Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Fall 2004
Homework 4b
Due on Tuesday, 09/28/04 (beginning of class)

1. **[5 Points] Sorting with Trees.** (This is a short review problem.)

   Imagine that we have a binary search tree with n elements in it. Recall that an in-order traversal of the tree is a recursive traversal which takes a node v as input and does the following: Recursively traverse the left subtree of v, print out the value of v, and then recursively traverse the right subtree of v.

   (a) In a binary search tree, an in-order traversal beginning at the root prints out the contents of the tree in increasing order. Explain briefly (in one or two sentences) why an in-order traversal of a tree with n elements takes $O(n)$ time regardless of how imbalanced the tree may be.

   (b) Professor Lai, who now teaches at the Massachusetts Institute of Tricycles, wants to use in-order traversals to get a fast sorting algorithm. Here is his idea: Given an unsorted array of n elements, first insert them one-at-a-time into an initially empty binary search tree. Then do an in-order traversal of the binary search tree. Each time the in-order traversal prints out a number, put that in the next available position in a new array. The new array will be the sorted version of the original array! Briefly explain why this algorithm can require $\Theta(n^2)$ in the worst case. Poor Professor Lai. No tenure again!

   (c) Briefly explain why, had the Professor used a tree that was guaranteed in advance to always remain balanced (e.g. use one of your favorites from CS70 or CS140), a sorting algorithm that is asymptotically "as good as it is possible to get" can be obtained. (Maybe you should be getting tenure instead!)

2. **[75 Points] Implementing B-Trees!** In this assignment you will implement B-Trees. In particular, your implementation can assume that $t = 2$. Your code should have a `Node` element similar to the one suggested in the lecture notes. Your program should support the operations `insert(x)`, which inserts the integer `x` into the tree, and `find(x)` which returns a boolean indicating whether or not `x` is in the tree. For insertion, use the one-phase method described in class (in which insertion is done in a single pass down the tree). In addition, your program should have a method called `height()` which returns the height of the tree. (A tree with a single node has height 0.) Please do this programming exercise without looking at the book. For one thing, the book's pseudocode isn't particularly clean or clear. In addition, not using the book will encourage you to first write your own pseudocode, resulting in your own summary of our rather involved high-level class discussion. Starting with clear pseudocode is recommended: if you're careful at the design stage, you might be able to get away with very little debugging.

You should turn in a copy of your code and a sample run of a test program which first inserts the even numbers from 0 to 20 (in order) into the tree. Next, invoke the `height()` method and print the height of your tree. Finally, the test program should call`find` on each of the numbers ranging from 0 to 20 (odd and even) and print each query's result.

You have several options when turning in a sample run of your program:

(a) Use script (do `man script` to learn more). Do not create such a script inside an emacs buffer because it won't print correctly! In fact, I don't recommend `script` because there are a variety of situations in which it won't print correctly.

(b) Run your program in an emacs `shell` buffer interactively. Cut and paste the parts of the interaction that show your program running into another buffer, save as a text file, and then print this.

(c) Once you're sure your program works the way you'd like, you could use redirection to generate text files.