Recursive Version

Thinking naively, we can reverse all elements except the head recursively and then put head in its correct place (i.e at the end of the reversed sub list). However this approach has very bad time complexity (Why?). Because in each recursive call, after the sub-list has been reversed, you'll end up travelling the entire list to get to the last element (in order to put *head* in its correct position). Upon further analysis, we find that it takes up $1 + 2 + 3 + ...n = n(n + 1)/2$ time which is $O(n^2)$.

How do we improve? Clearly the only thing which leads to this huge time complexity is the fact that we do not have access to the last element of the reversed list. But, think carefully, what would the last element of the reversed sub-list be? It would actually be the second element of the original list. Do we have access to it in constant time? Yes, we can easily access it via *head.next* as we haven't altered *head* yet. So now, we access the last element of the reversed sub-list in $O(1)$ and connect it to the first element of the original list,(its correct place in the new list).

We have to make sure that we block the access of the *head* (which was referencing to the last element of the reversed sub-list). Since *head* is now the last element, *head.next* would be *null*. Remember to return the new reversed head obtained from the recursive call.

Algorithm Reverse a Linked List recursively

```
1: function Reverse_List(head)
2:     if List is of length 0 or 1 then
3:         return head
4:     reversed_sub_list_head ← Reverse_List(head.next)
5:     sub_list_tail ← head.next
6:     sub_list_tail.next ← head
7:     head.next ← null
8:     return reversed_sub_list_head
```

Iterative Version

Algorithm Reverse a Linked List iteratively

```
1: function Reverse_List(head)
2:     previous ← null
3:     current ← head
4:     while current exists, do
5:         backup ← current.next
6:         current.next ← previous
7:         previous ← current
8:         current ← backup
9:     return previous                                  ▷ current is null
```