CS61B Week 2: Pointers

Draw box and pointer diagrams to represent the IntLists after each statement

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

See last page

Complete the following Java functions so that they perform as indicated in their comments

(a)
```
/** Given an integer n, return the given IntList L with every nth element removed.
 *  Don't use 'new'. You may modify the original IntList. */
public static IntList skipList(IntList L, int n) {
  IntList M = L;
  while (M != null) {
    for (int i = 0; i < n - 2; i++) {
      M = M.tail;
      if (M.tail == null) {
        break;
      }
    }
    if (M.tail == null) {
      break;
    }
    M.tail = M.tail.tail;
    M = M.tail;
  }
  return L;
}
```

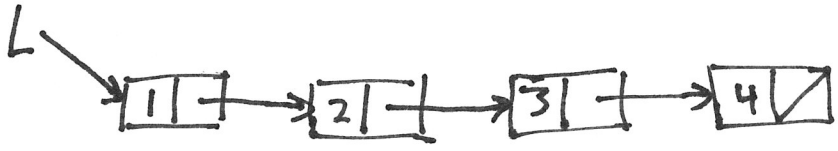
(b)
```
/** Return a new IntList that is the reverse of the given IntList L.
 *  Don't modify the original IntList. */
public static IntList reverseList(IntList L) {
  if (L == null) {
    return null;
  }
  IntList M = new IntList(L.head, null);
  L = L.tail;
  while (L != null) {
    M = new IntList(L.head, M);
    L = L.tail;
  }
  return M;
}
```

(c)
```
/** Return an IntList that is the reverse of the given IntList L.
 *  Don't use 'new'. You may modify the original IntList. */
public static IntList destructiveReverseList(IntList L) {
  if (L == null) {
    return null;
  }
  IntList M = L;
  IntList N = L.tail;
  L.tail = null;
  while (N != null) {
    IntList temp = N.tail;
    N.tail = M;
    M = N;
    N = temp;
  }
  return M;
}
```

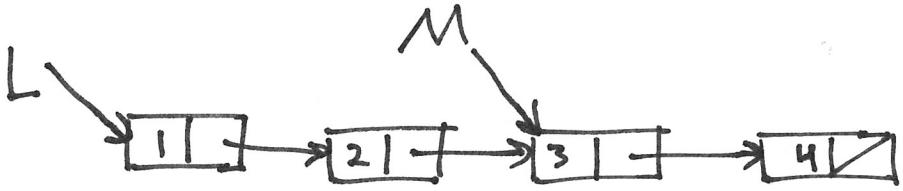
(d)
```
/** Return an IntList that is composed of the odd elements of the given
 *  IntList L followed by the even elements of that IntList, maintaining order.
 *  Ex. if L is {3, 4, 6, 5, 7, 2}, return {3, 5, 7, 4, 6, 2}
 *  Don't use 'new'. You may modify the original IntList. */
public static IntList oddEvenList(IntList L) {
  IntList firstEven = L;
  IntList walker = L;
  while (L != null && L.head % 2 == 0) {
    L = L.tail;
  }
  while (firstEven != null && firstEven.head % 2 != 0) {
    firstEven = firstEven.tail;
  }
  IntList oddWalker = L;
  IntList evenWalker = firstEven;
  while (walker != null) {
    if (walker.head % 2 != 0 && walker != oddWalker) {
      oddWalker.tail = walker;
      oddWalker = walker;
    } else if (walker.head % 2 == 0 && walker != evenWalker) {
      evenWalker.tail = walker;
      evenWalker = walker;
    }
    walker = walker.tail;
  }
  oddWalker.tail = firstEven;
  return L;
}
```

Sample Interview Question of the Week:
How would you figure out if an IntList contains a loop? Try to make your algorithm as efficient as possible.
Point two pointers to the first element of your IntList. Then, make a loop where one of the pointers steps forward once and the other steps forward twice during each iteration. If the faster pointer gets to the end of the list, there is no loop in your list. Otherwise, your pointers will collide at some point and you'll know your list loops back on itself somewhere!

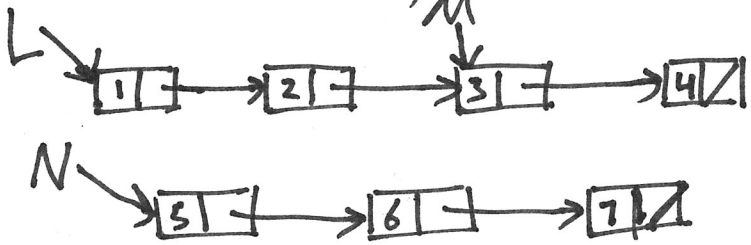
```
IntList L = IntList.list(1, 2, 3, 4);
```



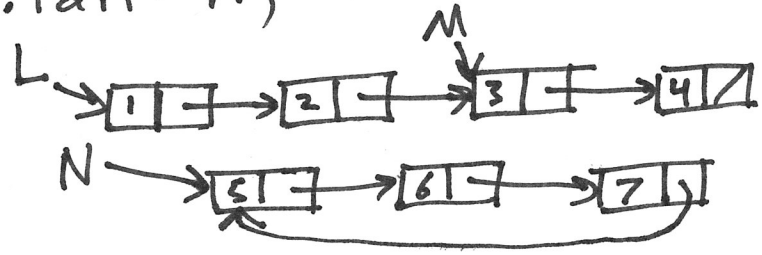

```
IntList M = L.tail.tail;
```



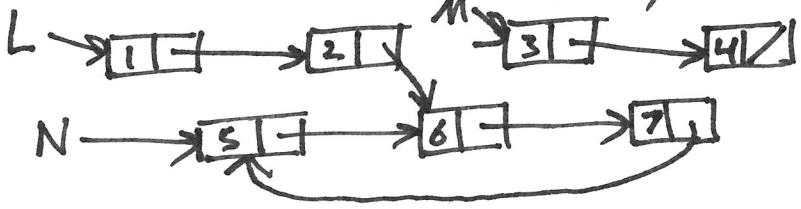

```
N = IntList.list(5, 6, 7);
```




```
N.tail.tail.tail = N;
```




```
L.tail.tail = N.tail.tail.tail.tail;
```




```
M.tail.tail = L;
```