**Question**

Reverse a singly linked list.

[click to show more hints.](https://leetcode.com/articles/reverse-linked-list/)

**Hint:**

A linked list can be reversed either iteratively or recursively. Could you implement both?

**Quick Navigation**

* [Solution](https://leetcode.com/articles/reverse-linked-list/#solution)
  + [Approach #1 (Iterative) [Accepted]](https://leetcode.com/articles/reverse-linked-list/#approach-1-iterative-accepted)
  + [Approach #2 (Recursive) [Accepted]](https://leetcode.com/articles/reverse-linked-list/#approach-2-recursive-accepted)

**Solution**

**Approach #1 (Iterative) [Accepted]**

Assume that we have linked list 1 → 2 → 3 → Ø, we would like to change it to Ø ← 1 ← 2 ← 3.

While you are traversing the list, change the current node's next pointer to point to its previous element. Since a node does not have reference to its previous node, you must store its previous element beforehand. You also need another pointer to store the next node before changing the reference. Do not forget to return the new head reference at the end!

public ListNode reverseList(ListNode head) {

ListNode prev = null;

ListNode curr = head;

while (curr != null) {

ListNode nextTemp = curr.next;

curr.next = prev;

prev = curr;

curr = nextTemp;

}

return prev;

}

**Complexity analysis**

* Time complexity : O(n)O(n)O(n). Assume that nnn is the list's length, the time complexity is O(n)O(n)O(n).
* Space complexity : O(1)O(1)O(1).

**Approach #2 (Recursive) [Accepted]**

The recursive version is slightly trickier and the key is to work backwards. Assume that the rest of the list had already been reversed, now how do I reverse the front part? Let's assume the list is: n1 → … → nk-1 → nk → nk+1 → … → nm → Ø

Assume from node nk+1 to nm had been reversed and you are at node nk.

n1 → … → nk-1 → **nk** → nk+1 ← … ← nm

We want nk+1’s next node to point to nk.

So,

nk.next.next = nk;

Be very careful that n1's next must point to Ø. If you forget about this, your linked list has a cycle in it. This bug could be caught if you test your code with a linked list of size 2.

public ListNode reverseList(ListNode head) {

if (head == null || head.next == null) return head;

ListNode p = reverseList(head.next);

head.next.next = head;

head.next = null;

return p;

}

**Complexity analysis**

* Time complexity : O(n)O(n)O(n). Assume that nnn is the list's length, the time complexity is O(n)O(n)O(n).
* Space complexity : O(n)O(n)O(n). The extra space comes from implicit stack space due to recursion. The recursion could go up to nnn levels deep.