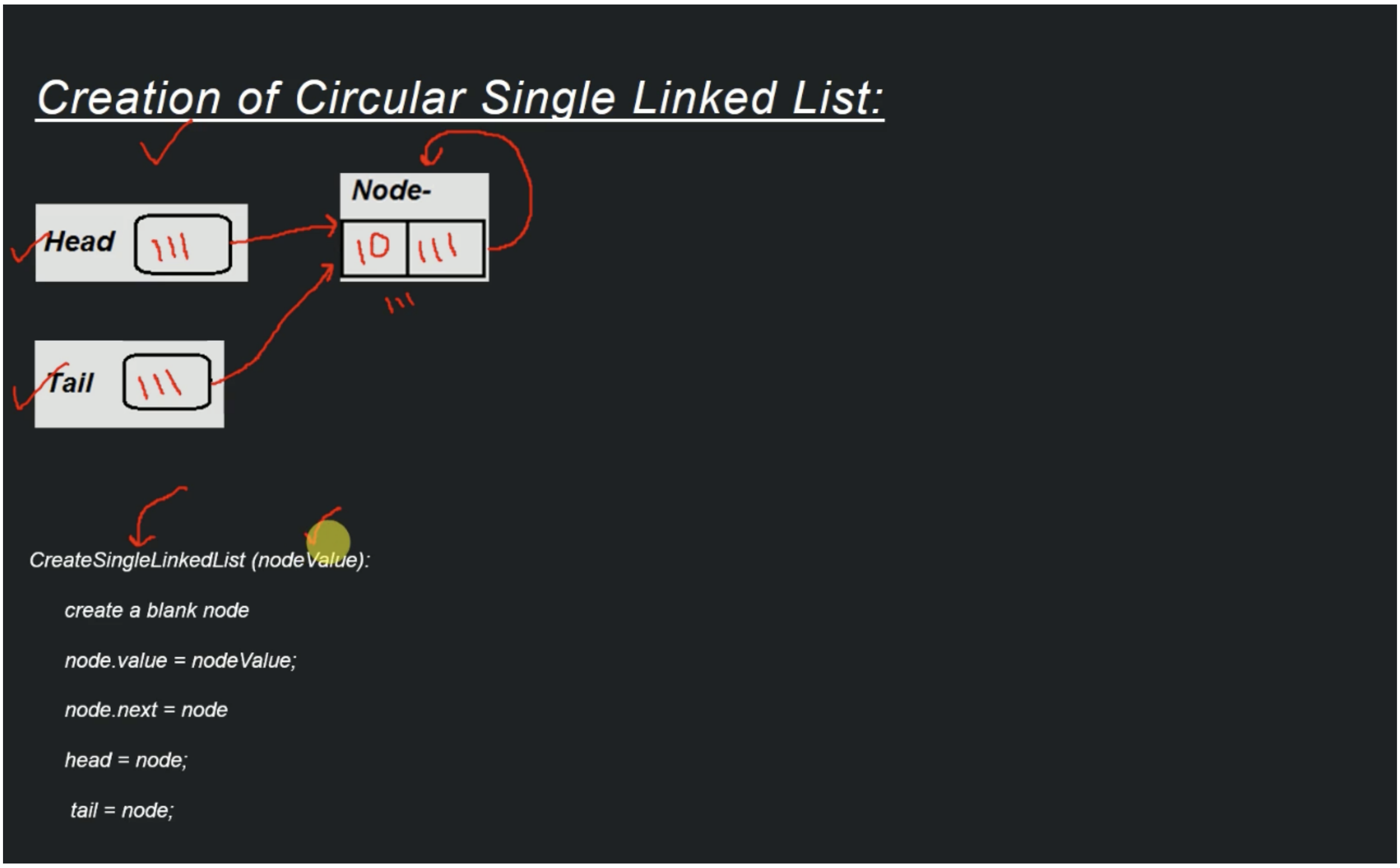
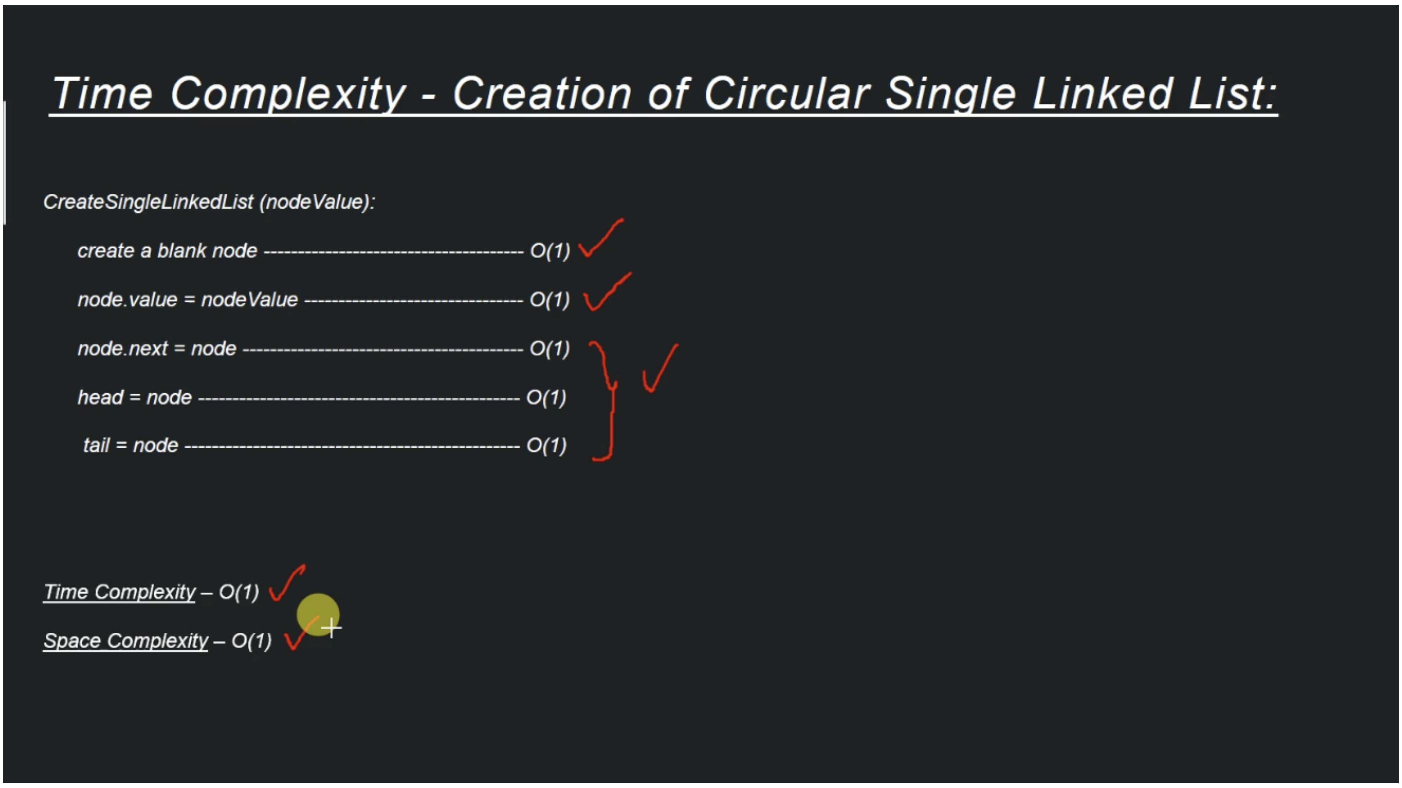


Creation of Circular Single Linked List:

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

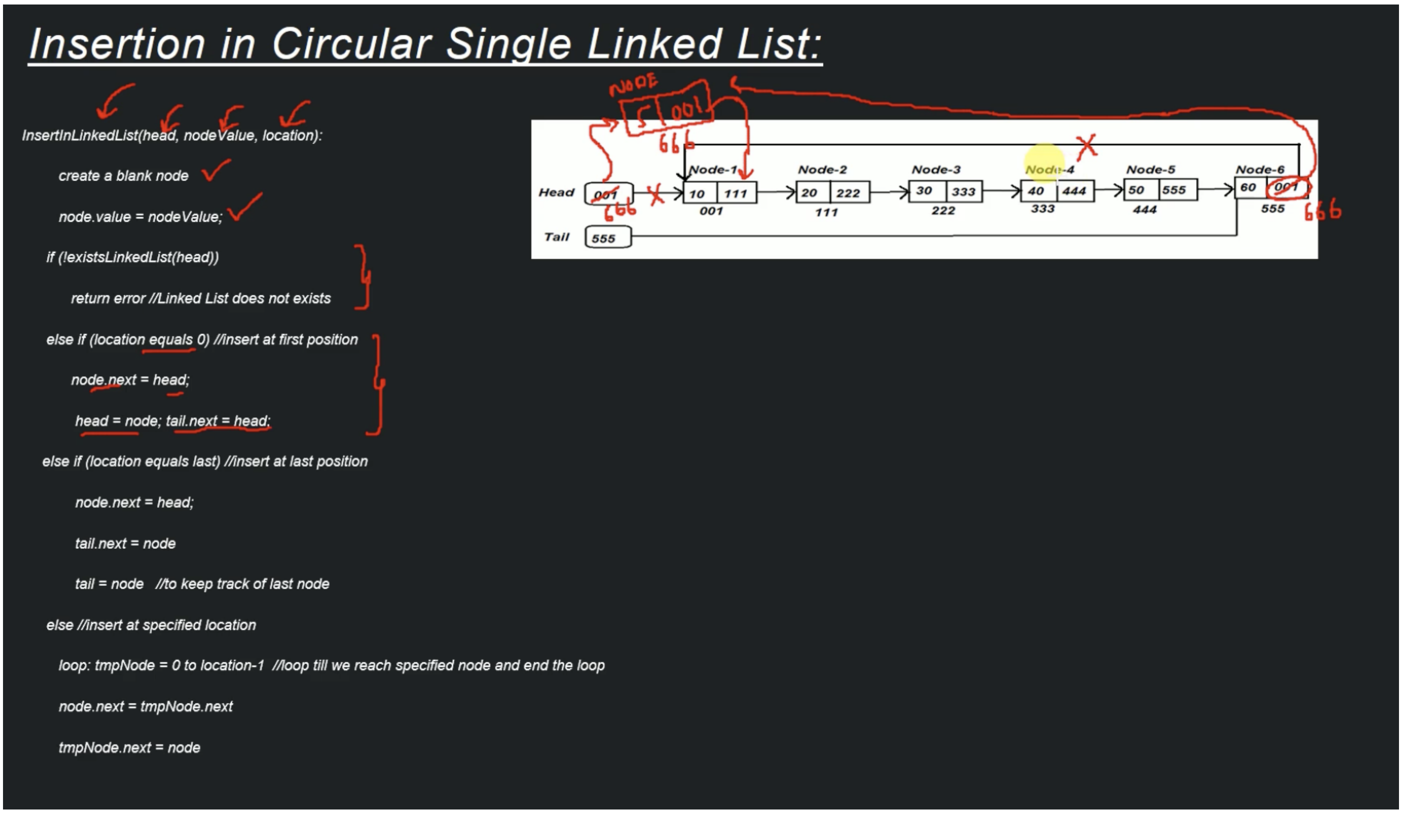


Time Complexity - Creation of Circular Single Linked List:

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

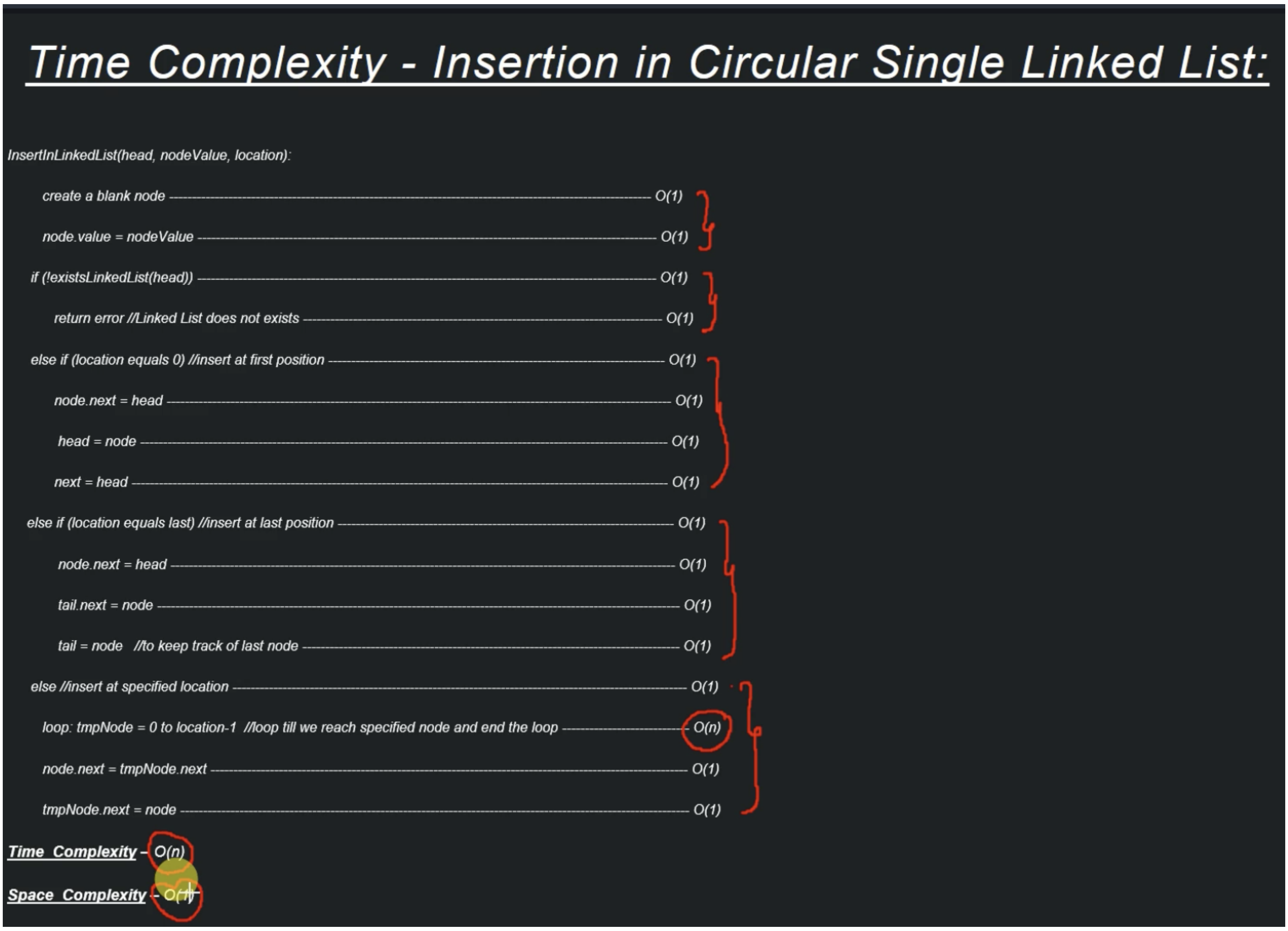
Time Complexity – O(1) ✓

Space Complexity – O(1) ✓



Insertion in Circular Single 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;
         head = node; tail.next = head;
   else if (location equals last) //insert at last position
         node.next = head;
         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 and end the loop
      node.next = tmpNode.next
      tmpNode.next = node
```

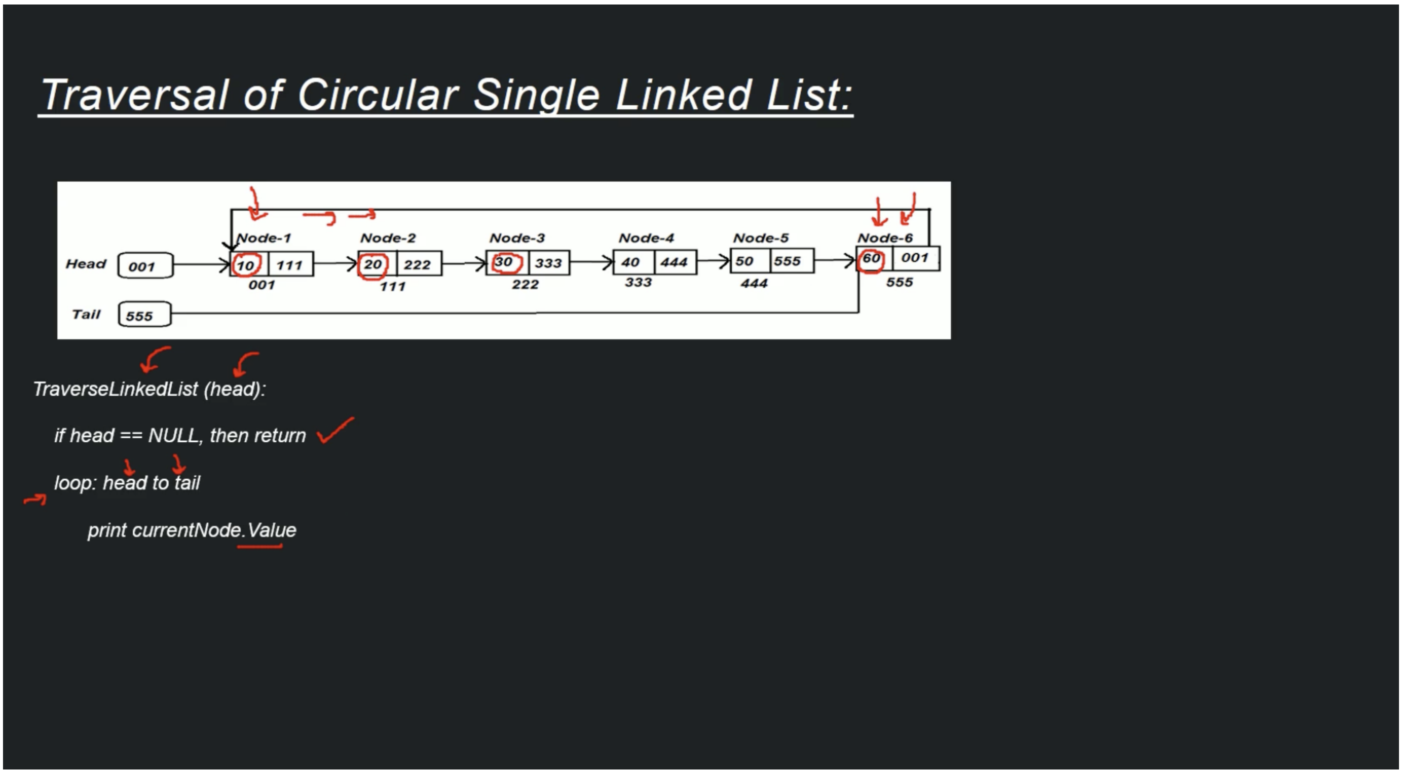


Time Complexity - Insertion in Circular Single Linked List:

```
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)
        head = node ------------------------------------------------ O(1)
        next = head ------------------------------------------------ O(1)
    else if (location equals last) //insert at last position ------- O(1)
        node.next = head ------------------------------------------- O(1)
        tail.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 and end the loop ---- O(n)
      node.next = tmpNode.next ------------------------------------- O(1)
      tmpNode.next = node ------------------------------------------ O(1)
```

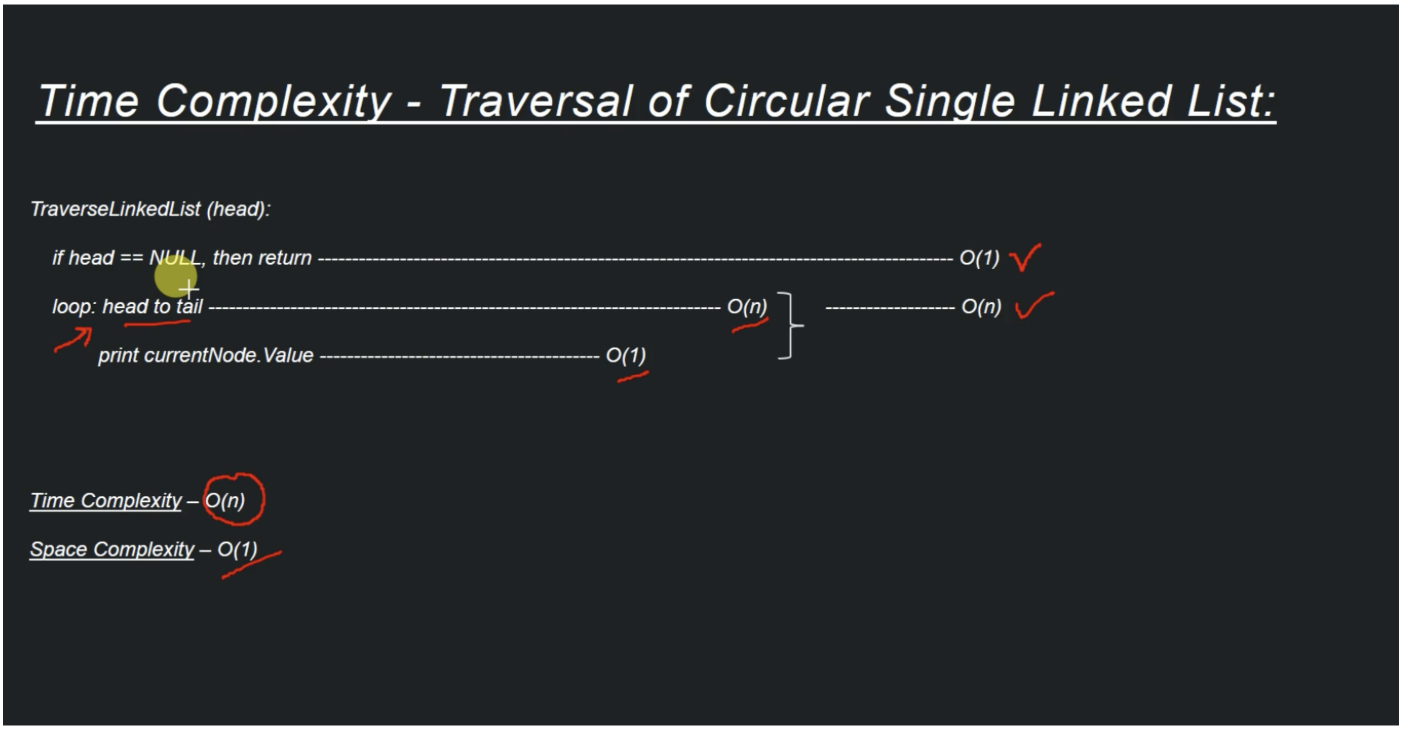
Time Complexity – O(n)

Space Complexity – O(1)



Traversal of Circular Single Linked List:

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

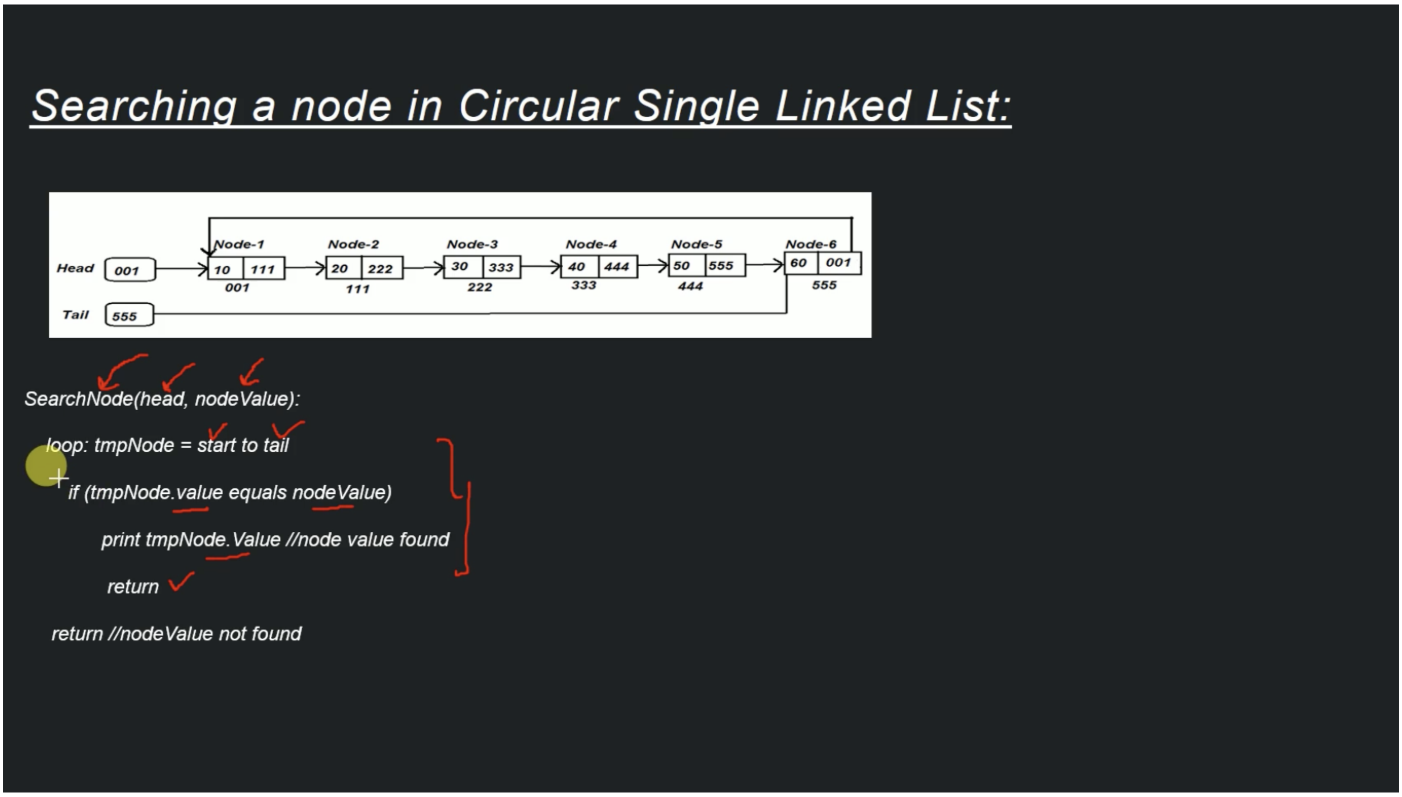


Time Complexity - Traversal of Circular Single Linked List:

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

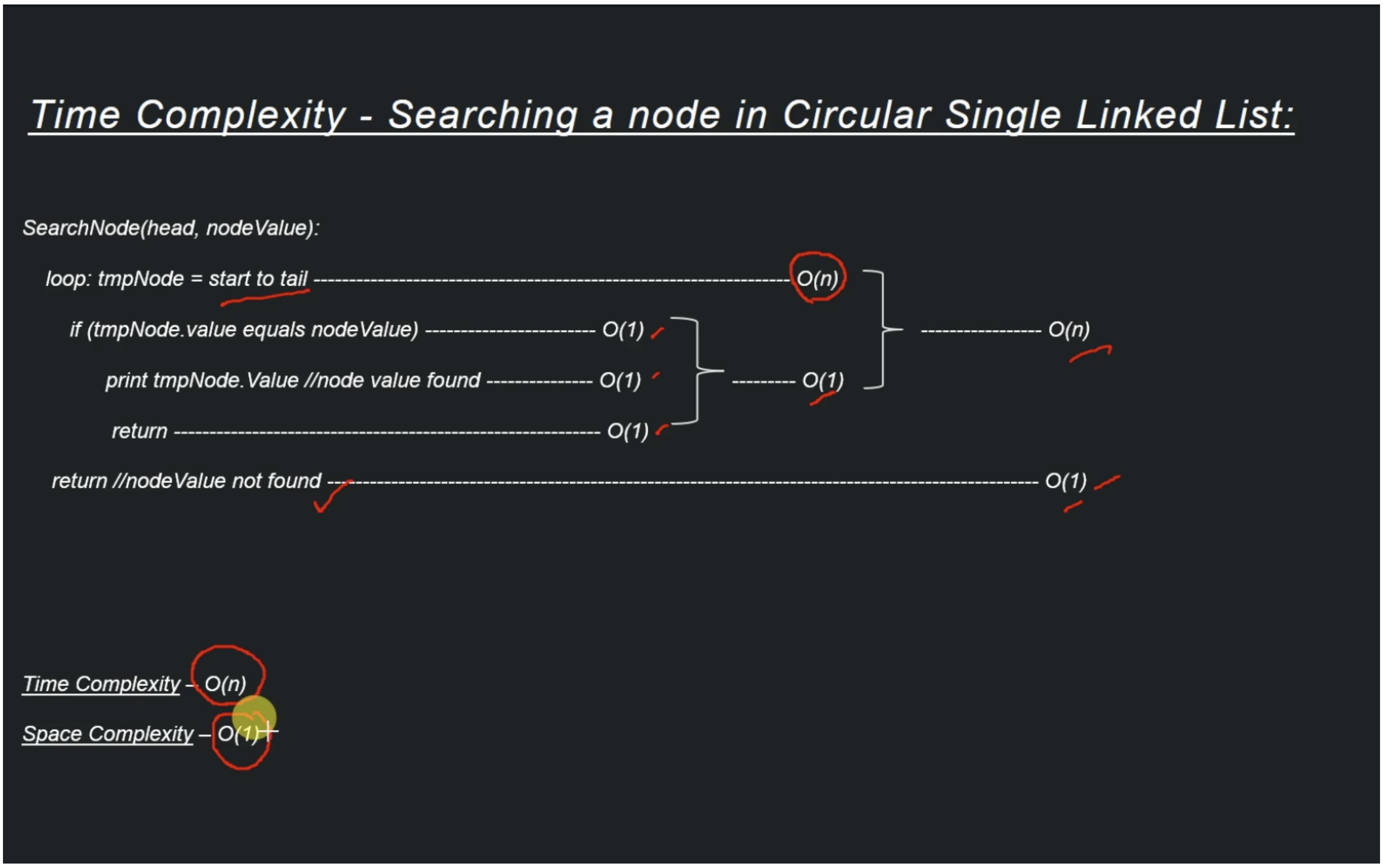
Time Complexity – O(n)

Space Complexity – O(1)



Searching a node in Circular Single Linked List:

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

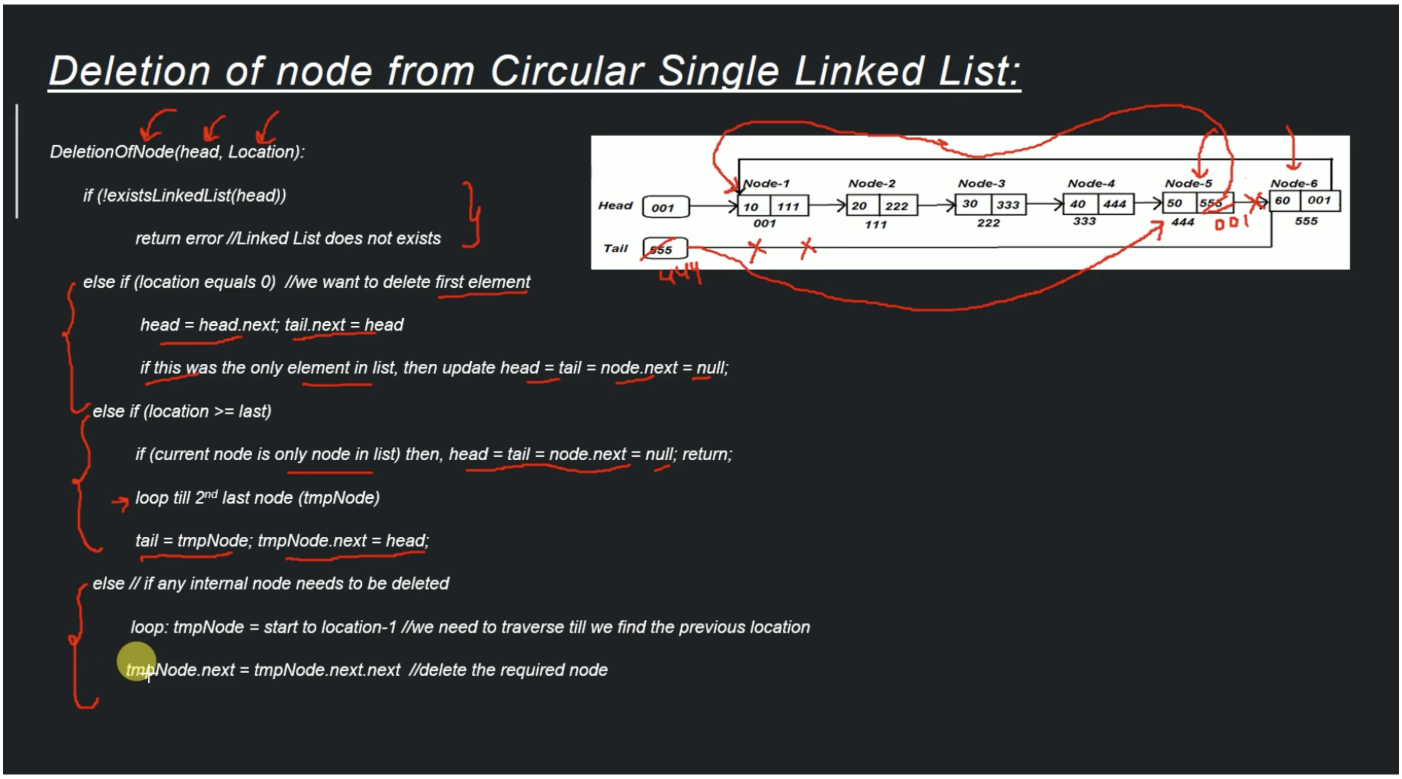


Time Complexity - Searching a node in Circular Single Linked List:

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

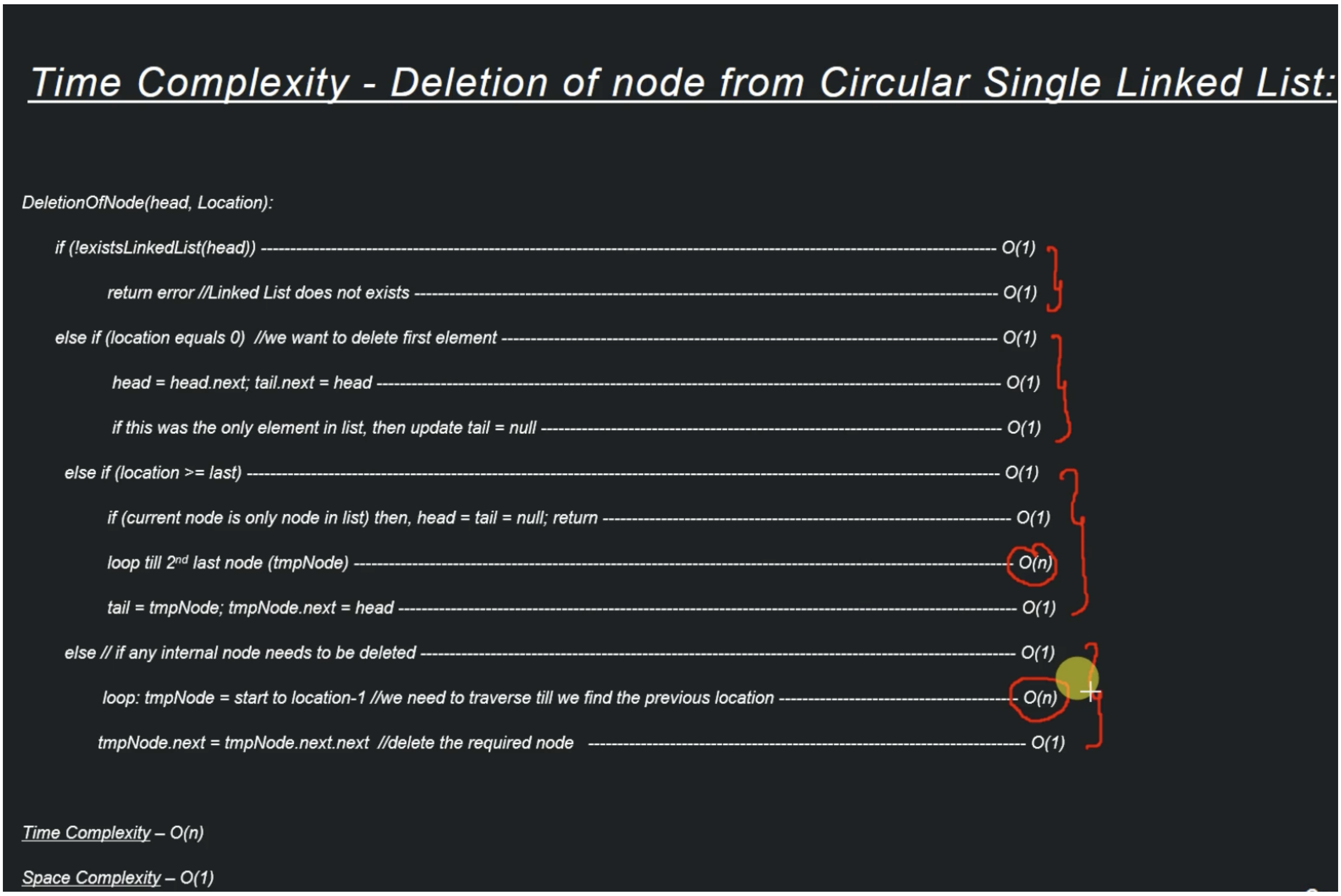
Time Complexity – O(n)

Space Complexity – O(1)



Deletion of node from Circular Single 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
        head = head.next; tail.next = head
        if this was the only element in list, then update head = tail = node.next = null;
    else if (location >= last)
        if (current node is only node in list) then, head = tail = node.next = null; return;
        loop till 2nd last node (tmpNode)
        tail = tmpNode; tmpNode.next = head;
    else // if any internal node needs to be deleted
        loop: tmpNode = start to location-1 //we need to traverse till we find the previous location
        tmpNode.next = tmpNode.next.next  //delete the required node
```

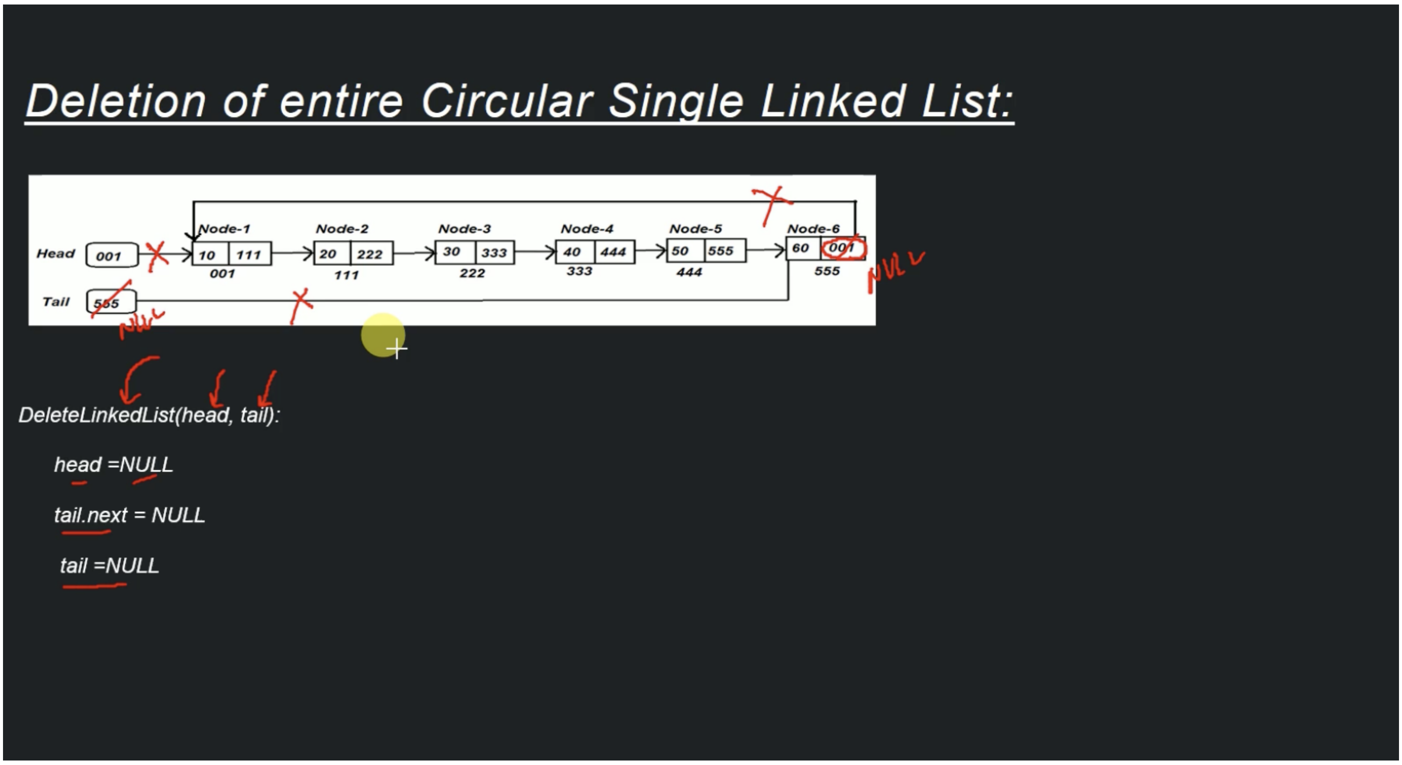


Time Complexity - Deletion of node from Circular Single Linked List:

```
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)
            head = head.next; tail.next = head ---------------------------------------------------------------------- O(1)
            if this was the only element in list, then update tail = null ----------------------------------------- O(1)
      else if (location >= last) ----------------------------------------------------------------------------------------- O(1)
            if (current node is only node in list) then, head = tail = null; return -------------------------------- O(1)
            loop till 2nd last node (tmpNode) ----------------------------------------------------------------------- O(n)
            tail = tmpNode; tmpNode.next = head ------------------------------------------------------------------ O(1)
      else // if any internal node needs to be deleted -------------------------------------------------------------- O(1)
            loop: tmpNode = start 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)
```

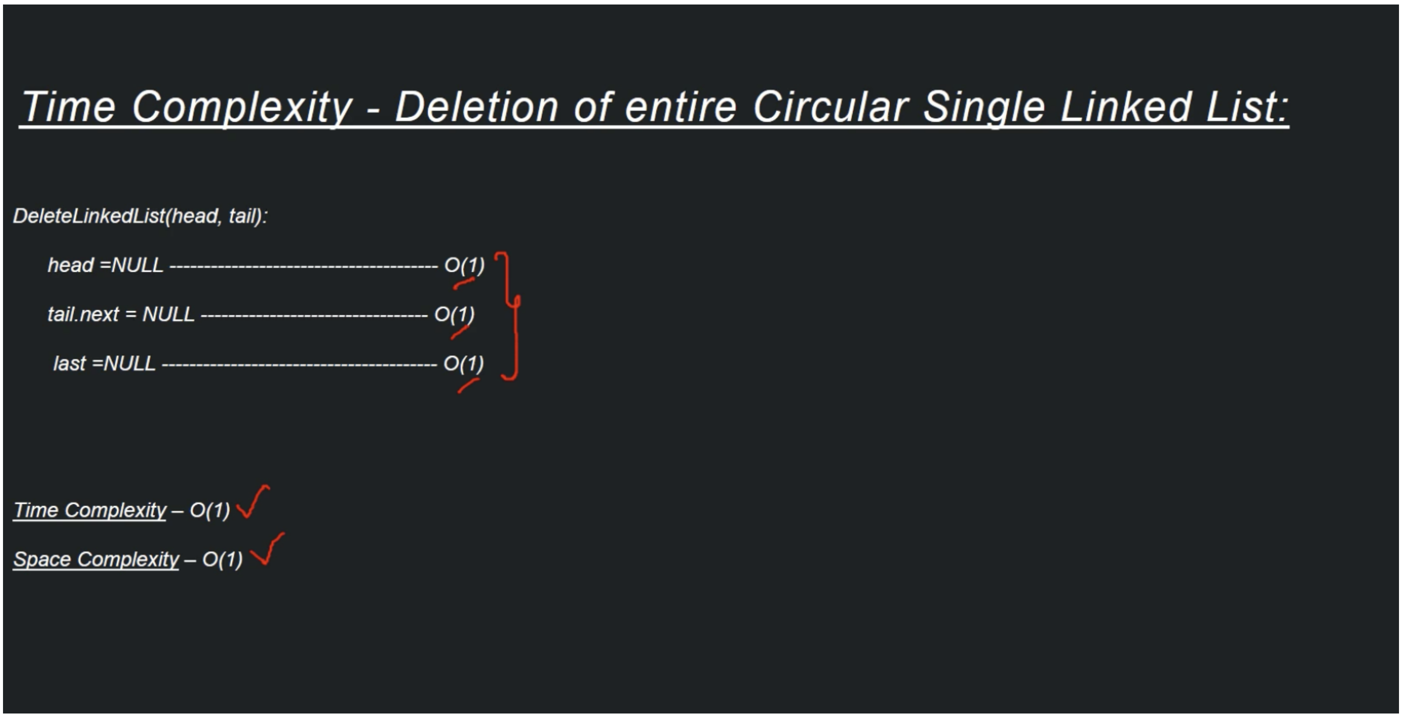
Time Complexity – O(n)

Space Complexity – O(1)



Deletion of entire Circular Single Linked List:

```
DeleteLinkedList(head, tail):
    head =NULL
    tail.next = NULL
    tail =NULL
```

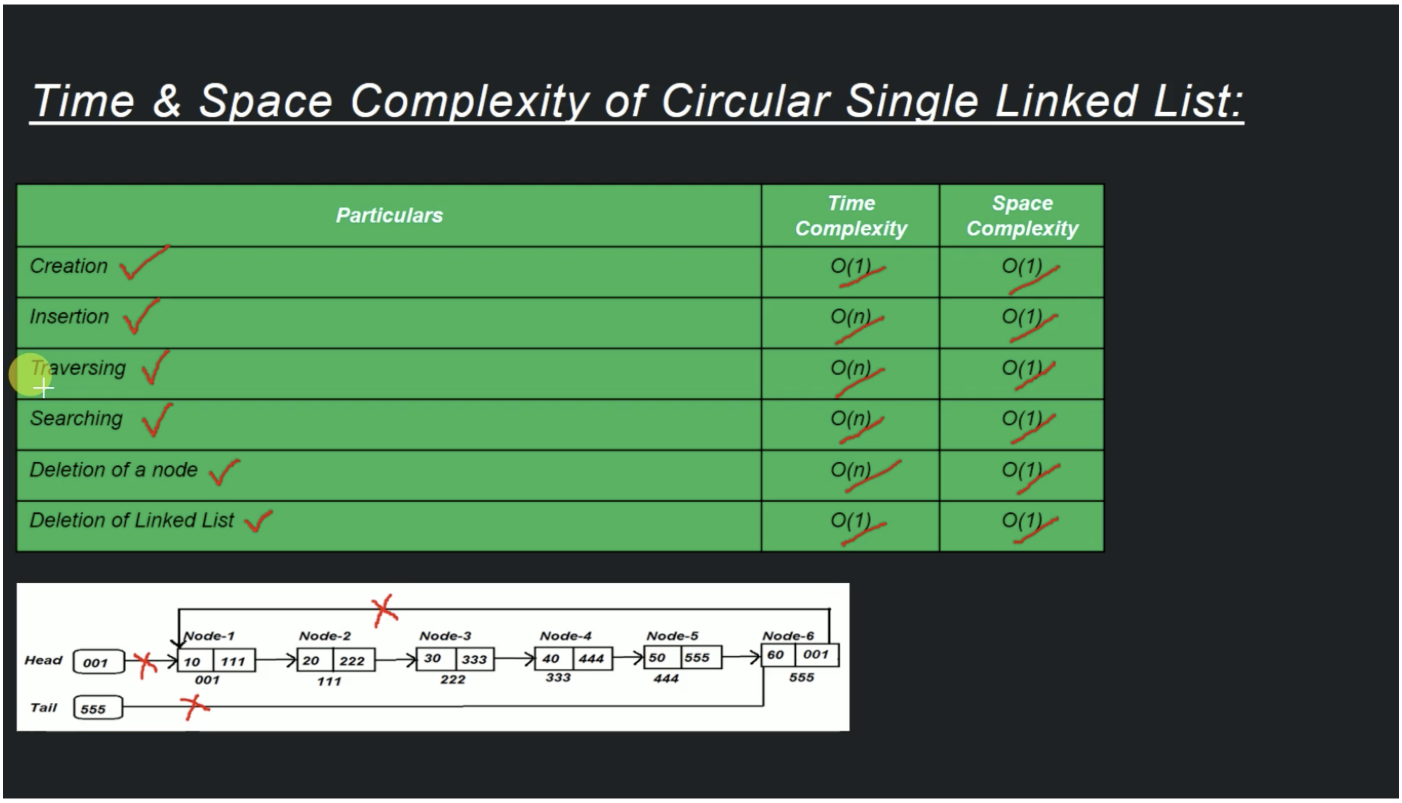


Time Complexity - Deletion of entire Circular Single Linked List:

```
DeleteLinkedList(head, tail):
     head =NULL ---------------------------------------- O(1)
     tail.next = NULL -------------------------------- O(1)
      last =NULL ----------------------------------------- O(1)
```

Time Complexity – O(1) ✓

Space Complexity – O(1) ✓



Time & Space Complexity of Circular Single Linked List:

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