Due: **Friday, April 25, 2008** at the beginning of class.

**Problem 1. Leftist Heaps**

In this problem, we will build leftist heaps by inserting values one at a time.

(a) Show the result of inserting keys 7 to 1 in descending order (i.e. insert 7, then 6, then 5, etc.) into an initially empty leftist heap. Use the leftist heap insert (i.e. merge) algorithm at each step. You don’t need to show each step for this process, but be warned that if all you write down is the final answer and you get it wrong, it will be hard to award any partial credit.

(b) Show the result of inserting keys 1 to 15 in ascending order (i.e. insert 1, then 2, then 3, etc.) into an initially empty leftist heap. Use the leftist heap insert (i.e. merge) algorithm at each step. Again, intermediate steps are not required, but may help you earn partial credit.

(c) Prove or disprove: A perfect binary tree forms if keys 1 to $2^k - 1$ are inserted in order (again this means 1 first, then 2 etc) into an initially empty leftist heap. $k$ is a positive integer. (Hint: use induction; you have already worked through the base case above.)

**Problem 2. Skew Heaps**

(a) Weiss 6.26. You only need to show the final result, but note that if you do this it will be hard to award partial credit if the final result has problems.

**Problem 3. Binomial Trees and Queues**

A binomial tree of height 0, $B_0$, is a one-node tree. A binomial tree of height $k$, $B_k$ is formed by attaching a binomial tree, $B_{k-1}$ to the root of another binomial tree another binomial tree $B_{k-1}$. (These are the same definitions as in Weiss.)

(a) Weiss 6.32.

(b) Prove that a binomial tree $B_k$ has $2^k$ nodes.
Problem 4. increaseKey and decreaseKey for Leftist Heaps and Binomial Queues

Recall that, given a pointer to an object in a binary heap, we can perform the decreaseKey operation on the object by decreasing its key value and then performing a percolateUp operation. Similarly, we can perform the increaseKey operation by increasing the key value and then doing a percolateDown. For binary heaps of size $n$, percolateUp and percolateDown are both $O(\log n)$ in the worst case; thus decreaseKey and increaseKey are $O(\log n)$ in the worst case for binary heaps.

Here we ask you to analyze the performance of decreaseKey and increaseKey for leftist heaps and binomial queues. For this problem, you will assume the following:

- decreaseKey and increaseKey are again implemented by adjusting the key value of a node and performing either a percolateUp or a percolateDown operation to satisfy the heap order property. As above, the pointer to the object (i.e., the node) is given.
- The parent of a node can be discovered in constant time (e.g., with an extra pointer per node).
- In the analysis below, “complexity” refers to the big-O bound.

Given these assumptions, answer the following questions. Note: the justification for each of your answers is a large part of the credit to be earned.

(a) What is the worst case complexity of performing a decreaseKey operation on a leftist heap? Justify your answer.

(b) What is the worst case complexity of performing an increaseKey operation on a leftist heap? Justify your answer.

(c) What is the worst case complexity of performing a decreaseKey operation on a binomial queue? Justify your answer.

(d) What is the worst case complexity of performing an increaseKey operation on a binomial queue? Justify your answer.