CS 6301.002. Implementation of advanced data structures and algorithms
Spring 2016
Short Project 1
Wed, Jan 20, 2016

Version 1.0: Initial description (Jan 20, 9:00 AM).
Version 1.1: Additional information added to problem c (Jan 20, 8:15 PM).
Version 1.2: Problem f added (Jan 21, 8:10 AM).

Due: 1:00 PM, Thu, Jan 28.

Submission procedure:
Create a folder whose name starts with the name of the group (e.g., "G13"). Place all files you are submitting in that folder. There is no need to submit binary files created by your IDE (such as class files). Make sure there is a "readme" file that explains the contents of the files being submitted. Zip the contents into a single zip or rar file. Upload that file on elearning. Submission can be revised before the deadline. Only the final submission before the deadline will be graded. Only one member of each group needs to submit project.

Solve at least one problem from the following list. First solution will be graded out of 10. Each additional problem will be considered for 1 extra point.

No sample data will be provided for short projects. Create your own input data sets for testing.

Topic: Lists, queues, stacks.

a. Given two linked lists implementing sorted sets, write functions for union, intersection, and set difference of the sets.

```
public static<T extends Comparable<? super T>>
    void intersect(List<T> l1, List<T> l2, List<T> outList) {
       // Return elements common to l1 and l2, in sorted order.
       // outList is an empty list created by the calling
       // program and passed as a parameter.
       // Function should be efficient whether the List is
       // implemented using ArrayList or LinkedList.
       // Do not use HashSet/Map or TreeSet/Map or other complex
       // data structures.
    }

public static<T extends Comparable<? super T>>
    void union(List<T> l1, List<T> l2, List<T> outList) {
       // Return the union of l1 and l2, in sorted order.
       // Output is a set, so it should have no duplicates.
    }

public static<T extends Comparable<? super T>>
    void difference(List<T> l1, List<T> l2, List<T> outList) {
       // Return l1 - l2 (i.e, items in l1 that are not in l2), in sorted order.
       // Output is a set, so it should have no duplicates.
    }
```

b. Suppose large numbers are stored in a list of integers. Write functions for adding and subtracting large numbers.

```
public static void add(List<Integer> x, List<Integer>  y,List<Integer> z, int b) {
       // Return z = x + y.  Numbers are stored using base b.
       // The "digits" are stored in the list with the least
       // significant digit first.  For example, if b = 10, then
       // the number 709 will be stored as 9 -> 0 -> 7.
       // Assume that b is small enough that you will not get any
       // overflow of numbers during the operation.
}
```

```
   public static void subtract(List<Integer> x, List<Integer>  y,List<Integer> z, int b)
{
          // Return z = x - y.  Numbers are stored using base b.
          // Assume that x >= y.
   }
```

c. Write the Merge sort algorithm that works on linked lists. This will be a member function of a linked list class, so that it can work with the internal details of the class. The function should use only O(log n) extra space (mainly for recursion), and not make copies of elements unnecessarily. You can start from the SinglyLinkedList class provided or create your own.

```
   static<T extends Comparable<? super T>> void mergeSort(SortableList<T> lst) { ... }
```

Download a skeleton of SortableList.java from the class page.
http://www.utdallas.edu/~rbk/teach/2016s/java/code/SortableList.java

d. Implement a recursive algorithm without recursion, by using a stack to simulate recursion. You may work on any recursive algorithm that has multiple recursive calls such as Merge Sort, Binary tree traversals, Quick sort, or, Linear-time median.

e. Extend the "unzip" algorithm discussed in class on Thu, Jan 21 to "multiUnzip" on the SinglyLinkedList class:

```
   void multiUnzip(int k) {
        // Rearrange elements of a singly linked list by chaining
        // together elements that are k apart.  k=2 is the unzip
        // function discussed in class.  If the list has elements
        // 1..10 in order, after multiUnzip(3), the elements will be
        // rearranged as: 1 4 7 10 2 5 8 3 6 9.  Instead if we call
        // multiUnzip(4), the list 1..10 will become 1 5 9 2 6 10 3 7 4 8.
   }
```

f. Write recursive and nonrecursive functions for the following tasks:
(i) reverse the order of elements of the SinglyLinkedList class
(ii) print the elements of the SinglyLinkedList class, in reverse order.
Write the code and annotate it with proper loop invariants.