Due: **Friday, April 18, 2008** at the beginning of class. Be sure to write your name on your homework. As before, LaTeX pdf or other typeset documents are preferred, but not required.

**Problem 1. Binary Min Heaps**

This problem will give you some practice with the basic operations on binary min heaps.

(a) Starting with an empty binary min heap, show the result of inserting, in the following order, 10, 12, 1, 14, 6, 5, 8, 3, 9, 13, and 2, one at a time (using percolate up each time), into the heap. By *show* here we mean “draw the resulting binary tree with the values at each node.”

(b) Now perform two `deleteMin` operations on the binary min heap you constructed in part (a). Show the binary min heaps that result from these successive deletions (“draw the resulting binary tree with values at each node”).

(c) Instead of inserting the elements in part (a) into the heap one at a time, suppose that you use the linear time worst case algorithm described on page 211 of Weiss (Floyd’s algorithm). Show the resulting binary min heap tree. (It would help if you showed the intermediate trees so if there are any bugs in your solution we will be better able to assign partial credit, but this is not required.)

**Problem 2. Merging “Perfect” Binary Heaps**

A binary heap has the structure property that it is complete. Merging two binary heaps can be performed with Floyd’s method. There are special values of \( n \) for which the performance is better. Here, we consider the case of merging two binary heaps that are full enough to be perfect binary trees, i.e., complete trees with all the leaves at the same height. (Weiss refers to these as “full complete trees”.)

(a) Weiss problem 6.17(a). A concise answer in words is sufficient. In addition, explain why your algorithm has the stated worst case complexity.

(b) Weiss problem 6.17(b). A concise answer in words is sufficient. In addition, explain why your algorithm has the stated worst case complexity.

(c) Based on the algorithms discussed or derived in this problem, what are the asymptotic bounds on the worst case running time for merging two binary heaps of equal size? (You may assume that the bounds in parts (a) and (b) are actually \( \Theta(\log n) \).)
Problem 3. Alternate `remove()` Algorithm for Heaps

One way to remove an object from a binary (min) heap is to decrease its priority value by $\infty$ and then call `deleteMin()`. An alternative is to simply remove it from the heap, thus creating a hole, and then repair the heap. This alternative is itself analogous to (but not the same as) the `deleteMin()` operation.

(a) Write pseudocode for an algorithm that will perform the `remove` operation according to the alternative approach described above. Your pseudocode should implement the method call `remove(int index)`, where `index` is the index into the heap array for the object to be removed. Your pseudocode can make calls to the following methods described in lecture: `insert()`, `deleteMin()`, `percolateUp()`, and `percolateDown()`. In the lecture pseudocode, the objects are the values themselves (rather than being objects that have a value field and a pointer to some other object of a different type); you may follow the lecture's example.

(b) What is the worst case complexity of this algorithm?

Problem 4. d-Heap Arithmetic

Binary heaps implemented using an array have the nice property of finding children and parents of a node using only multiplication and division by 2 and incrementing by 1. This arithmetic is often very fast on most computers, especially the multiplication and division by 2, since this corresponds to simple bitshift operations. In $d$-heaps, the arithmetic is also fairly straightforward, but is no longer necessarily as fast. In this problem you'll figure out exactly what this math is.


(b) For what values of $d$ will these operations be implementable with bit shifts instead of divisions and multiplications? How many bit shifts are needed?

Hint: There are fairly concise solutions to this problem; if your solution is becoming particularly complicated, you might want to rethink your approach. In particular, it is worth thinking about where to put the root in the array, as some choices may simplify the calculations compared to others.