**206. Reverse Linked List**

<https://leetcode.com/problems/reverse-linked-list/>

1. **Listen**

**Problem Statement:** Given the **head** of a **singly linked list**, **reverse** the list, and return *the reversed list*.

**Goal**: Reverse singly linked list

**Input**: head of a singly linked list

**Return**: head of reversed list

1. **Example**

Diagram

Description automatically generated

**Input:** head = [1,2,3,4,5]

**Output:** [5,4,3,2,1]

**Example 2:**

A picture containing text, clipart

Description automatically generated

**Input:** head = [1,2]

**Output:** [2,1]

**Constraints:**

* The number of nodes in the list is in range [0, 5000]
* -5000 <= Node.val <= 5000

**Test Cases:**

* Odd Length Linked List
* Even Length Linked List

**Edge Cases:**

* Empty Linked List
* Single Node in Linked List

1. **Brute Force**

Solution 1: Iterative

Iteratively traverse the list with two pointers, flipping the *next* pointers of each node to the previous node as we go along.

Initialize two pointers:

*previous as NULL*

*current as head*

To allow the algorithm to begin naturally, we need to initialize the previous (p) pointer as null.

This is because since we are completely reversing the list, the very first node’s next pointer will now point to null, as opposed to the very last node’s next pointer pointing to null.

While the *current* pointer is not null.

* 1. We save *current.next* as a *temp* pointer to to the next node in line because we will be cutting off the connection to the next node when we set current.next to the previous node (the act of reversing).
  2. Set *current.next* to *previous* (this is the actually act of reversing).
  3. Now we move the pointers up. *Previous* goes to *current*, and *current* goes to *temp*

Once the loop ends, *previous* should now be set to the last node in the list, which is also the new *head*.

Return *previous*.

The while loop condition is based off of the current *c* pointer because the *p* pointer has no way of reaching the end of the list before *c*.

We can accomplish this in O(N) time and O(1) space.

1. **Optimize**

We could shorten the code by making the algorithm recursive, but we would be increasing the space to O(N) while the runtime would stay at (N).

1. **Implement**

Iterative Solution:

private ListNode reverseListIterative(ListNode head)

{

ListNode p = null;

ListNode c = head;

while(c != null)

{

ListNode n = c.next;

c.next = p;

p = c;

c = n;

}

return p;

}

Recursive Solution:

The recursive solution is pretty much identical, except we pass the current and previous pointers as arguments to a helper function.

private ListNode reverseListRecursive(ListNode head)

{

ListNode p = null;

return reverseHelper(p, head);

}

private ListNode reverseHelper(ListNode p, ListNode c)

{

if(c == null) return p;

ListNode n = c.next;

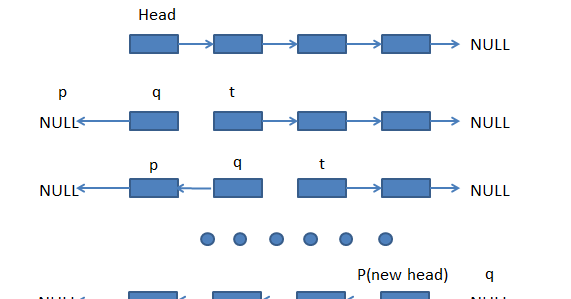
c.next = p;

p = c;

c = n;

return reverseHelper(p, c);

}



1. **Test**

* Odd Length Linked List
* Even Length Linked List
* Empty Linked List
* Single Node in Linked List