EECS 311: Data Structures and Data Management	Homework 4
Assigned: 10/14/08	Due: 10/21/08

1 Reading.

Read the supplemental splay tree handout. (If you didn't get one in class, you can find it on Blackboard.)

2 Problems.

1. Show the trees that result from inserting keys 10, ..., 1, in decreasing order, into an initially empty AVL tree. Show the tree after each insert.

2. Show the trees that result from inserting keys 312, 488, 682, 405, 170, in this order into an initially empty splay tree. Show the tree after each insert.

3. Suppose we have a dictionary that contains n keys. Given any key k, suppose we run *find(k)* repeatedly m times in a row. Answer the following questions as accurately as possible.

   (a) What is the worst case runtime of the first call to *find(k)* if the dictionary is implemented with an AVL tree? Explain.

   (b) What is the worst case runtime of the first call to *find(k)* if the dictionary is implemented with a splay tree? Explain.

   (c) What is the *total* worst case runtime of the second through mth call to *find(k)* if the dictionary is implemented with an AVL tree? Explain.

   (d) What is the *total* worst case runtime of the second through mth call to *find(k)* if the dictionary is implemented with a splay tree? Explain.

4. Given a splay tree implementation of a dictionary, give an algorithm for *chop* that takes two keys k_1 and k_2 and removes all keys strictly less than k_1 and strictly greater than k_2. You may use any of the standard splay tree operations, *create*, *insert*, *delete*, *find*, and *splay*. Prove that your chop algorithm has amortized runtime of $O(\log n)$, i.e., that any sequence of m operations on an initially empty dictionary has worst-case runtime $O(m \log n)$ where n is an upper bound on the number of keys in the tree at any one time. (Hint: You should be able to use the in-class analysis of splay trees as a black box to prove the runtime. The proof is very short.)