CSE 326 DATA STRUCTURES
HOMEWORK 6

Due: **Friday, Nov 16, 2007** at the beginning of class.

1. **Insertion Sort on Linked Lists**

   Implement insertion sort using a linked list as your input, instead of an array. Specifically, assume you have a structure:

   ```
   class Node {
     int data;
     Node next;
   }
   ```

   Your input comes as the head node of a linked list. Design pseudocode to implement a recursive version of insertion sort `Node insertionsort(Node list)`:

   - On an empty list input, return the empty list.
   - On a non-empty list input, recursively sort the tail of the list, then insert the head of the list into the appropriate place within the entire list.

   Note: You should never create or destroy record structures. You may temporarily unlink them from the list within your function but your list should essentially always contain all elements. Like sorting an array, this is a destructive operation: the original order of the list is lost. (In other words, you should never create a "`new Node`". You should not create a new list by copying data from the old one.)

2. **Mergesort on Linked Lists**

   Continuing our consideration of lists, we would like to implement recursive mergesort. There are two major steps: splitting a list into two equal (or almost equal) size lists and is merging two sorted lists. Again, you cannot create a `new Node`. Note that, unlike performing mergesort on arrays, this will require no extra memory space.

   (a) Assume that you the following class available to you:

   ```
   class Pair {
     Node first;
     Node second;
   }
   ```

   Design pseudocode for a function `Pair split(Node list)` where `Pair.first` and `Pair.second` point to two lists that differ in length by at most 1. Hint: if the list has zero, one, or two records, then it is easy to return the right pair of lists. Otherwise, recursively split the list.

   (b) Design pseudocode for a function `Node merge(Node left, Node right)` that given two sorted lists returns the result of merging them into one sorted list.

   (c) Use `split` and `merge` to define pseudocode for a function `Node mergesort(Node list)`.

3. **Some searching, some sorting, and some sums**

Suppose you are given as input n positive integers and a number k. Our goal is to write an algorithm to determine if there are any four of them, **repetitions allowed**, that sum to k. As an example, if $n = 7$, the input numbers are 6, 1, 7, 12, 5, 2, 14 and $k = 15$, the answer should be YES because $6 + 5 + 2 + 2 = 15$.

(a) First solve the simpler problem that determines if there are any two numbers in a list that sum to k (repetitions allowed). Show that your algorithm runs in time $O(n \log n)$.

(b) Now observe that the sum of four numbers is the sum of two pairs of numbers. Carefully explain how we can solve the original problem (i.e., whether a four number sum exists). For full credit, show that your algorithm runs in time $O(n^2 \log n)$.

4. **Extra Credit**

The input to this problem consists of a list of 7-digit phone numbers written as simple integers (e.g., 5551212 represents the phone number 555-1212). No number appears in the input more than once, but there is no other limit on the size of the input.

Write precise pseudocode for a program that prints out the phone numbers in the list in ascending order.

Your solution **must not** use more than 2MB of memory. You must explain why this is true of your solution. (Note: No cheating! It cannot use any external storage — hard disks, tapes, punched cards, the network, an infinite tape, etc.) Assume that as input you will get to see each number only *one time* (i.e., do **not** assume the data is in a file and available there for manipulation).

Extra-Extra Credit: Implement your solution to this problem in Java (or some other language). Email your solution to your TA. Your solution needs to accept input from "System.in". It will be tested as (for example):

```
java YourProgram < "phonenumbers.txt"
```