CSE143 Section #19 Problems

For these problems, assume that we are using the standard ListNode class:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}
```

And that we are writing methods for a class called LinkedIntList that has a single data field of type ListNode called front:

```
public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

In solving these problems, you may not call any other methods of the class, you may not construct new nodes and you may not use any auxiliary data structure to solve this problem (no array, ArrayList, stack, queue, String, etc). You also may not change any data fields of the nodes. You MUST solve these problems by rearranging the links of the lists involved.

1. Write a method evenSum that returns the sum of the values in even indexes in a list of integers. Assume we are using 0-based indexing where the first value in the list has index 0, the second value has index 1 and so on. The values we are interested in are the ones with even indexes (the value at index 0, the value at index 2, the value at index 4, and so on).

   For example, if a variable called list stores the following sequence of values:

   ```
   [1, 18, 2, 7, 39, 8, 40, 7]
   ```

   then the call:

   ```
   list.evenSum()
   ```

   should return the value 82 (1 + 2 + 39 + 40). Notice that what is important is the position of the numbers (index 0, index 2, index 4, etc), not whether the numbers themselves are even. If the list is empty, your method should return a sum of 0.

2. Write a method removeDuplicates that removes any duplicates from a list of integers. The resulting list should have the values in the same relative order as their first occurrence in the original list. In other words, a value i should appear before a value j in the final list if and only if the first occurrence of i appears before the first occurrence of j in the original list. For example, if a variable called list stores the following:

   ```
   [14, 8, 14, 12, 1, 14, 11, 8, 8, 10, 4, 9, 1, 2, 5, 2, 4, 12, 12]
   ```

   and the following call is made:

   ```
   list.removeDuplicates();
   ```

   then list should store these values after the call:

   ```
   [14, 8, 12, 1, 11, 10, 4, 9, 2, 5]
   ```

3. Write a method switchPairs that switches the order of elements in a linked list of integers in a pairwise fashion. Your method should switch the order of the first two values, then switch the order of the next two, switch the order of the next two, and so on. For example, if the list initially stores these values:

```
[3, 7, 4, 9, 8, 12]
```

your method should switch the first pair (3, 7), the second pair (4, 9) and the third pair (8, 12), to yield this list:

```
[7, 3, 9, 4, 12, 8]
```

If there are an odd number of values in the list, the final element is not moved. For example, if the original list had been:

```
[3, 7, 4, 9, 8, 12, 2]
```

It would again switch pairs of values, but the final value (2) would not be moved, yielding this list:

```
[7, 3, 9, 4, 12, 8, 2]
```

4. Write a method takeSmallerFrom that compares two lists of integers, making sure that the first list has smaller values in corresponding positions. For example, suppose the the variables list1 and list2 refer to lists that contain the following values:

```
list1: [3, 16, 7, 23]
list2: [2, 12, 6, 54]
```

If the following call is made:

```
list1.takeSmallerFrom(list2);
```

the method will compare values in corresponding positions and move the smaller values to list1. It will find that among the first pair, 2 is smaller than 3, so it needs to move. In the second pair, 12 is smaller than 16, so it needs to move. In the third pair, 6 is smaller than 7, so it needs to move. In the fourth pair, 54 is not smaller than 23, so those values can stay where they are. Thus, after the call, the lists should store these values:

```
list1: [2, 12, 6, 23]
list2: [3, 16, 7, 54]
```

One list might be longer than the other, in which case those values should stay at the end of their list. For example, for these lists:

```
list1: [2, 4, 6, 8, 10, 12]
list2: [1, 3, 6, 9]
```

the call:

```
list1.takeSmallerFrom(list2);
```

should leave the lists with these values:

```
list1: [1, 3, 6, 8, 10, 12]
list2: [2, 4, 6, 9]
```