Assignment 5. **Hashing and sorting.**

Given February 25, due Monday, March 9.

1. One advantage of storing a binary tree structure using pointers is that the storage needed (pointers plus data) cannot be more than twice the storage for the data alone. A node in a binary tree in "incomplete " if it has fewer than two children. A leaf is incomplete because it has no children. A node with a single child is also incomplete in this sense. A binary tree has imbalance l if the highest incomplete node is l higher than the lowest incomplete node.

a. Show that $l = 0$ only when the tree is a complete tree of height h for some h.

b. Suppose we store a binary tree in a heap array, that is, the data go into the array, a and the descendents of $a(i)$ are $a(2i)$ and $a(2i+1)$. Find the maximum and minimum percentage of unused elements of a if the tree has height h and imbalance l, and a has 2^h entries in all.

2. We are writing software for a wake-up call service. A person can call, give her name, and schedule a wake-up call. She can also call to ask whether she has a call scheduled (people are forgetful during stressful business trips), with the option to cancel if it is scheduled. The software also has to be able to get the next call from those scheduled and delete it once that call is made.

a. Describe the operations (what they return and remember) of the functions:

- `return_next()`
- `delete_next()`
- `insert( name, time)` ("name" is a character string.)
- `inquire_name( name)`
- `delete_name( name)`

b. Give pseudo code (or real code) that implements these operations using a heap, hash table, and pointers. The operations should be as efficient as possible (in order of magnitude) for these data structures.

3. A complicated data structure contains, as one field (or "a class contains as one member") a string of up to 100 characters. We have about a million such objects. We will often test whether the strings of two objects are the same, so it is important that this test be fast. We are willing to do some precomputation to make the comparisons faster. Would it help to precompute the hash values of each string? Suppose we have a perfect hash function and the hash value is a four byte (32 bit) integer. What is the probability that two of the strings hash to the same value? Is computing the hash value much more expensive than doing a character by character string comparison? Suppose the hash function is one of the simple ones.