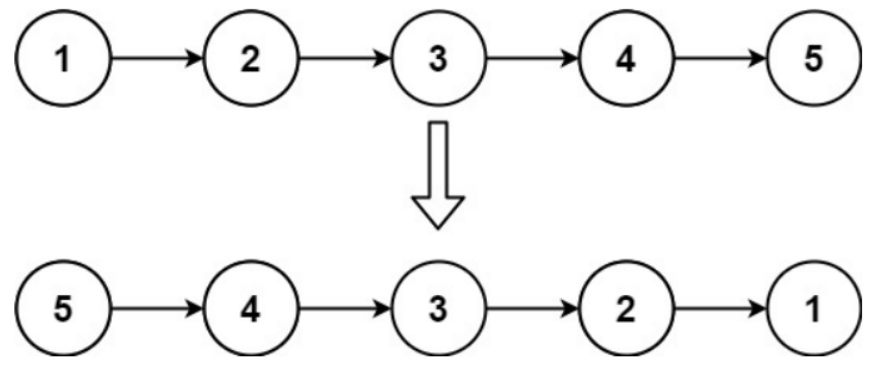
<https://leetcode.com/problems/reverse-linked-list/>

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

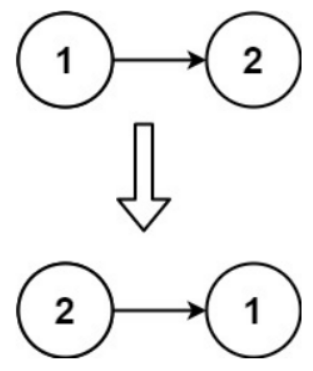
**Example 1:**



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

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

**Example 2:**



Input: head = [1,2]

Output: [2,1]

**Example 3:**

Input: head = []

Output: []

**Constraints:**

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

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

**Attempt 1: 2023-02-12**

**Solution 1:  Iterative Solution (10 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseList(ListNode head) {

if(head == null || head.next == null) {

return head;

}

ListNode prev = null;

ListNode cur = head;

while(cur != null) {

ListNode next = cur.next;

cur.next = prev;

prev = cur;

cur = next;

}

return prev;

}

}

**Solution 2:  Recursive Solution (10 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseList(ListNode head) {

if(head == null || head.next == null) {

return head;

}

return helper(head, null);

}

private ListNode helper(ListNode cur, ListNode prev) {

if(cur == null) {

return prev;

}

ListNode next = cur.next;

cur.next = prev;

return helper(next, cur);

}

}

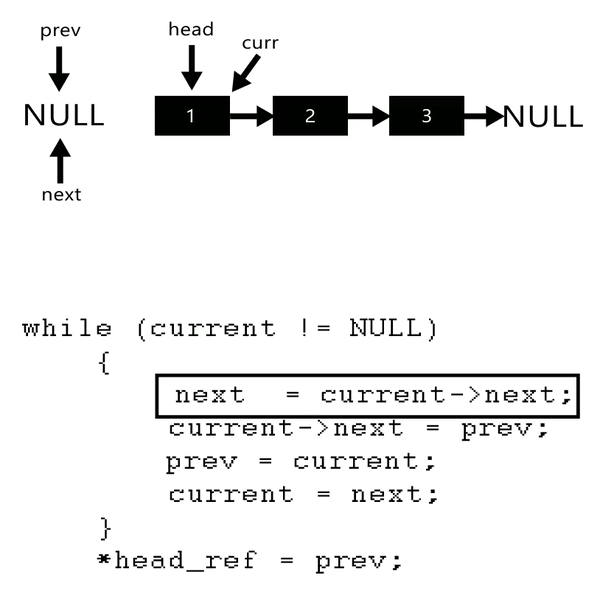
**Refer to**

<https://leetcode.com/problems/reverse-linked-list/solutions/803955/c-iterative-vs-recursive-solutions-compared-and-explained-99-time-85-space/>

Not sure how this problem is expecting me to use less memory than this, but here is the deal:

* we are going to use 3 variables: prevNode, head and nextNode, that you can easily guess what are meant to represent as we go;
* we will initialise prevNode to NULL, while nextNode can stay empty;
* we are then going to loop until our current main iterator (head) is truthy (ie: not NULL), which would imply we reached the end of the list;
* during the iteration, we first of all update nextNode so that it acquires its namesake value, the one of the next node indeed: head->next;
* we then proceeding "reversing" head->next and assigning it the value of prevNode, while prevNode will become take the current value of head;
* finally, we update head with the value we stored in nextNode and go on with the loop until we can. After the loop, we return prevNode.

I know it is complex, but I find this gif from another platform to make the whole logic much easier to understand (bear in mind we do not need curr and will just use head in its place):



The code:

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode \*nextNode, \*prevNode = NULL;

while (head) {

nextNode = head->next;

head->next = prevNode;

prevNode = head;

head = nextNode;

}

return prevNode;

}

};

Relatively trivial refactor (the function does basically the same) with recursion and comma operator to make it one-line:

class Solution {

public:

ListNode\* reverseList(ListNode \*head, ListNode \*nextNode = NULL, ListNode \*prevNode = NULL) {

return head ? reverseList(head->next, (head->next = prevNode, nextNode), head) : prevNode;

}

};

**Refer to**

<https://leetcode.com/problems/reverse-linked-list/solutions/58125/in-place-iterative-and-recursive-java-solution/>

We always put a node's previous node as one's next

Take 1 -> 2 -> 3 -> N for example, we reverse the list by

put 1's previous node null as 1's next,

put 2's previous node 1 as 2's next,

put 3's previous node 2 as 3's next,

return 3 // put null's previous node 3 as null's next

The code is as follows:

public ListNode reverseList(ListNode head) {

/\* iterative solution \*/

ListNode newHead = null;

while (head != null) {

ListNode next = head.next;

head.next = newHead;

newHead = head;

head = next;

}

return newHead;

}

public ListNode reverseList(ListNode head) {

/\* recursive solution \*/

return reverseListInt(head, null);

}

private ListNode reverseListInt(ListNode head, ListNode newHead) {

if (head == null)

return newHead;

ListNode next = head.next;

head.next = newHead;

return reverseListInt(next, head);

}