Due Thursday 8.2.2007. Please keep a copy.

1. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of distinct items totally ordered such that $a_i < a_j$ iff $i < j$. Assume that over a long sequence of $m$ accesses to items in $A$, $a_i$ is accessed $q(i) \geq 1$ times. Describe an algorithm to find a binary search tree $T$ where each item $a_i$ resides in a leaf of $T$ such that if we serve the accesses using $T$, each time going from the root to the corresponding leaf, the total time its takes to serve the whole sequence is minimized. Prove 1) that your algorithm indeed constructs a tree that minimizes the total access time. 2) As tight upper bound as you can, on the total time it takes to serve the sequence using $T$. 3) An upper bound on the running time of your algorithm for constructing $T$. (Any polynomial time algorithm would be fine.)

2. We define the following variation on the splay algorithm. This variation looks 3 steps (edges) towards the root from the node $x$ and applies one of the rules in Figure 1 (or their mirror image) if possible. If it is not possible to apply one of the rules in Figure 1 we apply one of the regular zig-zig, zig-zag, or zig rules (Note that zig or zig-zig would apply only if $x$ is at distance 1 or 2 from the root, respectively).

Prove that the access lemma holds for this variation as well (with a different constant).

3. On a set of $n$ nodes we perform a sequence of operations, each of which is one of the followings.
   (a) $\text{insert}(u, v)$: Adds an edge between $u$ and $v$.
   (b) $\text{delete - oldest}$: Removes the edge that was inserted first among the edges currently in the graph.
   (c) $\text{connected}(u, v)$: Answers true if there is a path from $u$ to $v$ in the current graph. Otherwise answers false.
Describe a data structure that supports these operations and analyze its performance. (A structure supporting each operation in $O(\log n)$ amortized time would receive maximum score.)

4. We define the following data structure representing a string $s$ to answer $occ(\sigma, i)$ queries. Recall that $occ(\sigma, i)$ should return the number of occurrences of $\sigma$ up to and including position $i$. We put a binary tree $T$ over the symbols in $\Sigma$ (each symbol is at a leaf, the order does not matter). At the root $r$ we store a bit vector $B_r$ of length $n$ where $B_r[i] = 0$ if the character $s[i]$ is stored at the left subtree of $r$ and $B_r[i] = 1$ otherwise.

Similarly, for a node $v$ let $\Sigma_v$ be the set of characters in the subtree of $v$. Let $s_v$ be the substring obtained from $s$ by deleting all characters not in $\Sigma_v$. We store at $v$ a bit vector $B_v$ of length $|s_v|$, such that $B_v[i] = 0$ if the character $s_v[i]$ is in the left subtree of $v$ and $B_v[i] = 1$ otherwise.

a) Analyze the space required by this data structure.

b) Show how to add auxiliary information to $T$ without significantly changing the space taken by the data structure so that you can perform $occ(\sigma, i)$ queries efficiently. Describe the algorithm for answer an $occ(\sigma, i)$ query and analyze it.

5. Find a data structure to represent sorted lists such that 1) you can concatenate two sorted lists in $O(\log \log n)$ worst case time where $n$ is the length of the larger list, and 2) You can search for the predecessor of a key $x$ in a list of length $n$ in $O(\log n)$ worst case time.

6. Show how to extend dynamic trees to support the operation $lca(u, v)$. This operation assumes that $u$ and $v$ are in the same actual tree and returns the lowest common ancestor of $u$ and $v$ in their tree.

Try to get a running time of $O(\log n)$ without hurting the running time of the other operations.

7. We propose the following data structure for the union-find problem. We represent each set by a tree of depth exactly 2. Each leaf contains an item which is at distance exactly 2 from the root. Each internal node in the tree maintains 1) the number of children it has, 2) a pointer to its parent, 3) a pointer to a singly linked list containing its children. We maintain the following invariants:

1. Each node other than the root has between 1 and $2 \log n$ children.

2. Say a child of the root is full if it has at least $\log n$ children. At most one child of the root is not full.

A find returns the root and takes $O(1)$ time. We implement union of two sets with roots $x$ and $y$ as follows. Assume without loss of generality that $x$ has at least as many children as $y$. We make each full child $v$ of $y$, a child of $x$. (This requires inserting $v$ into the list of children of $x$, changing its parent to be $x$, and updating the number of children of $x$.) Let $x'$ and $y'$ be the non-full children of $x$ and $y$, respectively. If either $x'$ or $y'$ does not exist, we make the other non-full child the non-full child of $x$. Otherwise, assume without loss of generality that $x'$ has at least as many children as $y'$. We make every child of $y'$ a child of $x'$. Node $x'$ remains a child of $x$ which is possibly not full.

Prove that a sequence of $m$ union and find operations on a sequence of $n$ elements takes $O(m + n \log \log n)$ time.