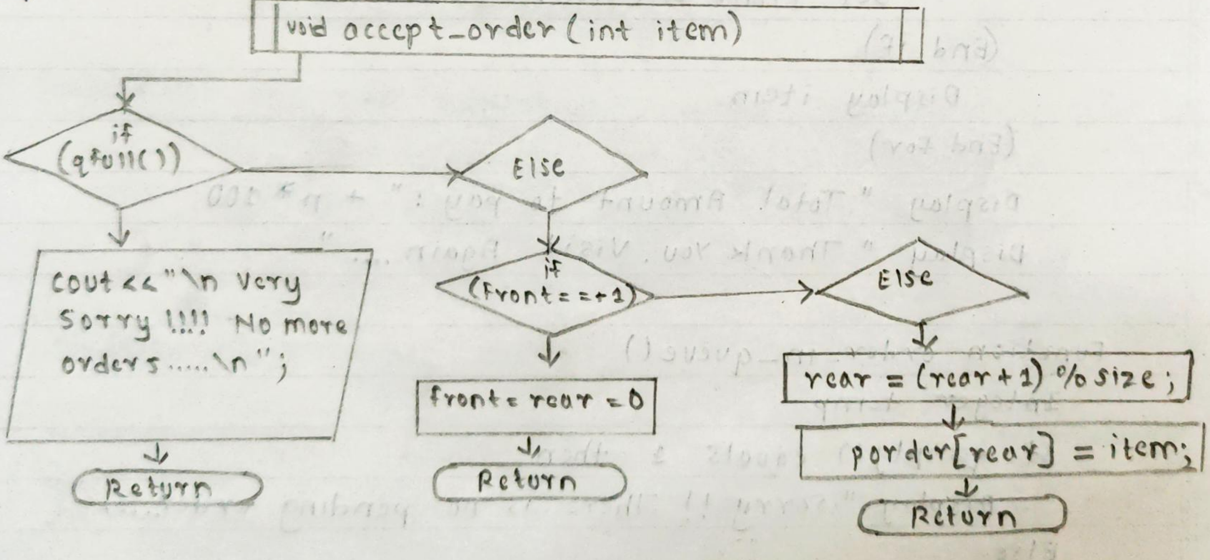
(2.) FLOWCHART FOR THE FUNCTION:- int qempty()

→



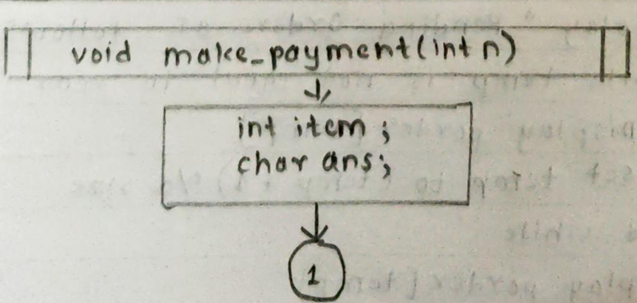
(3.) FLOWCHART FOR THE FUNCTION : accept_order(int item)

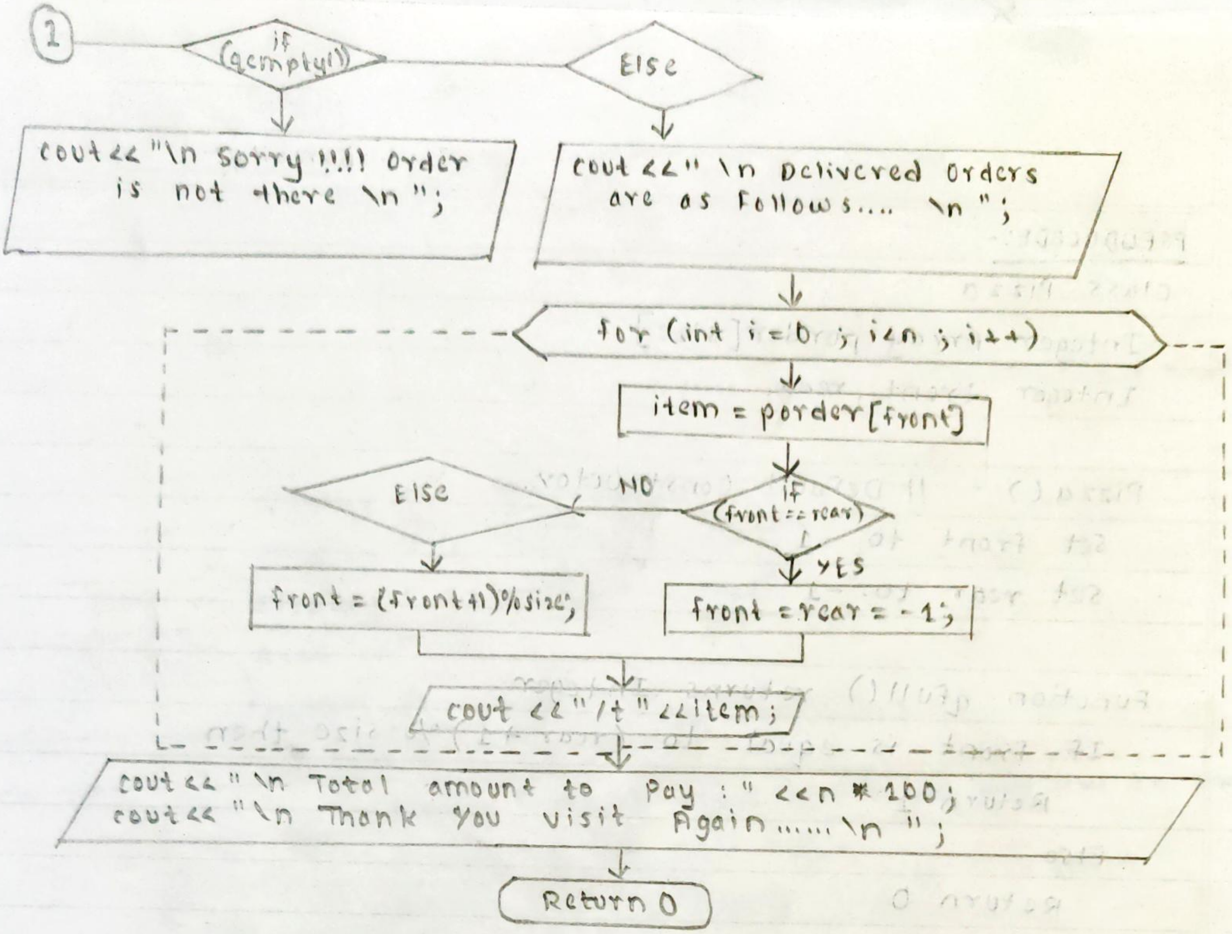
→



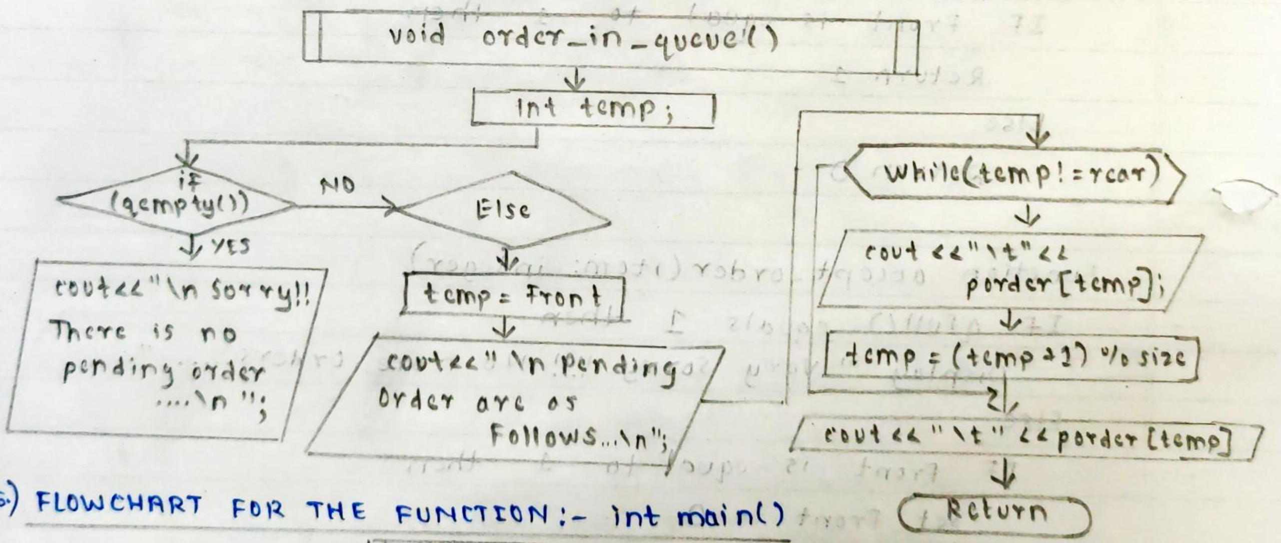
(4.) FLOWCHART FOR THE FUNCTION: void make_payment (int n)

→

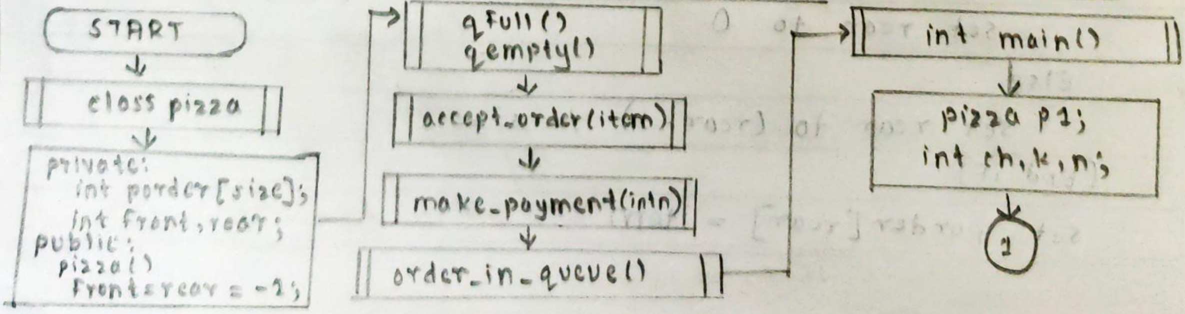


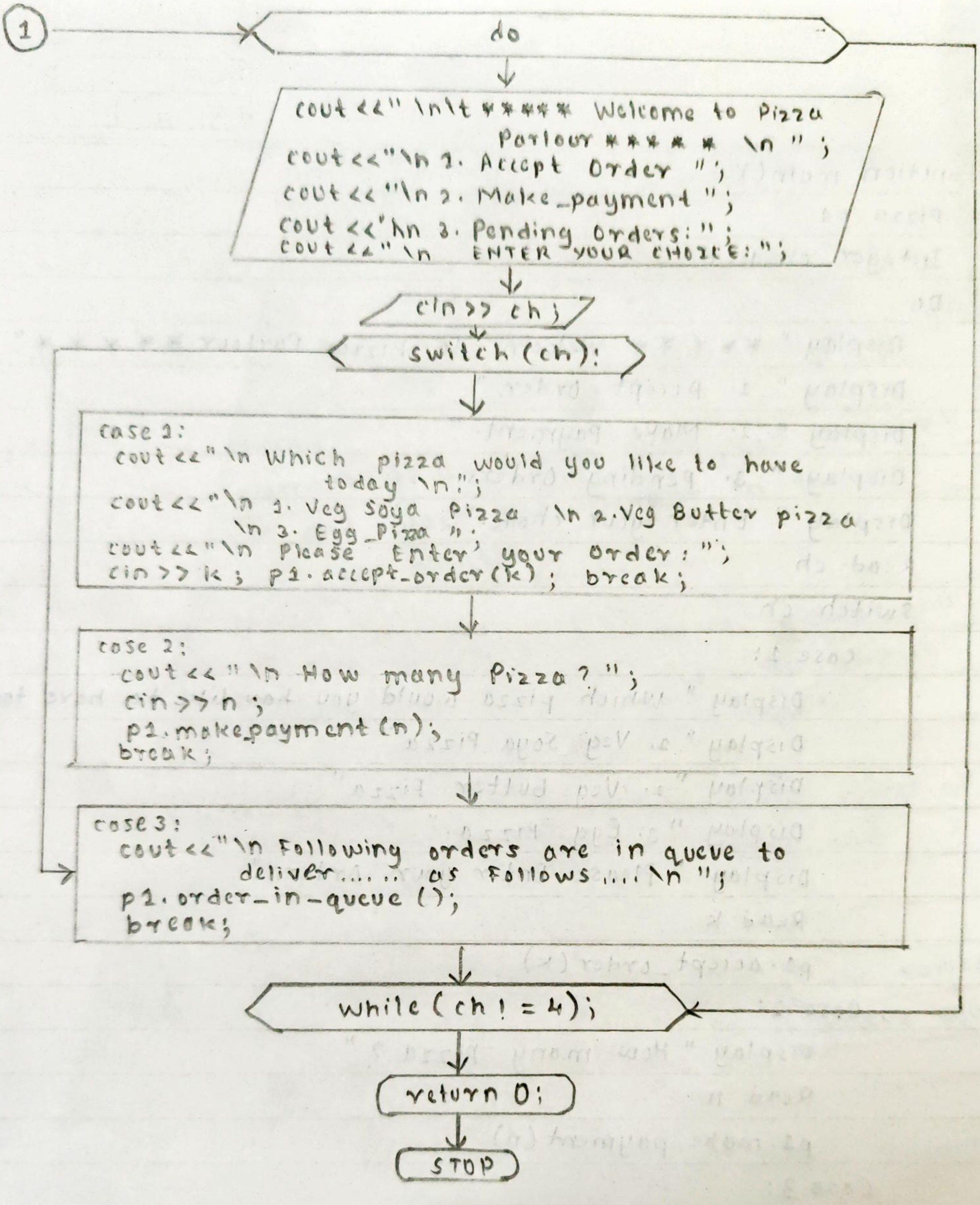

1
if (qempty())
Else
cout<<"\n sorry !!!! order is not there \n";
cout<<" \n Delivered Orders are as Follows.... \n";
for (int i=0 ; i<n ; i++)
item = porder[front]
Else
NO
if (front==rear)
YES
front = (front+1)%size;
front = rear = -1;
cout << "/t" <<item;
cout<<"\n Total amount to pay :" <<n * 100;
cout<<"\n Thank you visit Again...... \n";
Return 0

(5.) FLOWCHART FOR THE FUNCTION:- order_in_queue()


void order_in_queue()
int temp;
if (qempty())
NO
Else
YES
cout<<"\n Sorry!! There is no pending order ....\n";
temp = front
cout<<" \n Pending Order are as Follows...\n";
while(temp!=rear)
cout <<"\t"<< porder[temp];
temp = (temp+1) % size
cout << "\t" <<porder[temp]
Return

(6.) FLOWCHART FOR THE FUNCTION :- int main()





```
do

cout<<" \n\t ***** Welcome to Pizza
            Parlour ***** \n ";
cout<<"\n 1. Accept Order ";
cout<<"\n 2. Make_payment ";
cout<<"\n 3. Pending Orders: ";
cout<<" \n  ENTER YOUR CHOICE: ";

cin>> ch;

switch (ch):

case 1:
 cout<<"\n Which pizza would you like to have
              today \n";
 cout<<"\n 1. Veg Soya Pizza \n 2.Veg Butter pizza
        \n 3. Egg_Pizza ";
 cout<<"\n Please Enter your Order: ";
 cin>> k; p1.accept_order(k); break;

case 2:
  cout<<"\n How many Pizza? ";
  cin>>n;
  p1.makepayment(n);
  break;

case 3:
  cout<<"\n Following orders are in queue to
        deliver...... as Follows....\n ";
  p1.order_in_queue ();
  break;

while (ch != 4);

return 0;

STOP
```

ASSIGNMENT: 3 E-32 :-

D QUESTIONS:-

Q1.] Describe Circular Queue Operations.

ANS. The primary circular Queue Operations are as follows:-

(1.) Intialization

- To create a circular queue, we need to specify its size or capacity. This determines the maximum number of elements that can be stored in a queue.

(2.) Enqueue (Insertion)

- To add an element to a circular queue, you increment the 'rear' pointer and place the element in the slot indicated by the rear pointer.

(3.) Dequeue (Deletion)

- To remove an element from a circular queue, you increment the front pointer and remove the element from the slot indicated by the front pointer.

(4.) Is Full (Over Flow Check)

- This condition is checked by comparing the values of the "front" and "rear" pointers. If they are adjacent, the circular queue is 'Full'.

(5.) Is Empty (Under Flow Check)

- This condition is checked by comparing the values of the "front" and "rear" pointers. If they are equal, the circular queue is 'Empty'.

(6.) Front Element (Peek).

To Examine the element at the front of the circular queue without removing it.

(7.) Size check.

You can check the current number of elements in the circular queue by calculating the absolute difference between the front and rear pointers.

(8.) Circular Behaviour:-

Circular queue automatically wrap around when the Front or rear pointer reaches the end of the queue

(9.) Display

Display operation to show the elements in a circular queue.

Q2.) How is order processing convendrant using circular queue ended queue

ANS. The order prucessing is convenient in oircular queue than in linear queve as:-

i.) The insertion in the queue is from the rear end and in the case of Linear queue of fixed size insertion is not possible when rear reaches the End of the queue.

ii.) But in the case of circular queue, the rear end moves from the last position to the front position circulorly.

iii.) Thus circular queue is more useful than a linear queve due to ease of insertion - deletion and performing various operations. It provides efficient utlization of memory.

Q3.) Describe Time Complexity of a circular Queue.

ANS. The Time Complexity of a common circular queue operations in a well implemented circular queue is typically O(1) (constont time), which means that the time required to do these operations does not depend on the size of the circular queue. The same time complexity O(1) is ochieved in various circular queue operations such as Enqueue, Dequeue, IsFull, Is Empty, Front Element (Peek), Size check. This can be achieved by managing the circular buffer and the front and rear pointers properly.