143. Reorder List

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

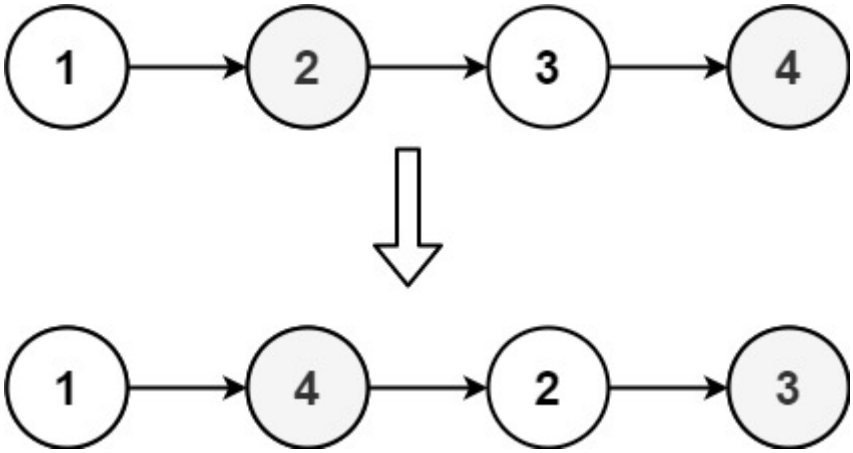
```
L<sub>0</sub> → L<sub>1</sub> → … → L<sub>n - 1</sub> → L<sub>n</sub>
```

Reorder the list to be on the following form:

```
L<sub>0</sub> → L<sub>n</sub> → L<sub>1</sub> → L<sub>n - 1</sub> →
L<sub>2</sub> → L<sub>n - 2</sub> → …
```

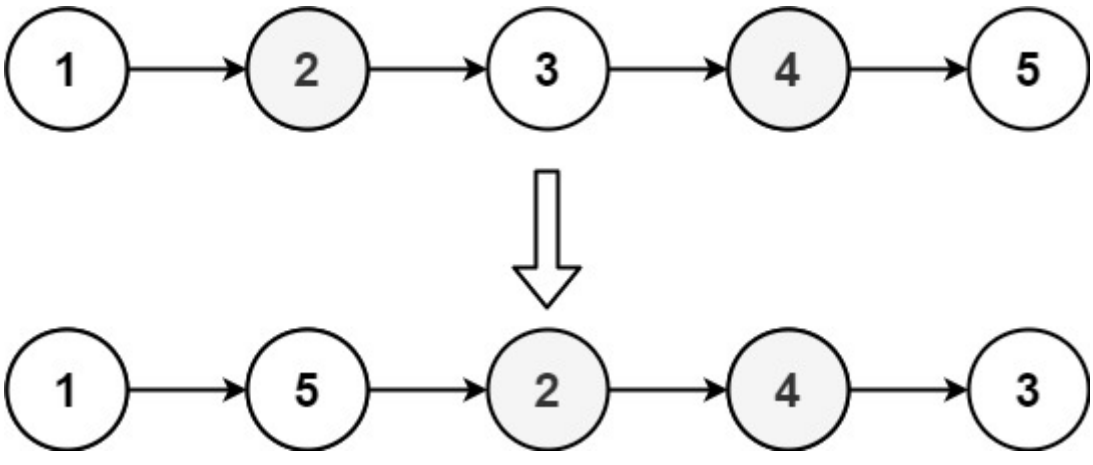
You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:




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

Example 2:




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

Constraints:

- The number of nodes in the list is in the range `[1, 5 * 10<sup>4</sup>]`.
- `1 <= Node.val <= 1000`

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reorderList(self, head: Optional[ListNode]) -> None:
        if head is None or head.next is None:
            return head
        mid = self.findMid(head)

        tempHead = mid.next
        mid.next  = None
        newHead = self.reverseLL(tempHead)

        curr1 = head
        curr2 = newHead
        forward1 = None
        forward2 = None

        while curr2!= None:
            forward1 = curr1.next
            forward2 = curr2.next

            curr1.next = curr2
            curr2.next = forward1

            curr1 = forward1
            curr2 = forward2
        return head


    def reverseLL(self,head):
        if head is None or head.next is None:
            return head

        curr = head
        prev = None
```

```
        while curr!=None:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr  = nxt
        return prev

    def findMid(self,head):
        if head is None or head.next is None:
            return head

        slow = head
        fast = head

        while fast.next is not None and fast.next.next is not None:
            slow = slow.next
            fast = fast.next.next
        return slow
```