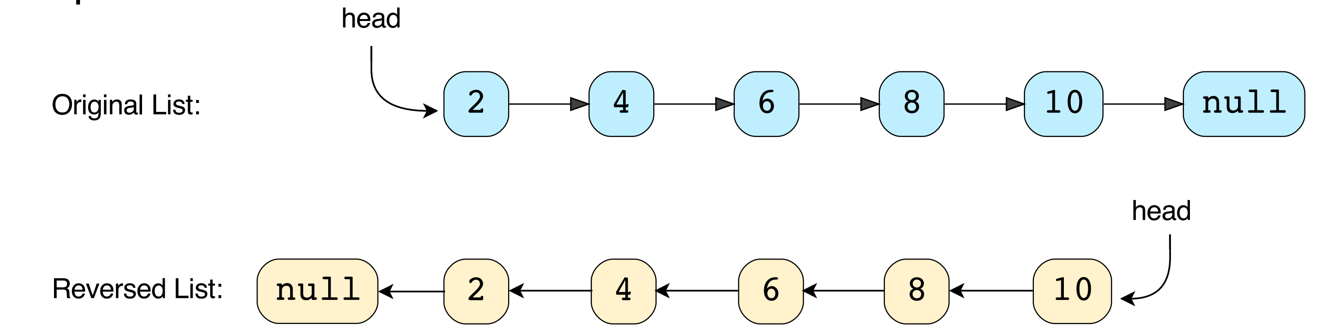
# Reverse a LinkedList (easy)

**We'll cover the following**

## Problem Statement #

Given the head of a Singly LinkedList, reverse the LinkedList. Write a function to return the new head of the reversed LinkedList.

**Example:**



## Try it yourself #

Try solving this question here:

Java | Python3 | JS | C++

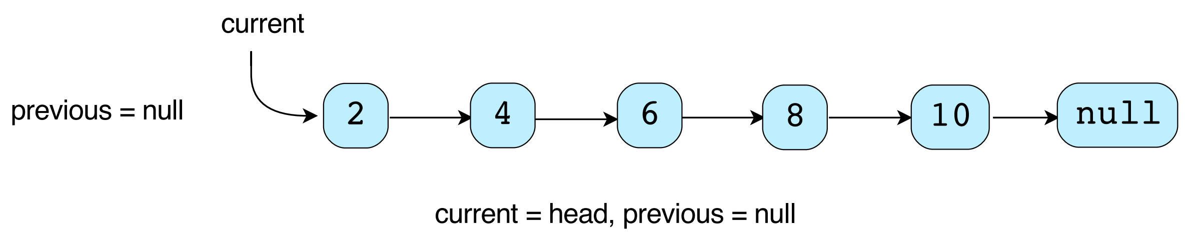
```
class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class ReverseLinkedList {

  public static ListNode reverse(ListNode head) {
    // TODO: Write your code here

    ListNode prev = null;
    ListNode dummyHead = head;
    ListNode next = null;

    while(dummyHead!=null){
          next = dummyHead.next;
          dummyHead.next = prev;
          prev = dummyHead;
          dummyHead=next;
        }

        return prev;
      }

```

Run Save Reset

## Solution #

To reverse a LinkedList, we need to reverse one node at a time. We will start with a variable `current` which will initially point to the head of the LinkedList and a variable `previous` which will point to the previous node that we have processed; initially `previous` will point to `null`.

In a stepwise manner, we will reverse the `current` node by pointing it to the `previous` before moving on to the next node. Also, we will update the `previous` to always point to the previous node that we have processed. Here is the visual representation of our algorithm:



**1** of 7

## Code #

Here is what our algorithm will look like:

| Java | Python3 | C++ | JS |
|---|---|---|---|

```java
class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class ReverseLinkedList {

  public static ListNode reverse(ListNode head) {
    ListNode current = head; // current node that we will be processing
    ListNode previous = null; // previous node that we have processed
    ListNode next = null; // will be used to temporarily store the next node

    while (current != null) {
      next = current.next; // temporarily store the next node
      current.next = previous; // reverse the current node
      previous = current; // before we move to the next node, point previous to
      current = next; // move on the next node
    }
    // after the loop current will be pointing to 'null' and 'previous' will be
    return previous;
  }

  public static void main(String[] args) {
    ListNode head = new ListNode(2);
    head.next = new ListNode(4);
    head.next.next = new ListNode(6);
    head.next.next.next = new ListNode(8);
    head.next.next.next.next = new ListNode(10);

    ListNode result = ReverseLinkedList.reverse(head);
    System.out.print("Nodes of the reversed LinkedList are: ");
    while (result != null) {
      System.out.print(result.value + " ");
      result = result.next;
    }
  }
}
```

Run Save Reset

## Time complexity #

The time complexity of our algorithm will be $O(N)$ where 'N' is the total number of nodes in the LinkedList.

## Space complexity #

We only used constant space, therefore, the space complexity of our algorithm is $O(1)$.

← Back
Introduction

Next →
Reverse a Sub-list (medium)

✓ Completed

Report an Issue