Foundations of Algorithms

Homework #1

Collaboration groups will be set up in Blackboard by the end of the week; however, there are no collaborative problems on this first assignment. All solutions are to be the result of individual effort.

Self-Study Problems

All of the following problems come from the textbook and have solutions posted on the web at

http://mitpress.mit.edu/algorithms.

You are permitted to use this site to examine solutions for these problems as a means of self-checking your solutions. These problems will not be graded.

Problems: 2.2-2, 2.3-5, 12.1-2, 12.3-3, 21.2-6.

Problems for Grading

1. [20 points] Use induction to prove $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$.

2. CLRS 2-1: Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to *coarsen* the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

   (a) [5 points] Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

   (b) [5 points] Show how to merge the sublists in $\Theta(n \lg(n/k))$ worst-case time.

   (c) [5 points] Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?

   (d) [5 points] How should we choose k in practice?

3. [20 points] Write a $\Theta(m+n)$ algorithm that prints the in-degree and the out-degree of every vertex in an m-edge, n-vertex directed graph where the directed graph is represented using adjacency lists.

4. [20 points] Consider the following algorithm for doing a postorder traversal of a binary tree with root vertex *root*.

Algorithm 1 Postorder Traversal

```
POSTORDER(root)
  if root ≠ null then
    POSTORDER(root.left);
    POSTORDER(root.right);
    visit root;
  end if;
```

Prove that this algorithm runs in time $\Theta(n)$ when the input is an n-vertex binary tree.

5. [20 points] We define an AVL binary search tree to be a tree with the binary search tree property where, for each node in the tree, the height of its children differs by no more than 1. For this problem, assume we have a team of biologists that keep information about DNA sequences in an AVL binary search tree using the specific weight (an integer) of the structure as the key. The biologists routinely ask questions of the type, "Are there any structures in the tree with specific weight between a and b, inclusive?" and they hope to get an answer as soon as possible. Design an efficient algorithm that, given integers a and b, returns true if there exists a key x in the tree such that $a \leq x \leq b$, and false if no such key exists in the tree. Describe your algorithm in pseudocode *and* English. What is the time complexity of your algorithm? Explain.