Technique: Append Function

Level: Easy

Odd Even Linked List: Given a Linked List L, separate it into 2 Linked Lists. One contains L's odd nodes and the other contains L's even nodes.

For example:
Input: Head -> 1 -> 2 -> 3 -> 4 -> 5

Result 1: Head -> 1 -> 3 -> 5
Result 2: Head -> 2 -> 4

Note: Odd and Even refer to the node's position, not value.

Questions to Clarify:
Q. Is the first node considered to have value of 1 or 0?
A. The first node will be 1, so it will be in the odd list.

Q. What if we have an empty input list, or only 1 node in the input list. What do we return for the 2 results?
A. If you have an empty input list, return 2 empty lists. If you have 1 node in the input list, return an odd list with the node and an empty even list.

Q. Should the result contain new nodes, or should I reuse nodes from the input list?
A. Rearrange the nodes from the input list. At the end of your program, the input list will be empty.

Solution:
We initialize 2 empty lists - odd and even.

We walk through nodes of the input list, and add each node to the correct list using the append function.

Pseudocode:
(Note: Never write pseudocode in an actual interview. Unless you're writing a few lines quickly to plan out your solution. Your actual solution should be in a real language and use good syntax.)

```
Pair<LinkedList> getOddEvenLists(LinkedList input)
     LinkedList odd = empty
     LinkedList even = empty

     int nodeIndex = 0;
     Node current = input.getHead();
```

```
while (current != null)
    nodeIndex++
    if (nodeIndex is even)
        append(even, current)
    else
        append(odd, current)

    current = current.next
 even.tail.next = null
 odd.tail.next = null

 return new Pair(odd, even)
```

Test Cases:
Edge Cases: empty input list, null list
Base Cases: single item input list, 2 items in input list
Regular Cases: even number of items, odd number of items

Time Complexity: O(n)

Space Complexity: O(1) because we are rearranging the nodes

```
public Pair<LinkedList> getOddEven(LinkedList input) {
    LinkedList odd = new LinkedList();
    LinkedList even = new LinkedList();

    Node current = input.getHead();
    int index = 0;
    while (current != null) {
        index++;
        LinkedList destination = index % 2 == 0 ? even : odd;
        destination.append(current);
        current = current.getNext();
    }

    // set last nodes' next = null

    if (even.getTail() != null)
        even.getTail().setNext(null);

    if (odd.getTail() != null)
        odd.getTail().setNext(null);

    return new Pair<LinkedList>(odd, even);
}

/*
```

```
 * Helper Code. Ask the interviewer if they want you to implement this.
 */
class LinkedList {
    Node head;
    Node tail;

    public LinkedList() {
        head = null;
        tail = null;
    }

    public void append(Node toAdd) {
        if (head == null) {
            head = toAdd;
        } else {
            tail.setNext(toAdd);
        }
        tail = toAdd;
    }

    public Node getHead() {
        return head;
    }

    public Node getTail() {
        return tail;
    }

    public void setHead(Node head) {
        this.head = head;
    }

    public void setTail(Node tail) {
        this.tail = tail;
    }
}


public class Node {
    Node next;
    int data;

    public Node(int data) {
        this.data = data;
    }

    public Node getNext() {
        return next;
    }

    public void setNext(Node next) {
        this.next = next;
    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }
}
```