Reverse a Singly-Linked List

Example:

Input: 

Output: 

- size is unknown

Iterative solution

```
function reverseList(head){
    if (head === null) { return null; }

    let currentNode = head.next;
    let reversedListHead = head;
    reversedListHead.next = null;

    while( currentNode !== null ){
        let reversedCurrentNode = currentNode;
        currentNode = currentNode.next;
        reversedCurrentNode.next = reversedListHead;
        reversedListHead = reversedCurrentNode;
    }

    return reversedListHead;
}
```

Implementation Explanation:

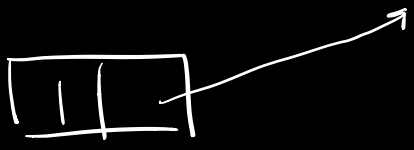
reversedListHead always points to the head of the new, reversed linked list, which changes as we iterate. since we add to the beginning of this linked list, we continue adding until the iteration is finished.

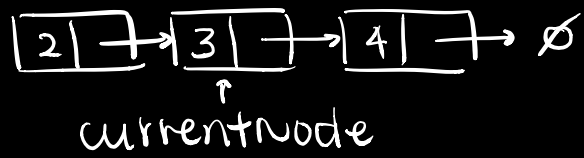
Complexity [O(n) time
O(1) space

Illustrated Test:

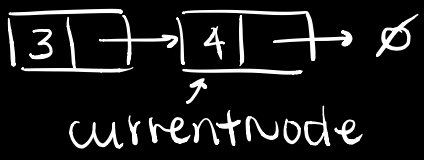
given: 

1) currentNode: 

reversedListHead: 

2) reversedCurrentNode = 

reversedCurrentNode =
reversedListHead = 

3) reversedCurrentNode = 

reversedCurrentNode =
reversedListHead = 

4) reversedCurrentNode = 

currentNode 

reversedCurrentNode =
reversedListHead = 

5) currentNode is null, return reversedListHead

—end—

Recursive Solution
O(n) time complexity
O(n) space complexity

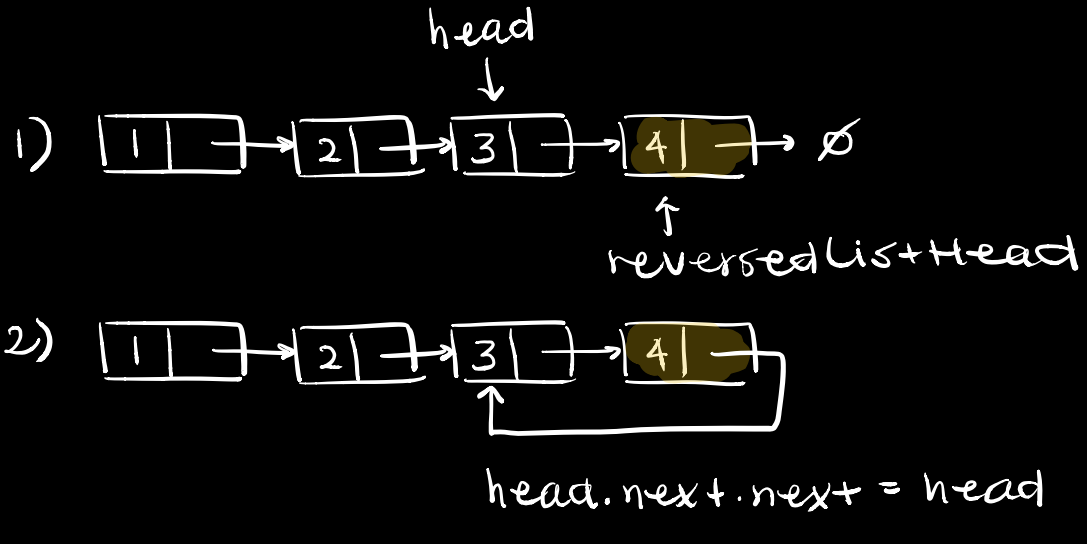
Recursive solution

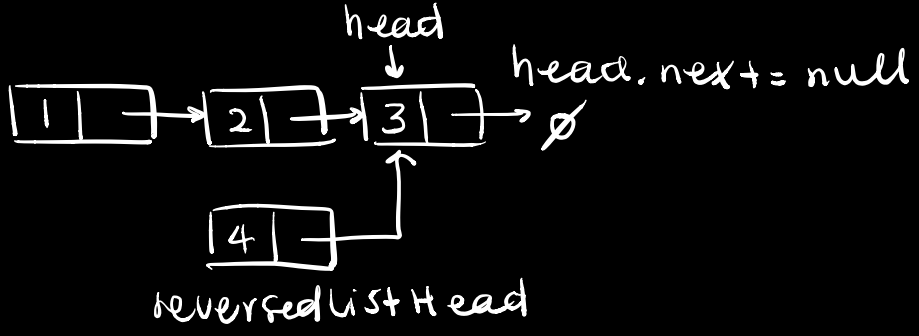
- Recursively visit each node until we reach the last node (which is the new head of the reversed LL)
- On each return, append the node to the end of the reversed linked list

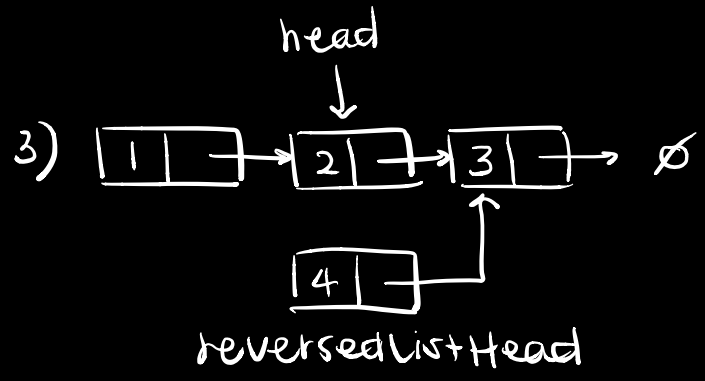
Implementation

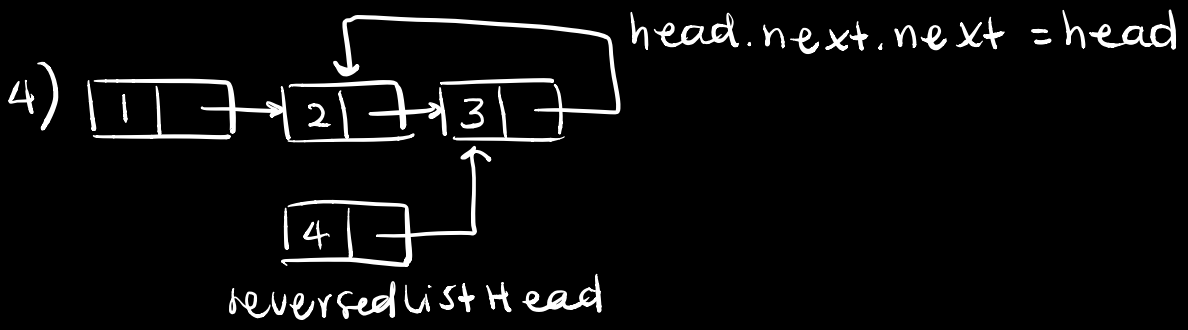
```
function reverseLinkedList(head){
    if (!head || !head.next) { return head; }
    let reversedListHead = reverseLinkedList(head.next);
    head.next.next = head;
    head.next = null;
    return reversedListHead;
}
```

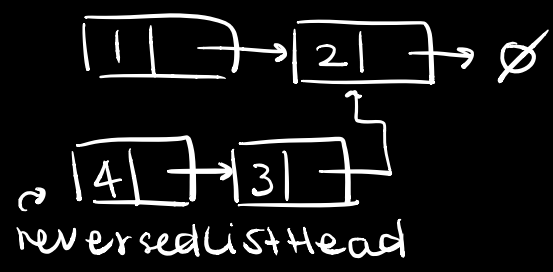
Illustrated

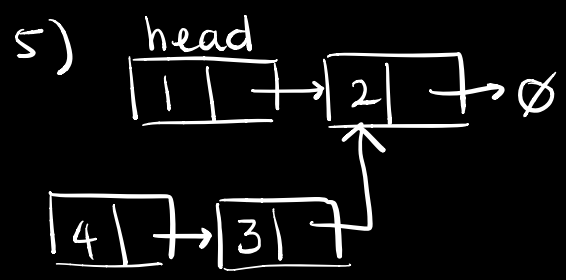


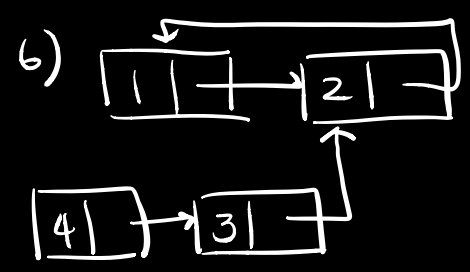


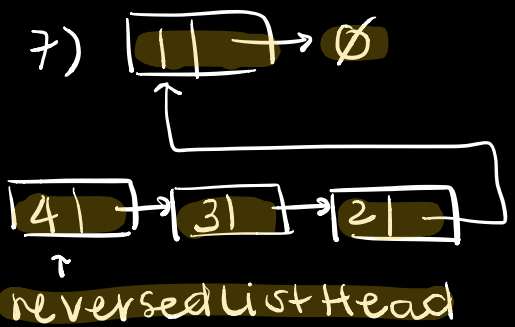

head
3)
1
2
3
4
reversedListHead


head.next.next = head
4)
1
2
3
4
reversedListHead


1
2
4
3
reversedListHead


5)
head
1
2
4
3


6)
1
2
4
3


7)
1
4
3
2
reversedListHead

- end -