Due Date: March 5, 2013

Problem 1: We wish to perform the following two operations on a set $X$ of real numbers:

- **INSERT($x$)**: first delete from $X$ all numbers not larger than $x$ and then insert $x$ into $X$.
- **FIND-MIN**: return the smallest element of $X$.

Describe a data structure that supports each of these operations in $O(1)$ amortized time. (Hint: Consider using a stack)

Problem 2: Given a binary search tree, add to each node $v$ an extra attribute $v.size$ indicating the number of keys stored in the subtree rooted at $v$. Let $\ell(v), r(v)$ denote the left and right child of $v$, respectively, and let $\alpha$ be a constant such that $1/2 \leq \alpha < 1$. A node $v$ is $\alpha$-balanced if $\ell(v).size \leq \alpha \cdot v.size$ and $r(v).size \leq \alpha \cdot v.size$. The binary search tree is $\alpha$-balanced if every node in the tree is $\alpha$-balanced.

In the following, assume that the constant $\alpha$ satisfies $1/2 < \alpha < 1$. Suppose that INSERT is implemented as usual for an $n$-node binary search tree, except that after every insertion, if any node in the tree is no longer $\alpha$-balanced, then we “rebuild” the subtree rooted at the highest such node in the tree so that it becomes 1/2-balanced. (Note: in this way, at most one “rebuild” is performed at each insertion or deletion.)

We use the potential method to analyze the above rebuilding scheme. For a node $v$ in a binary search tree $T$, define $\Delta(v) = |\ell(v).size - r(v).size|$, and define the potential of $T$ as

$$\Phi(T) = c \sum_{v \in T : \Delta(v) \geq 2} \Delta(v),$$

where $c$ is a sufficiently large constant that depends on $\alpha$.

1. Argue that any binary search tree has nonnegative potential and a 1/2-balanced tree has potential 0.
2. Suppose that $m$ units of potential can pay for rebuilding an $m$-node subtree. How large must $c$ be in terms of $\alpha$ in order for it to take $O(1)$ amortized time to rebuild a subtree that is not $\alpha$-balanced?
3. Show that inserting an item into an $n$-node $\alpha$-balanced tree costs $O(\log n)$ amortized time. (Hint: Refer to [Er:15] for a different analysis of this algorithm.)

Problem 3: Any skip list $L$ can be transformed into a binary search tree $T(L)$ as follows. The root of $T(L)$ is the leftmost node on the highest non-empty level of $L$; the left and right subtrees are constructed recursively from the nodes to the left and to the right of the root. Let’s call the resulting tree $T(L)$ a skip list tree.
(1) Show that any search in $T(L)$ is no more expensive than the corresponding search in $L$.

(2) Describe an algorithm to insert a new search key into the skip list tree in $O(\log n)$ expected time. Inserting key $x$ into $T(L)$ should produce exactly the same tree as inserting $x$ into $L$ and then transforming $L$ into a tree. (**Hint:** You will need to maintain some additional information in the tree nodes.)

**Problem 4:** Given a set of variables $\{x_1, x_2, \ldots, x_n\}$, an **equality** constraint is of the form “$x_i = x_j$” and an **disequality** constraint is of the form “$x_i \neq x_j$”. Describe an efficient algorithm that takes as input $m$ constraints (some equality and some disequality) over $n$ variables and decides whether all the constraints can be satisfied. For instance, the constraints

$$x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$$

cannot be satisfied.

**Problem 5:** Let $\{x_1, x_2, \ldots, x_n\}$ be a set of real numbers on the real line, describe an efficient algorithm that decides the smallest set of unit-length closed intervals such that each $x_i$ is in at least one of those intervals. Show the correctness of your algorithm (that is, the set of intervals output by your algorithm is indeed smallest possible) and analyze the running time.