# 15-750 — Graduate Algorithms — Spring 2006

**Miller and Derryberry**

**Assignment 1 Due date: Monday, February 6, 2006.**

**Some Reminders:**

- Read the Policies section on the course website before you start working on this assignment.
- You may work in **groups of size up to 3** for this problem set if you wish. However, **you should write up your solutions separately**. That is, collaboration should be limited to talking about the problems, so that your writeup is written entirely by you and not copied from your partner. In addition, state whether you worked alone and **list all collaborators**.
- Please refrain from consulting external materials when solving these problems.
- Please **submit both an electronic version, as well as a hard copy of your solutions** at the beginning of class on the due date.
- In all problems, it is implicit that you should show that your answer is correct, even when this is not explicitly stated.
- If you have questions, contact the course staff.

## 1 Binary Search Trees

(a) (5 points) Suppose we are storing income data for a set of people, and want to be able to draw random samples from the set meeting certain criteria. In particular, we are given pairs of the form $(name, income)$, and we want to perform queries of the form $randomSample(lower, upper)$, which returns, uniformly at random, a member of the set of people with incomes between *lower* and *upper* (inclusive). In addition, we want to be able to insert and delete $(name, income)$ pairs to and from the set of data. Show how to satisfy all these operations as efficiently as you can.

(b) (5 points) Suppose we are keeping track of changes to an integer counter, so that we can support the ability to query the value of the counter at any time in the past. Specifically, we are given pairs of the form $(++, t)$ and $(--, t)$, where $(++, t)$ means that the counter was incremented at time $t$. We must support the ability to add or remove such a pair from the history of changes to the counter, and to query the value of the counter at any point during the history. Show how to satisfy all of these operations as efficiently as you can.

## 2 Treaps and Red-Black Trees

In this problem, you will compare the performance of treaps with the performance of red-black trees for satisfying a query to a random element in the set of items that are stored. In all of the following parts, define the cost of a query to be the number of nodes touched on the access path of the queried node (querying the root costs 1). In this problem, you are not to use asymptotic notation when computing costs (i.e., it does not suffice to show that the expected cost is $\Theta(\log n)$). Your answers should be of the form $c \log_2 n$ plus/minus some lower order terms.

(a) (15 points) If we build a random treap containing all of the integers in $\{1, \ldots, n\}$, what is the expected cost of querying a random integer from $\{1, \ldots, n\}$?

(b) (15 points) We define a family of red-black trees as follows. Define $T_0$, the red-black tree of rank 0, to consist of just a root node, which is black. Define $T_r$, the red-black tree of rank $r > 0$, to consist of a black root with a red left-child and 3 red-black trees of rank $r - 1$ forming the 3 subtrees hanging off the root and its left-child (See Figure 1). Convince yourself that this is a valid red-black tree. How many nodes are in $T_r$? What

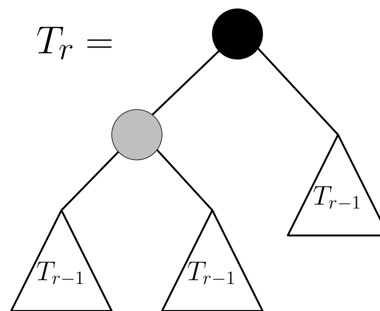


Figure 1: A visual depiction of the recursive definition of $T_r$. The gray node represents the red left-child of the black root.

is the expected cost, in terms of $n$, of accessing a random element of $T_r$, where $n$ is the number of nodes in $T_r$?

(c) (Extra Credit) Show how to create an even more costly red-black tree, where cost is defined, as above, as the expected cost of accessing a random element of the tree. Credit will be assessed in terms of the value of the coefficient of $\log_2 n$ in your expression for expected cost.

## 3 Random Trees

Let $\pi$ be a random permutation of the elements of $\{1, \ldots, n\}$. Let $T_\pi$ denote the tree that results from inserting $\{1, \ldots, n\}$ into an initially empty BST in the order of appearance in $\pi$ using the standard BST insert rules (with no rotations). A tree $T$ that is sampled from the distribution of all such trees, uniformly sampling over all permutations, is called a *random tree*. Define the *size* of a node in a BST to be the number of elements in its subtree (including itself). Denote the sizes of $1, \ldots, n$ by $s_1, \ldots, s_n$.

(a) (10 points) Prove that $\Pr[T] = \prod_{i=1}^{n} s_k^{-1}$.

(b) (10 points) Suppose we use the following insertion scheme (called $INSERT(i)$). After inserting element $i$ at a leaf using standard BST insert, traverse the access path upwards one node at a time. For each node $j$ encountered on this path, mark $j$ with probability $(1+s_j)^{-1}$. Rotate $i$ with its parent until $i$ rotates over the highest marked node on the access path (and erase all marks).

Let $T$ be a BST with root $w$, left-subtree $T_L$ and right-subtree $T_R$. Define $Join(T_L, T_R)$ to be the set of trees that can be created by rotating nodes with $w$ until $w$ is a leaf, and then deleting $w$. Thus, $Join(T_L, T_R)$ is the set $\hat{T}$ of trees such that inserting $w$ into $T' \in \hat{T}$ yields $T$ with some positive probability (convince yourself of this fact).

Prove that $\Pr[T] \cdot n = \Pr[T_L] \cdot \Pr[T_R] = \sum_{T' \in Join(T_L, T_R)} \Pr[T']$.

(c) (10 points) Use parts (a) and (b) to show that $INSERT(1), \ldots, INSERT(n)$ creates a random tree.

## 4 Double Cut Fibonacci heaps

In lecture, we considered Fibonacci heaps where non-root nodes may have at most one missing child. In this problem we would like to design and analyze heaps similar to Fibonacci heaps, but composed of *double-cut trees*, in which a non-root node may have up to two missing children.

(a) (5 points) Give pseudocode for the procedure cut that allows for up to two missing children.

(b) (10 points) Show that these double-cut trees have size exponential in their rank.

(c) (10 points) Give a potential based amortized timing analysis for your procedure cut.