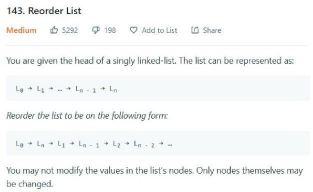
Reorder a linkedlist (FOLD).



143. Reorder List

Medium 5292 198 Add to List Share

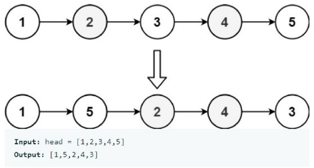
You are given the head of a singly linked-list. The list can be represented as:

```
L0 → L1 → … → Ln - 1 → Ln
```

Reorder the list to be on the following form:

```
L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
```

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

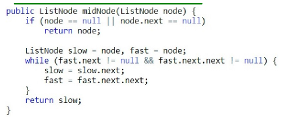


```
Input: head = [1,2,3,4,5]
Output: [1,5,2,4,3]
```

Steps.

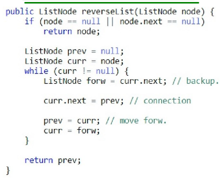
1. Find first mid. (fast.next != N & fast.next.next != N)
2. Reverse 2nd part.
3. Reordor two list.

1. Find first mid.



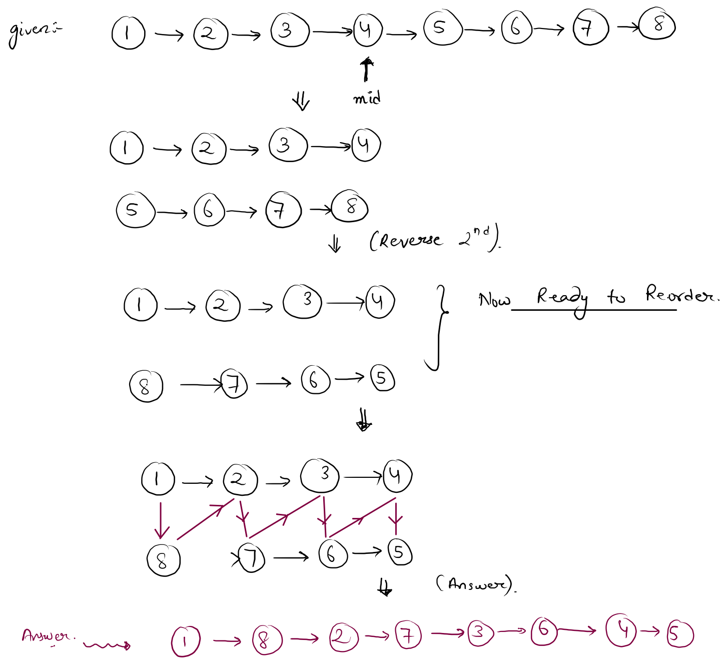
```
public ListNode midNode(ListNode node) {
    if (node == null || node.next == null)
        return node;

    ListNode slow = node, fast = node;
    while (fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}
```

2. Reverse 2nd part.

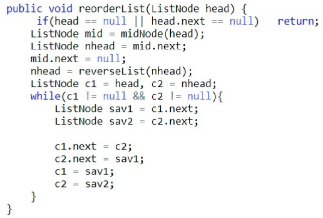


```
public ListNode reverseList(ListNode node) {
    if (node == null || node.next == null)
        return node;

    ListNode prev = null;
    ListNode curr = node;
    while (curr != null) {
        ListNode forw = curr.next; // backup.

        curr.next = prev; // connection

        prev = curr; // move forw.
        curr = forw;
    }

    return prev;
}
```

after step ① & ②

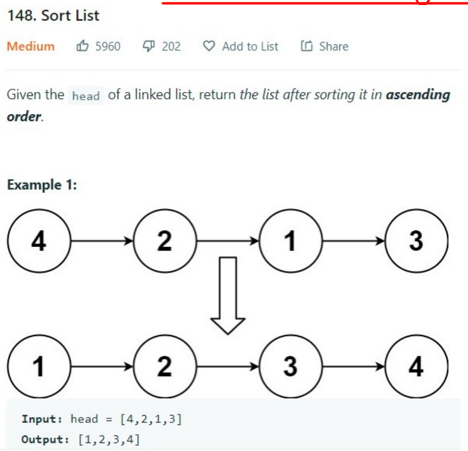


Step 3. Reorder.



```
public void reorderList(ListNode head) {
     if(head == null || head.next == null)   return;
    ListNode mid = midNode(head);
    ListNode nhead = mid.next;
    mid.next = null;
    nhead = reverseList(nhead);
    ListNode c1 = head, c2 = nhead;
    while(c1 != null && c2 != null){
        ListNode sav1 = c1.next;
        ListNode sav2 = c2.next;

        c1.next = c2;
        c2.next = sav1;
        c1 = sav1;
        c2 = sav2;
    }
}
```

Sort List (Merge Sort).



148. Sort List

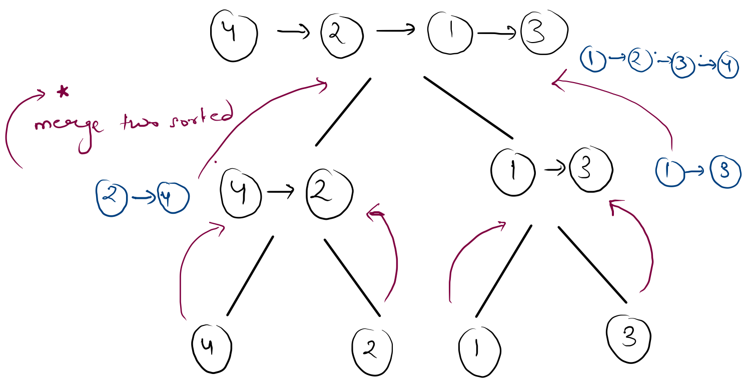
Medium 5960 202 Add to List Share

Given the `head` of a linked list, return *the list after sorting it in* ***ascending order***.

Example 1:

```
Input: head = [4,2,1,3]
Output: [1,2,3,4]
```

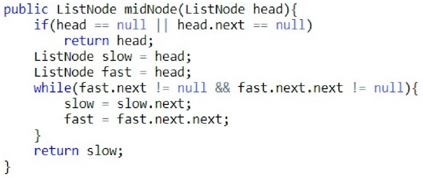
Recursive approach.



Steps. In every recursive state :

1. find mid. & new Head
2. break into 2 list
3. merge sort both list.
4. Return sorted list using merge 2 sorted.

1. find mid & new Head.

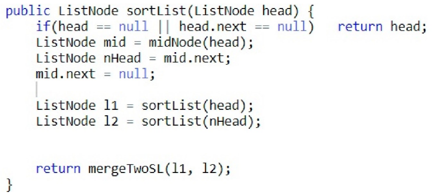


```
public ListNode midNode(ListNode head){
    if(head == null || head.next == null)
        return head;
    ListNode slow = head;
    ListNode fast = head;
    while(fast.next != null && fast.next.next != null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}
```

2. B eak. into 2 parts.

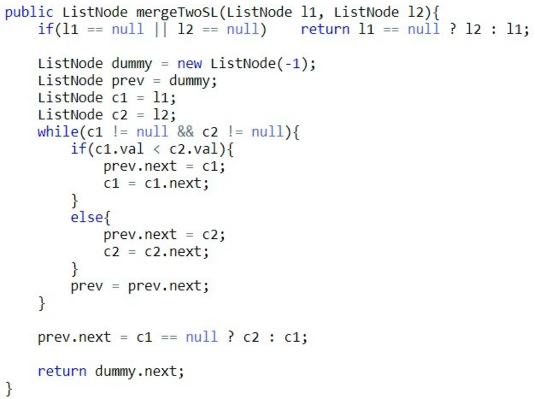
3. ge sort both list

4. rn sorted list.



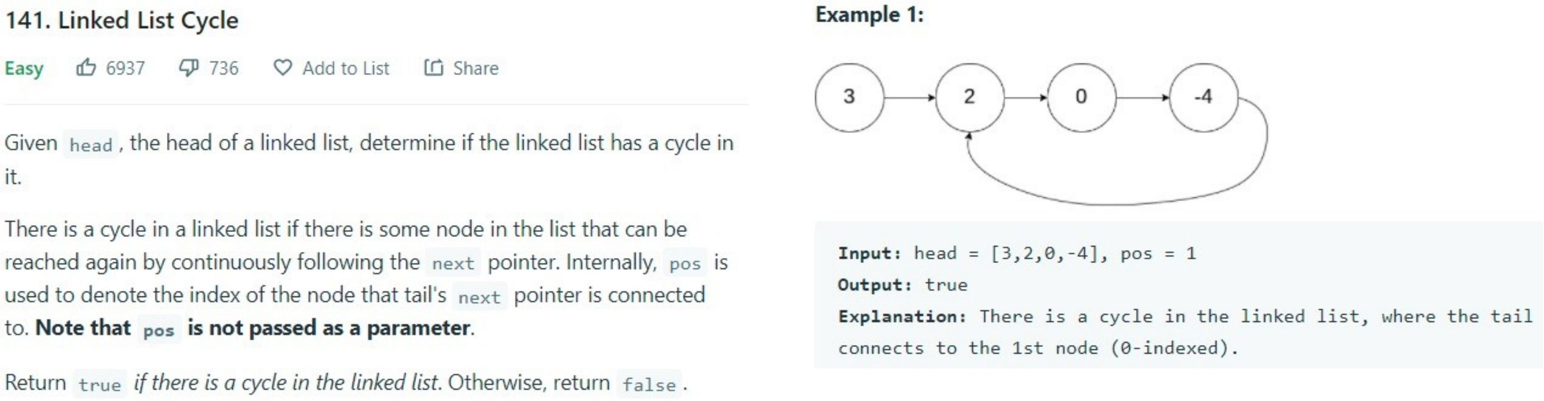
```
public ListNode sortList(ListNode head) {
    if(head == null || head.next == null)    return head;
    ListNode mid = midNode(head);
    ListNode nHead = mid.next;
    mid.next = null;

    ListNode l1 = sortList(head);
    ListNode l2 = sortList(nHead);


    return mergeTwoSL(l1, l2);
}
```

★ Merge 2 sorted list.



```
public ListNode mergeTwoSL(ListNode l1, ListNode l2){
    if(l1 == null || l2 == null)    return l1 == null ? l2 : l1;

    ListNode dummy = new ListNode(-1);
    ListNode prev = dummy;
    ListNode c1 = l1;
    ListNode c2 = l2;
    while(c1 != null && c2 != null){
        if(c1.val < c2.val){
            prev.next = c1;
            c1 = c1.next;
        }
        else{
            prev.next = c2;
            c2 = c2.next;
        }
        prev = prev.next;
    }

    prev.next = c1 == null ? c2 : c1;

    return dummy.next;
}
```

Linked List Cycle



141. Linked List Cycle

Easy 6937 736 Add to List Share

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

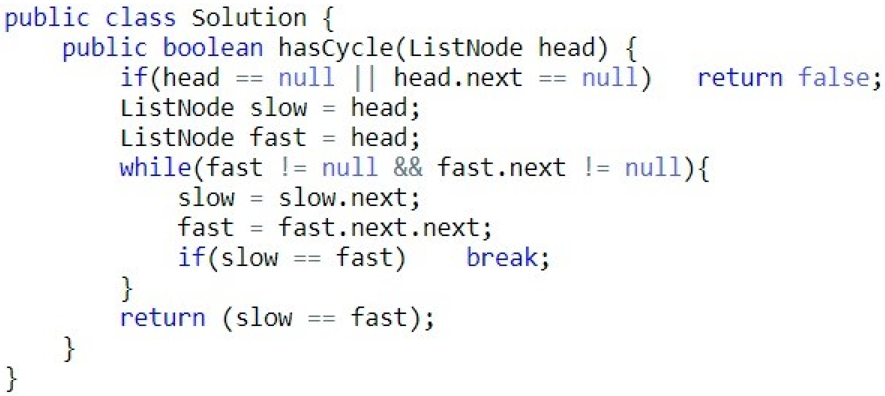
```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail
connects to the 1st node (0-indexed).
```

Approach. We will take two pointers (slow & fast) & check if they can meet at certain point.

⇝ If they meet, there is a cycle.

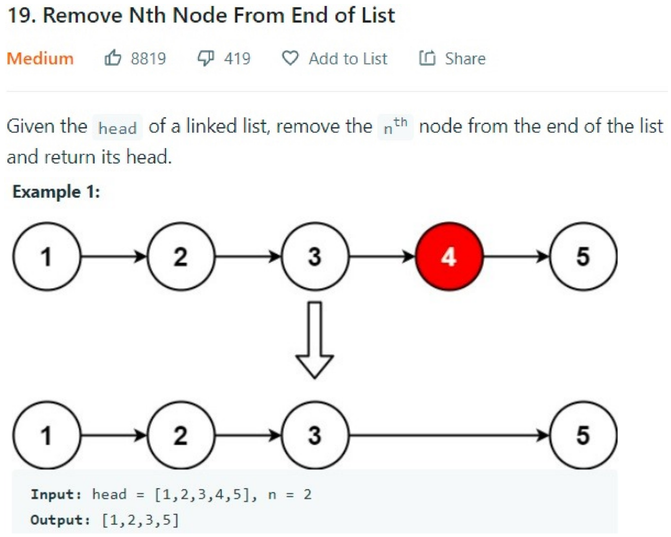
Edge Cases:- size == 0 || size == 1

Code:



```
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null || head.next == null)   return false;
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
            if(slow == fast)    break;
        }
        return (slow == fast);
    }
}
```

Remove N^{th} Node from end of list.



19. Remove Nth Node From End of List

Given the `head` of a linked list, remove the n^{th} node from the end of the list and return its head.

Example 1:

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

Approach:-

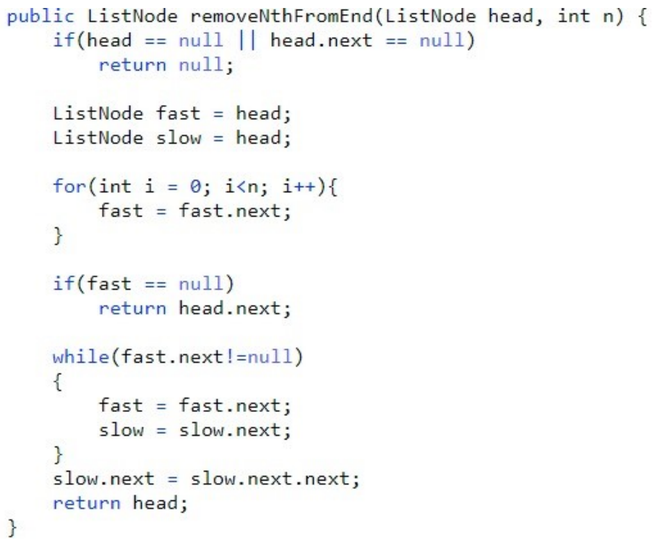
1. Take two pointers slow & fast.
2. Maintain gap of 'n' b/w slow & fast.
3. Check for case (size==n).
4. Move slow & fast together till end.
5. Virtually remove node & return head.

Edge Case:

1. head == null
   head.next == null } null

2. size == n

Code.



```
public ListNode removeNthFromEnd(ListNode head, int n) {
    if(head == null || head.next == null)
        return null;

    ListNode fast = head;
    ListNode slow = head;

    for(int i = 0; i<n; i++){
        fast = fast.next;
    }

    if(fast == null)
        return head.next;

    while(fast.next!=null)
    {
        fast = fast.next;
        slow = slow.next;
    }
    slow.next = slow.next.next;
    return head;
}
```