1 Boxes and Pointers

(a) Draw a box and pointer diagram to represent the IntLists L, M, and N after each statement.

```
1 IntList L = IntList.list(1, 2, 3, 4);
2 IntList M = L.tail.tail;
3 IntList N = IntList.list(5, 6, 7);
4 N.tail.tail.tail = N;
5 L.tail.tail = N.tail.tail.tail.tail;
6 M.tail.tail = L;
```

(b) Extra Draw a box and pointer diagram to represent the IntLists L1, L2, and L3 after each statement.

```
1 IntList L1 = IntList.list(1, 2, 3);
2 IntList L2 = new IntList(4, L1.tail);
3 L2.tail.head = 13;
4 L1.tail.tail.tail = L2;
5 IntList L3 = IntList.list(50);
6 L2.tail.tail = L3;
```
Destructive or Nondestructive?

The method below takes in an IntList and returns the value of the head of the IntList. Assume that L is never null.

```java
/** Returns the head of IntList L. Assumes that L is not null. */
public static int getHead(IntList L) {
    int listHead = L.head;
    L = new IntList(5, null);
    return listHead;
}
```

Is the above method destructive or nondestructive? Explain.
3 Reversing a Linked List

Implement the following method, which reverses an IntList nondestructively. The original IntList should not be modified. Instead, the method should return a new IntList that contains the elements of L in reverse order.

/** Nondestructively reverses IntList L. */
public static IntList reverseNondestructive(IntList L) {

}
Extra Implement the following method which destructively reverses an IntList.

/** Destructively reverses IntList L using recursion. */
public static IntList reverseDestructive(IntList L) {

}
4 Inserting into a Linked List

Implement the following method to insert an element item at a given position position of an IntList L. For example, if L is (1 → 2 → 4) then the result of calling insert(L, 3, 2) yields the list (1 → 2 → 3 → 4). This method should modify the original list (do not create an entirely new list from scratch) Use recursion.

/** Inserts item at the given position in IntList L and returns the resulting IntList. If the value of position is past the end of the list, inserts the item at the end of the list. Uses recursion. */
public static IntList insertRecursive(IntList L, int item, int position) {

}
Implement the method described above using iteration. `insertIterative` is a destructive method and should therefore modify the original list (just like the previous problem, do not create an entirely new list from scratch).

```java
/** Inserts item at the given position in IntList L and returns the resulting
 * IntList. If the value of position is past the end of the list, inserts the
 * item at the end of the list. Uses iteration. */
public static IntList insertIterative(IntList L, int item, int position) {
```
Implement the following method to circularly shift an IntList to the left by one position *destructively*. For example, if the original list is $(5 \rightarrow 4 \rightarrow 9 \rightarrow 1 \rightarrow 2 \rightarrow 3)$ then this method should return the list $(4 \rightarrow 9 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5)$. Because it is a destructive method, the original IntList should be modified. Do not use the word `new`.

```java
/** Destructively shifts the elements of the given IntList L to the
 * left by one position. Returns the first node in the shifted list. */
public static IntList shiftListDestructive(IntList L) {
}
```