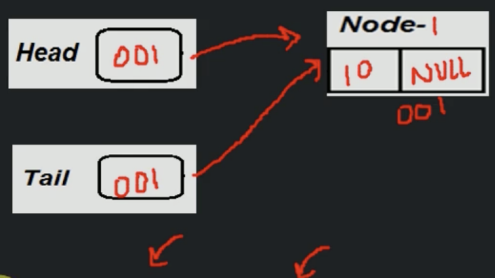
Creation of Single Linked List:



CreateSingleLinkedList(nodeValue):

1. create a head, tail pointer and initialize with NULL
2. create a blank node
3. node.value = nodeValue;
4. node.next = null;
5. head = node;
6. tail = node;

Time Complexity - Creation of Single Linked List:

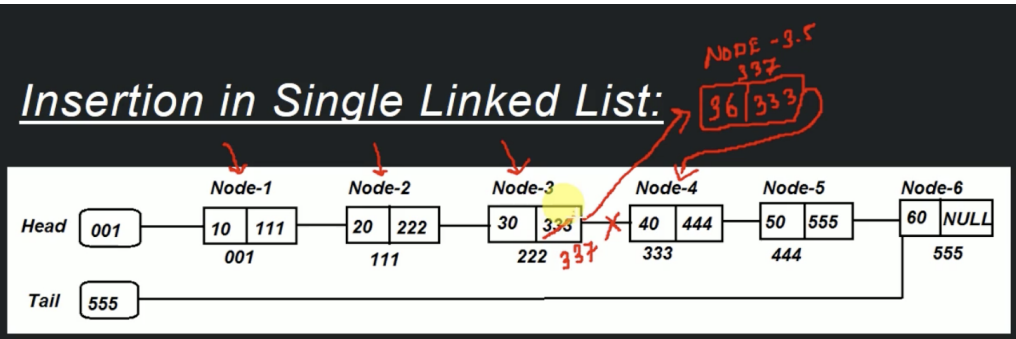
CreateSingleLinkedList(nodeValue):

- *create a head, tail pointer ---------- O(1)*
- *create a blank node ---------- O(1)*
- *node.value = nodeValue ---------- O(1)*
- *node.next = null ---------- O(1)*
- *head = node ---------- O(1)*
- *tail = node ---------- O(1)*

Time Complexity – O(1)

Space Complexity – O(1)

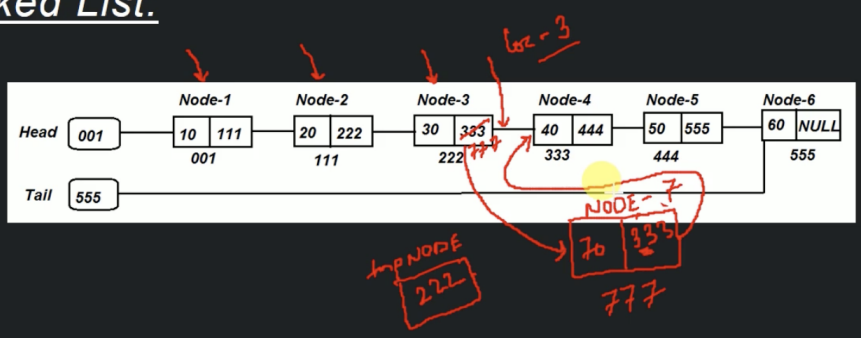
Insertion in Single Linked List:



- ✓ There can be 3 cases:
  - ✓ Insert at start of Linked List
  - ✓ Insert at end of Linked List
  - ✓ Insert at a specified Location in Linked List

Insertion in 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;
  else if (location equals last) //insert at last position
       node.next = null;
       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
```



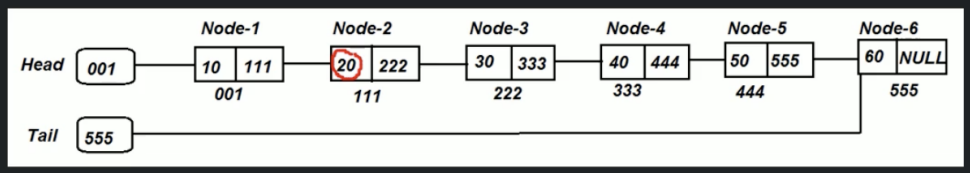
Time Complexity - Insertion in Single Linked List:

```
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)
    else if (location equals last) //insert at last position ---------- O(1)
        node.next = null ---------- O(1)
        last.next = node ---------- O(1)
        last = 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 Single Linked List:




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

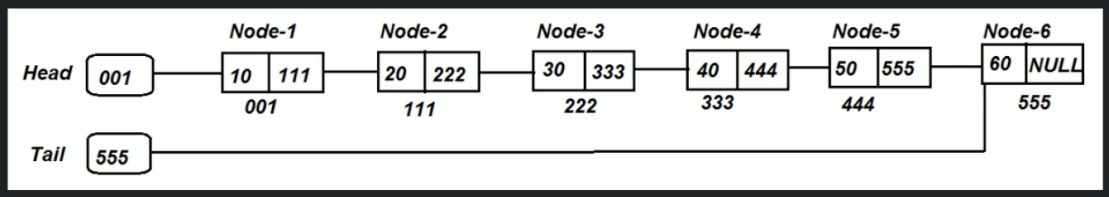
Time Complexity - Traversal of Single Linked List:

```
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)

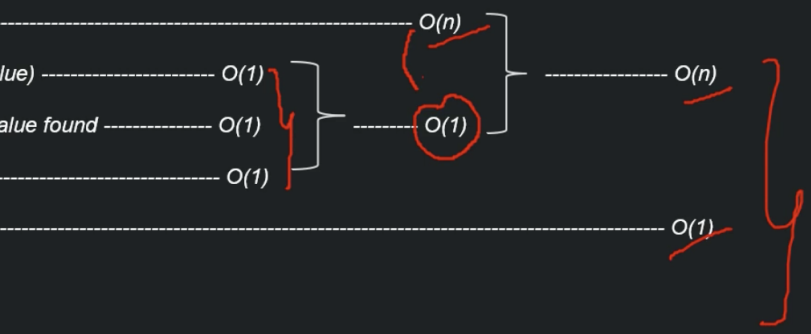
Searching a node in Single Linked List:




```
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 Single Linked List:

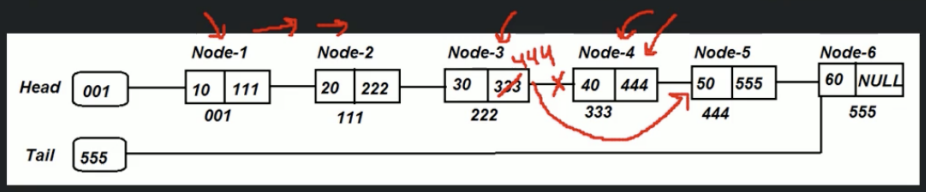
```
SearchNode(head, nodeValue):
    loop: tmpNode = start to tail ---------- O(n)
        if (tmpNode.value equals nodeValue) ---------- O(1)
            print tmpNode.Value //node value found ---------- O(1)
            return ---------- O(1)
    return //nodeValue not found ---------- O(1)
```



Time Complexity – O(n)

Space Complexity – O(1)

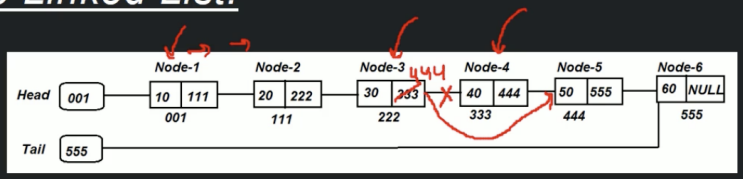
Deletion of node from Single Linked List:



- ✓ There can be 3 cases:
  - ✓ Delete first element
  - ✓ Delete last node
  - ✓ Delete any node apart from above 2

Deletion of node from 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 node
        head = head.next;
        if this was the only element in list, then update tail = null;
    else if (location >= last)
        if (current node is only node in list) then, head = tail = null; return;
        loop till 2nd last node (tmpNode)
        tail = tmpNode; tmpNode.next = null;
    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
```



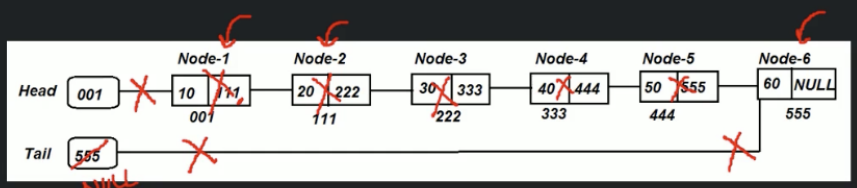
Time Complexity - Deletion of node from Single Linked List:

```
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 node ---------- O(1)
        head = head.next ---------- 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 = null ---------- O(1)
    else // if any internal node needs to be deleted ---------- O(1)
        loop: tmpNode = start to location-1 ---------- O(n)
        tmpNode.next = tmpNode.next.next  //delete the required node ---------- O(1)
```

Time Complexity – O(n)

Space Complexity – O(1)

Deletion of entire Single Linked List:



DeleteLinkedList(head, tail):

head =null

tail = null

Time Complexity - Deletion of entire Single Linked List:

DeleteLinkedList(head, tail):

head =null ---------- O(1)

tail = null ---------- O(1)

Time Complexity – O(1) ✓

Space Complexity – O(1) ✓

Time & Space Complexity of Single Linked List:

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