Problem Set 6

This problem set is due **at 9:00pm** on **Wednesday, April 4, 2012**.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of the first page of your solution with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated. The homework template (LaTeX) is available on the course website.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *that are described clearly*. Convoluted and opaque descriptions will receive lower marks.

Exercise 6-1. Do Exercise 17.1-1 on page 456 of CLRS.

Exercise 6-2. Do Exercise 17.2-1 on page 458 of CLRS.

Exercise 6-3. Do Exercise 17.3-1 on page 462 of CLRS.

Exercise 6-4. Do Exercise 17.3-3 on page 462 of CLRS.

Exercise 6-5. Do Exercise 17.3-4 on page 462 of CLRS.

Exercise 6-6. Do Exercise 17.3-6 on page 463 of CLRS.

Exercise 6-7. Do Exercise 17.4-3 on page 471 of CLRS.

Exercise 6-8. Do Problem 17-5 on page 476 of CLRS.

Problem 6-1. Amortization in Balanced Binary Trees

In a binary search tree, we say that a node x is a descendant of y if x is equal to y, or if the parent of x is a descendant of y. The size of the subtree rooted at y is equal to the number of descendants of y. Most self-balancing binary search trees (such as AVL trees and red-black trees) keep the tree fairly balanced, thereby ensuring that the tree has depth logarithmic in the number of elements. In this problem, we will develop an alternative self-balancing binary search tree using amortization.

Suppose that we have a binary tree augmented with subtree sizes. Define the *imbalance* of a node x to be the difference between the number of descendants of the left child of x and the number of descendants of the right child of x. More formally, we define:

$$imbalance(x) = |x.left.size - x.right.size|$$

If either child of x is null, then the imbalance is equal to the size of the non-null child. If both children are null, then the imbalance is equal to 0.

(a) Give an algorithm that takes as input a pointer to some node x in a BST and rebalances the subtree rooted at x. Your algorithm should run in $\Theta(\ell)$ time, where ℓ is the number of descendants of x. Your algorithm should not move any node in the tree that is not a descendant of x, but should reorganize or reconstruct the subtree rooted at x so that each node in the resulting subtree has imbalance at most 1.

(b) Suppose that you pick some constant c and modify your code for INSERT and DELETE so that you rebalance a node x as soon as $imbalance(x) > c$. Show that if you start with a tree of size n such that every node has imbalance 0, there exists a sequence of INSERT and DELETE operations such that the total runtime is not $O((\text{\# of ops}) \cdot \lg n)$. Explain why this implies that this rebalancing scheme is bad.

Clearly we need a more nuanced view of what it means for a node to be highly unbalanced. To that end, we say that the *imbalance ratio* of a node x is equal to the imbalance of x divided by the size of x. More formally:

$$imb\text{-}ratio(x) = \frac{imbalance(x)}{x.size} = \frac{|x.left.size - x.right.size|}{x.size}$$

(c) Show that any binary search tree in which every node has imbalance ratio at most $1/2$ has depth $O(\lg n)$.

Suppose that you modify your code for INSERT and DELETE so that you rebalance a node x as soon as $imb\text{-}ratio(x) > 1/2$. In the next two parts of this problem, you will show that under this rebalancing scheme the amortized runtime of INSERT and DELETE is $O(\lg n)$.

(d) Suppose that you have a BST of size n such that all nodes have imbalance at most 1. Use the accounting method to find the amortized runtime of a sequence of k INSERT and DELETE operations on this BST.

Hint: A rebalance of x occurs when $imb\text{-}ratio(x) > 1/2$. We can also write this condition as $imbalance(x) > x.size/2$. How many descendants of x must be inserted or deleted in order to cause the imbalance to become that high? And what is the cost of rebalancing x?

(e) Suppose that you have a BST of size n such that all nodes have imbalance at most 1. Use the potential method to find the amortized runtime of a sequence of k INSERT and DELETE operations on this BST.

Hint: Remember that when using the potential method, the final potential should always be at least as large as the initial potential, no matter what the sequence of operations is. Keeping in mind that an imbalance of 1 is, in some sense, as balanced as possible, is there a way to define your potential function so that the initial potential is 0, but the potential function is always non-negative?

Problem 6-2. House for Sale

You would like to sell your house, but there are very few buyers due to the housing market crash. Assume that at any given time there is at most one buyer interested in your house. You know with certainty that the range of offer from any buyer will be from 1 to M dollars. After getting an offer from each buyer, you make a decision to either take it or leave it. If you leave it, the buyer will buy some other house, so you will lose the offer forever. Note that the list of buyers is finite, and you do NOT know how many there will be. If you reject all of the offers, you will end up selling your house to the bank for 1 dollar. You want to maximize the return from selling your house. To this end, your knowledge of competitive analysis from 6.046 comes in very handy. Note that a clairvoyant algorithm will always pick the best offer.

(a) Give a deterministic algorithm that is $\frac{1}{\sqrt{M}}$-competitive. In other words, for any input sequence where x is the best offer, your algorithm should always choose an offer of at least $\frac{x}{\sqrt{M}}$ dollars.

(b) Devise a randomized algorithm that is expected $\frac{1}{2 \lg M}$-competitive. In other words, for any input sequence where x is the best offer, the expected value of the offer selected by your algorithm should be at least $\frac{x}{2 \lg M}$ dollars.

Hint: You may want to consider powers of 2 from 1 to M.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2012