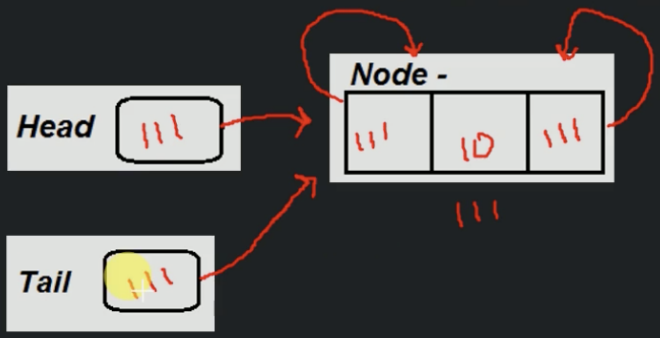
Common operations of Circular Double Linked List:

- ✓Creation of circular Double Linked List
- ✓insertion of node in circular Double Linked List
- ✓Traversal of circular Double Linked List
- ✓Searching a value in circular Double Linked List
- ✓Deletion of a node from a circular Double Linked List
- ✓Deletion of circular Double Linked List

Creation of Circular Double Linked List:




```
CreateCircularDoubleLinkedList(nodeValue):
    create a blank node
    node.value = nodeValue;
    head = node;
    tail = node;
    node.next = node.prev = node;
```

Time Complexity - Creation of Circular Double Linked List:

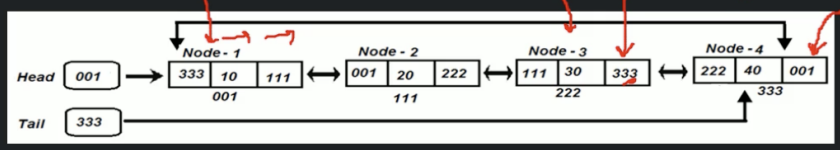
```
CreateCircularDoubleLinkedList(nodeValue):
    create a blank node ---------- O(1)
    node.value = nodeValue ---------- O(1)
    head = node ---------- O(1)
    tail = node ---------- O(1)
    node.next = node.prev = node ---------- O(1)
```

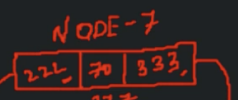
Time Complexity – O(1)

Space Complexity – O(1)

Insertion in Circular Double Linked List:

```
InsertInLinkedList(head, nodeValue, location):
    create a blank node
    node.value = nodeValue;
    if (!existsLinkedList(head))
        return error //Linked List does not exists
    else if (location equals 0) //insert at first position
        node.next = head;   node.prev = tail;
        head.prev= node
        head = node; tail.next = node;
    else if (location equals last) //insert at last position
        node.next = head;  node.prev = tail
        head.prev = node
        tail.next = node
        tail = node   //to keep track of last node
    else //insert at specified location
        loop: tmpNode = 0 to location-1  //loop till we reach specified node
        node.next = tmpNode.next; node.prev = tmpNode;
        tmpNode.next = node; node.next.prev = node;
```





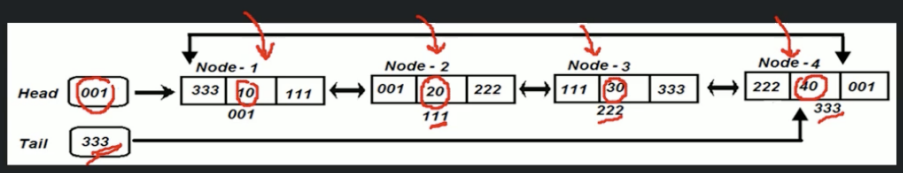
Time Complexity - Insertion in Circular Double Linked List:

```
InsertInLinkedList(head, nodeValue, location):
    create a blank node ---------- O(1)
    node.value = nodeValue ---------- O(1)
    if (!existsLinkedList(head)) ---------- O(1)
        return error //Linked List does not exists ---------- O(1)
    else if (location equals 0) //insert at first position ---------- O(1)
        node.next = head ---------- O(1)
        node.prev = tail ---------- O(1)
        head.prev= node ---------- O(1)
        head = node; tail.next = node ---------- O(1)
    else if (location equals last) //insert at last position ---------- O(1)
        node.next = head;  node.prev = last ---------- O(1)
        head.prev = node ---------- O(1)
        last.next = node ---------- O(1)
        tail = node   //to keep track of last node ---------- O(1)
    else //insert at specified location ---------- O(1)
        loop: tmpNode = 0 to location-1  //loop till we reach specified node ---------- O(n)
        node.next = tmpNode.next; node.prev = tmpNode ---------- O(1)
        tmpNode.next = node; node.next.prev = node ---------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Traversal of Circular Double Linked List:




```
TraverseLinkedList ():
    if head == NULL, then return
    loop: head to tail
        print currentNode.Value
```

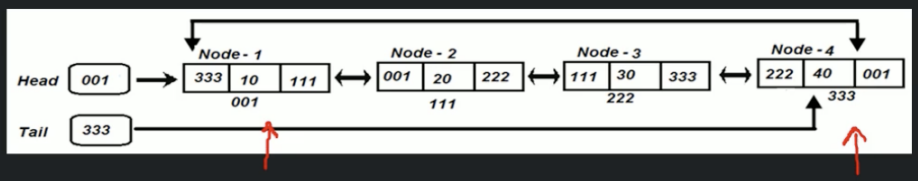
Time Complexity - Traversal of Circular Double Linked List:

```
TraverseLinkedList (head):
    if head == NULL, then return ---------- O(1)
    loop: head to tail ---------- O(n)  ---------- O(n)
        print currentNode.Value ---------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Reverse Traversal of Circular Double Linked List:




```
ReverseTraverseLinkedList ():
    if head == NULL, then return
    loop: tail to head
        print currentNode.Value
```

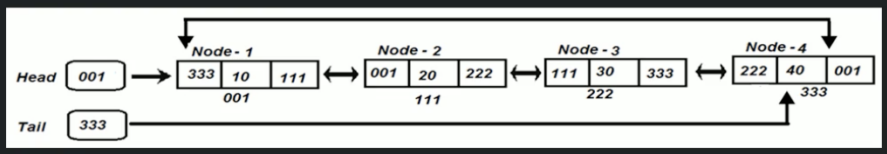
Time Complexity – Reverse Traversal of Circular Double Linked List:

```
ReverseTraverseLinkedList (head):
    if head == NULL, then return ---------- O(1)
    loop: tail to head ---------- O(n)        ---------- O(n)
        print currentNode.Value ---------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Searching a node in Circular Double Linked List:




```
SearchNode(nodeValue):
    loop: tmpNode = head to tail
        if (tmpNode.value equals nodeValue)
            print tmpNode.Value //node value found
            return
    return //nodeValue not found
```

Time Complexity - Searching a node in Circular Double Linked List:

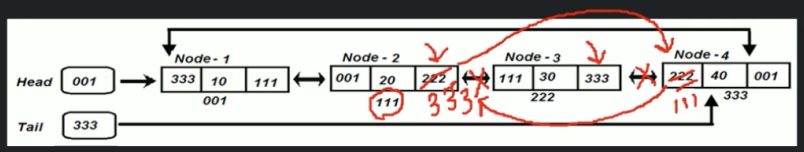
```
SearchNode(nodeValue):
    loop: tmpNode = head to tail ---------- O(n)
        if (tmpNode.value equals nodeValue) ---------- O(1)
            print tmpNode.Value //node value found ---------- O(1)
            return ---------- O(1)
    return //nodeValue not found ---------- O(1)
```

O(1) + O(1) ---------- O(n)

Time Complexity – O(n)

Space Complexity – O(1)

Deletion of node from Circular Double Linked List:




```
DeletionOfNode(head, Location):
    if (!existsLinkedList(head))
        return error //Linked List does not exists
    else if (location equals 0)  //we want to delete first element
        if this was the only element in list, then update head.next = head.prev = head = tail = null; return
        head = head.next; head.prev = tail; tail.next = head;
    else if (location >= last)
        if this was the only element in list, then update head.next = head.prev = head = tail = null; return
        tail = tail.prev; tail.next = head; head.prev=tail
    else // if any internal node needs to be deleted
        loop: tmpNode = head to location-1 //we need to traverse till we find the previous location
        tmpNode.next = tmpNode.next.next  //delete the required node
        tmpNode.next.prev = tmpNode
```

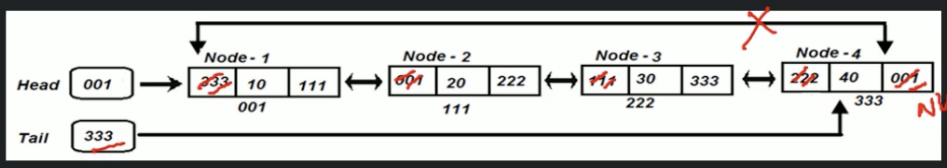
Time Complexity - Deletion of node from Circular Double Linked List:

```
DeletionOfNode(head, Location):
    if (!existsLinkedList(head)) ---------- O(1)
        return error //Linked List does not exists ---------- O(1)
    else if (location equals 0)  //we want to delete first element ---------- O(1)
        if this was the only element in list, then update head.next = head.prev = head = tail = null return ---------- O(1)
        head = head.next; head.prev = null; tail.next = head ---------- O(1)
    else if (location >= last) ---------- O(1)
        if this was the only element in list, then update head.next = head.prev = head = tail = null; return ---------- O(1)
        tail = tail.prev; tail.next = head; head.prev=tail ---------- O(1)
    else // if any internal node needs to be deleted ---------- O(1)
        loop: tmpNode = head to location-1 //we need to traverse till we find the previous location ---------- O(n)
        tmpNode.next = tmpNode.next.next  //delete the required node ---------- O(1)
        tmpNode.next.prev = tmpNode  ---------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Deletion of entire Circular Double Linked List:




```
DeleteLinkedList(head, tail):
    tail.next = null;
    loop(tmp : head to tail)
        tmp.prev = null;
    head = tail = null
```

Time Complexity - Deletion of entire Circular Double Linked List:

```
DeleteLinkedList(head, tail):
    tail.next = null ---------- O(1)
    loop(tmp : head to tail) ---------- O(n)    ---------- O(n)
        tmp.prev = null ---------- O(1)
    head = tail = null ---------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Time & Space Complexity of Circular Double Linked List:

Particulars	Time Complexity	Space Complexity
Creation	O(1)	O(1)
Insertion	O(n)	O(1)
Traversing (Both)	O(n)	O(1)
Searching	O(n)	O(1)
Deletion of a node	O(n)	O(1)
Deletion of Linked List	O(n)	O(1)