0001\_Easy\_ReverseLinkedList\_#206\_Breakdown

Problem:

Given the head of a singly linked list, reverse the list, and return the reversed list.

Examples:

**Example 1:**

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]

**Example 3:**

**Input:** head = []

**Output:** []

What needs to be true for this problem to work:

* The returned list needs to be a reversed version of the original list.

How would a person solve this problem:

* Look at the end of the list, make it the head of the new list, make the second to the last value the next value, repeat.
* Make the first value the end value of a new list, make the next value the previous value to the new list, repeat.

Brute Force:

* Create a counter, iterate through the linked list until the end is found, make a new linked list the end value the head value. Decrement the counter, iterate through the original list until the place of the counter is reached, make this value the next value on the new list. Repeat.
* Time Complexity: O(n^2), for each element in the linked list, the linked list is iterated through.
  + Specifically O(n(n-1)/2), the subsequent iterations get smaller.
* Space Complexity: O(1), the iterations happen in place.

Optimize (BUD, bottlenecks, unnecessary work, duplicated work):

* Instead of repeatedly iterating through the linked list, the linked list can be iterated through once, building the new linked list from the end first.
* Time Complexity: O(n), the linked list is iterated through once.
* Space Complexity: O(1), no extra space is needed.

Pseudocode:

* Create a node called reversed for the start of the reversed list (which will become the end).
* Create a node called current to iterate through the list and set it to point to the start of the original list.
* While the end of the original list hasn’t been reached.
  + Create a temporary node to point to the node after current to be used as a place holder.
  + Set the next node of current to point to reversed (this builds the list backwards while not overwriting the original lists order).
  + Set the reversed node to point to current to move backwards in the list.
  + Set the current node to point to the temporary node start at what is the new head of the original list
* Return the node reversed.

Recursive:

* Determine the base case,
  + If head is null (empty list) or the next value to head is null, this is the end of the list,
  + So return head.
* Call the reverseList function, with an input of the next node after head, this grows linearly, as when the end of the list is reached, the current input (depending on the size of the list would be something like the next next next next next node to head.
* Now that the end of the original list has been reached, the list needs to be reversed.
* The end of the original list is returned as head, the stack then moves to the next call, so head is now the second to last item of the original list.
* To point backwards, the current head (the second to last item) points to the last item and sets that point to itself, so now the last item is point to the second to last item.
* The current head is still pointing to the last item though, which is the wrong direction, so set the current head to point to null.
* This occurs with each stack call as the function works its way back.
* Time Complexity: O(n), the linked list is iterated through once.
* Space Complexity: O(n), stack calls

Pseudocode:

* Establish a base case, if the current head or next node is null, return head.
* Create a new Listnode called reversed to call the function, with the input value as the node after head.
* Set the next, next, value of head to head.
* Set the next value of head to null.
* Return the node reversed.