

## 1 Boxes and Pointers

---

Draw a box and pointer diagram to represent the IntLists after each statement.

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

## 2 Insertion

---

Implement the following method to insert an element into the given position of an IntList.

```
/** Insert a new item at the given position in L and return the resulting
 * IntList. If the position is past the end of the list, insert a new
 * node at the end of the list. */
public static IntList insert(IntList L, int item, int position) {
```

```
}
```

## 3 Reverse

---

Implement the following method, which reverses an IntList non-destructively.

```
/** Non-destructively reverses an IntList L. Do not modify the original
 * IntList. */
public static IntList reverseNondestructive(IntList L) {
```

```
}
```

## 4 Extra for Experts: Shifting a Linked List

---

Implement the following methods to circularly shift an `IntList` to the left destructively and non-destructively.

```
/** Destructively shifts the elements of the given IntList L to  
 * the left by one position (e.g. if the original list is  
 * (5, 4, 9, 1, 2, 3) then this method should return the list  
 * (4, 9, 1, 2, 3, 5)). Returns the first node in the shifted list.  
 * Don't use 'new'; modify the original IntList. */
```

```
public static IntList shiftListDestructive(IntList L) {
```

```
}
```

```
/** Non-destructively shifts the elements of the given  
 * IntList L to the left by one position. Returns the first node in the  
 * shifted list. Don't modify the original IntList. */
```

```
public static IntList shiftListNondestructive(IntList L) {
```

```
}
```