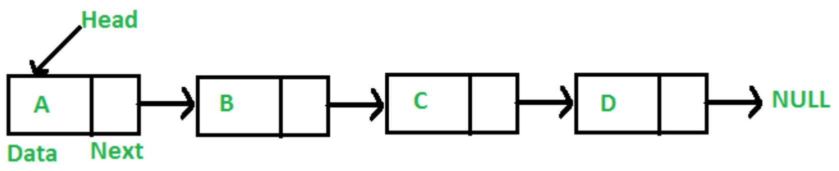
Linked List



Python Code for Singly Linked List:

```
# Define a class named Node representing a single node in a singly linked list.
class Node:
        #  method for the Node class. Node will have two attributes: data and next.
    def __init__(self, data):
          # Initialize the data attribute of the node with the given data.
        self.data = data
          # Initialize the next attribute of the node as None, indicating that it initially does not point to any other node as its empty.
        self.next = None

# Define a class named SinglyLinkedList representing a singly linked list.
class SinglyLinkedList:
     # Define the constructor method for the SinglyLinkedList class.
    def __init__(self):
         # Initialize the head attribute of the linked list as None, indicating an empty list.
        self.head = None

     # Define a method named append to add a new node with the given data to the end of the linked list.
    def append(self, data):
        # Create a new node object with the given data.
        new_node = Node(data)
        # Check if the linked list is empty.
```

```python
        if not self.head:
             # If the linked list is empty, set the new node as the head of the linked list and return.
            self.head = new_node
            return
         # If the linked list is not empty, traverse the linked list to find the last node.
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
         # Once the last node is found, set the next attribute of the last node to the new node, effectively adding it to the end of the linked list.
        last_node.next = new_node

     # Define a method named display to print the elements of the linked list.
    def display(self):
        # Start from the head of the linked list.
        current = self.head
         # Traverse the linked list and print the data of each node.
        while current:
            print(current.data, end=" -> ")
            # Move to the next node in the linked list.
            current = current.next
        # Print "None" to signify the end of the linked list.
        print("None")

# Example usage:
# Create an instance of the SinglyLinkedList class.
sll = SinglyLinkedList()
# Append some elements to the linked list.
sll.append(1)
sll.append(2)
sll.append(3)
# Display the elements of the linked list.
sll.display()
```

Sample Solution of Leetcode problems:

1. **Reversed Linked List** : leetcode.com/problems/reverse-linked-list/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

class Solution(object):
    # The function to reverse a linked list. It takes the head of the list as input.
    def reverseList(self, head):
         prev = None  # Initialize a pointer to None. This will eventually become the new tail of the list.
         curr = head  # Initialize another pointer to the head of the list, to start traversing the list.

         # Loop until curr is None, which means we've reached the end of the list.
        while curr:
             next_node = curr.next  # Store the next node temporarily since we're about to change curr.next.

             curr.next = prev  # Reverse the link: point the current node's next pointer to the previous node.

             prev = curr  # Move the prev pointer forward: it now points to the current node.

            curr = next_node  # Move the curr pointer forward: it now points to the next node in the original list.

        # At the end of the loop, curr is None (we've reached the end of the list),
         # and prev is the last node we processed, which is the new head of the reversed list.
        return prev  # Return the new head of the reversed list.
```

Complexity Analysis:
Time Complexity: O(n), as the function iterates through the entire linked list once.
Space Complexity: O(1), as it uses a fixed number of variables (prev, curr, next_node).

2. **Merge Two Sorted Lists :**
   leetcode.com/problems/merge-two-sorted-lists/description/

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def mergeTwoLists(self, list1, list2):
        # Initialize a dummy node with value -1.
        # This dummy node serves as the head of the merged list.
        dummy = ListNode(-1)

        # Initialize a pointer 'current' to the dummy node.
        # 'current' will be used to build the merged list.
        current = dummy

        # Loop until both list1 and list2 have nodes to merge.
        while list1 and list2:
            # Compare the values of the current nodes in list1 and list2.
            if list1.val < list2.val:
                    # If the value of the current node in list1 is smaller,
attach it to 'current.next'.
                current.next = list1
                # Move the pointer of list1 to its next node.
                list1 = list1.next
            else:
                  # If the value of the current node in list2 is smaller or
equal, attach it to 'current.next'.
                current.next = list2
                # Move the pointer of list2 to its next node.
                list2 = list2.next

               # Move the 'current' pointer to the next node in the merged
list.
            current = current.next

         # If list1 still has remaining nodes, attach them to the end of the
merged list.
        if list1:
            current.next = list1
         # If list2 still has remaining nodes, attach them to the end of the
merged list.
        elif list2:
            current.next = list2

         # Return the head of the merged list, which is the next node after
the dummy node.
        return dummy.next
```

Complexity Analysis:
Time Complexity: O(m+n), where m and n are the lengths of the input lists list1 and list2, respectively.
Space Complexity: O(1), as it creates a dummy node (dummy) and a current pointer (current) to build the merged list in place

3. **Remove Nth Node From End of List :**
leetcode.com/problems/remove-nth-node-from-end-of-list/

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def removeNthFromEnd(self, head, n):
         # Initialize a dummy node with value 0 and point its next to the
head of the original list.
        dummy = ListNode(0, head)

        # Initialize two pointers 'slow' and 'fast' to the dummy node.
        # Both pointers will be used to find the node to be removed.
        slow = fast = dummy

        # Move the 'fast' pointer n steps ahead of the 'slow' pointer.
        for i in range(n):
            fast = fast.next

          # Move both 'slow' and 'fast' pointers until 'fast' reaches the
last node.
          # At this point, 'slow' will be pointing to the node before the
one to be removed.
        while fast.next:
            slow = slow.next
            fast = fast.next
        # Remove the Nth node from the end by updating 'slow.next' to skip
the next node.
        slow.next = slow.next.next

         # Return the head of the modified list (next node after the dummy
node).
        return dummy.next
```

Complexity Analysis:

Time Complexity: O(n), In the worst case scenario the for loop has to iterate till the last element, i,e, when the first node of the given linked list is to be removed.

Space Complexity: O(1), as it creates a dummy node (dummy) and two pointers (slow and fast) to traverse the list