Material Covered

- Fast sorting – the Merge Sort

Notes:

- The three exercises in this lab form a Merge Sort algorithm on a LinkedList. (The actual LinkedList and Node classes are supplied – you do not have to work with nodes and links directly at all.)
- The Bronze and Silver exercises are independent. The Gold exercise requires that you have completed the other two.
- For a change, the Gold exercise it *not* the most difficult of the three – it is actually quite short and there's no "trick" to it. It's there only to combine the other two exercises to complete the sorting algorithm.
- Try to do all three, since a partially-complete sorting program is not of much use.

Preparation:

- Download and compile the `Node.java` and `LinkedList.java` files. These give you a complete implementation of a linked list of integers. Look over the contents of these files to see the various methods that are available to you. You will need them in all three exercises.
- The implementation details of an algorithm are strongly influenced by the type of data structure being used (e.g. array vs. ArrayList vs. linked list vs. …) But the fundamental algorithm always remains the same. The Merge Sort is:
  1. Split the list into two equally-sized sublists (plus or minus one element).
  2. Sort the two sublists.
  3. Use a merge algorithm to combine them into a sorted list.
- The Bronze exercise will implement step 1. The Silver exercise will implement step 3. The Gold exercise will put them together, and add step 2.

Bronze Medal The `split` method

1. Start with the file `TemplateLab11Bronze.java`.
2. Complete the `split` method at the end of the file. This method should accept a linked list containing 2 or more nodes (`mainList`) and two other linked lists (`list1` and `list2`) which should initially be empty. It should split up the nodes in `mainList`, putting half of them in `list1`, and half of them in `list2`, by adding nodes alternately to the two lists. It will not create any new nodes – it will re-link the existing ones. The list `mainList` should become empty because of this. Make use of the methods already provided in `LinkedList.java`, which will do most of the low-level work for you.

3. Run the supplied main program. The output should look like this. (The data will be random.) Note that the integers from the original list appear alternately in the two sublists, and that the original list becomes empty.

```
Original list:
<< 13  27  36  86  92  0  85  97  3  99  88  28  87  9  41 >>
First sublist:
<< 13  36  92  85  3  88  87  41 >>
Second sublist:
<< 27  86  0  97  99  28  9 >>
Original list:
<<>>
```

Silver Medal The `merge` method

1. Start with the file `TemplateLab11Silver.java`.
2. Complete the `merge` method at the end of the file. It accepts two sublists `list1` and `list2` which must already be sorted into ascending order. It also accepts a list `combinedList` which should initially be empty. It should perform a standard merge algorithm to combine the nodes from both `list1` and `list2`, placing them into `combinedList` so that it will also be in ascending order. Again, no new nodes are created. The existing nodes are simply re-linked into a new list. As a result, both `list1` and `list2` will become empty.
3. Run the supplied main program. The output should look like this. (The data will be random.) Note that the sublists become empty.

```
First sublist:
<< 5  24  30  53  57  57  70  89  95  103 >>
Second sublist:
<< 2  7  30  49  57  62  86  105  113  136 >>
Merged list:
<< 2  5  7  24  30  30  49  53  57  57  57  62  70  86  89  95  103  105  113  136 >>
First sublist:
<<>>
Second sublist:
<<>>
```

Gold Medal The `mergeSort` method

1. Begin with the file `TemplateLab11Gold.java`.
2. Add the `split` and `merge` methods from the Bronze and Silver exercises.
3. Complete the `mergeSort` method, which will accept a `LinkedList` and sort it into ascending order using a recursive Merge Sort algorithm. This should be quite short (shorter than the `merge` method), since most of the work is done by the other two methods. (Hint: There is a `oneOrLess()` method in the `LinkedList` class which will be useful.)
4. Run the supplied main program. The output should look like this. (The data will be random.)

```
Original list:
<< 63  90  58  26  91  78  81  84  42  23  43  71  89  2  15 >>
Sorted list:
<< 2  15  23  26  42  43  58  63  71  78  81  84  89  90  91 >>
```