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

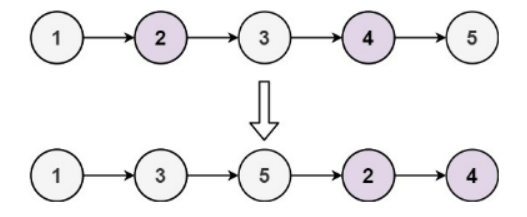
Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in O(1) extra space complexity and O(n) time complexity.

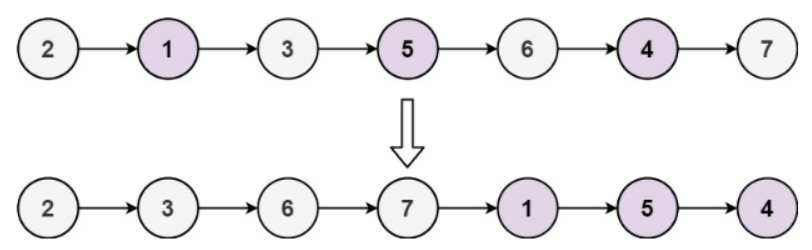
**Example 1:**



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

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

**Example 2:**



Input: head = [2,1,3,5,6,4,7]

Output: [2,3,6,7,1,5,4]

**Constraints:**

* The number of nodes in the linked list is in the range [0, 104].
* -106 <= Node.val <= 106

**Attempt 1: 2023-02-20**

**Solution 1: Fast and Slow pointer (30 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 oddEvenList(ListNode head) {

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

return head;

}

ListNode oddIter = head;

ListNode evenHead = head.next;

ListNode evenIter = evenHead;

while(evenIter != null && evenIter.next != null) {

oddIter.next = evenIter.next;

oddIter = oddIter.next;

evenIter.next = evenIter.next.next;

evenIter = evenIter.next;

}

oddIter.next = evenHead;

return head;

}

}

Time complexity:O(n). There are total n nodes and we visit each node once.

Space complexity:O(1). All we need is the four pointers.

**Refer to**

<https://leetcode.com/problems/odd-even-linked-list/solutions/127831/odd-even-linked-list>

## **Solution**

**Intuition**

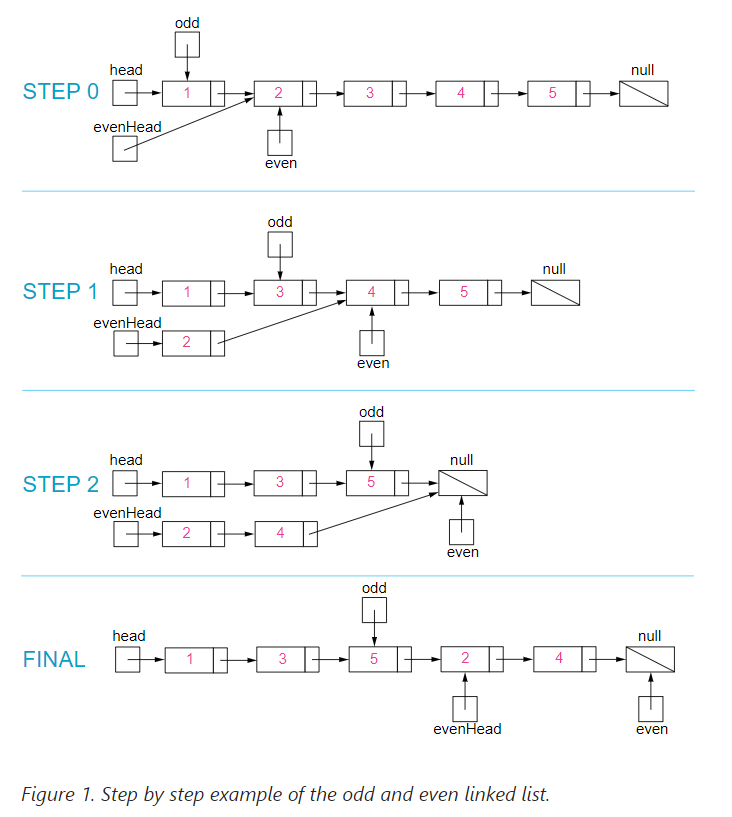
Put the odd nodes in a linked list and the even nodes in another. Then link the evenList to the tail of the oddList.

**Algorithm**

The solution is very intuitive. But it is not trivial to write a concise and bug-free code.

A well-formed LinkedList need two pointers head and tail to support operations at both ends. The variables head and odd are the head pointer and tail pointer of one LinkedList we call oddList; the variables evenHead and even are the head pointer and tail pointer of another LinkedList we call evenList. The algorithm traverses the original LinkedList and put the odd nodes into the oddList and the even nodes into the evenList. To traverse a LinkedList we need at least one pointer as an iterator for the current node. But here the pointers odd and even not only serve as the tail pointers but also act as the iterators of the original list.

The best way of solving any linked list problem is to visualize it either in your mind or on a piece of paper. An illustration of our algorithm is following:



public class Solution {

public ListNode oddEvenList(ListNode head) {

if (head == null) return null;

ListNode odd = head, even = head.next, evenHead = even;

while (even != null && even.next != null) {

odd.next = even.next;

odd = odd.next;

even.next = odd.next;

even = even.next;

}

odd.next = evenHead;

return head;

}

}

**Complexity Analysis**

* Time complexity : O(n). There are total n nodes and we visit each node once.
* Space complexity : O(1). All we need is the four pointers.