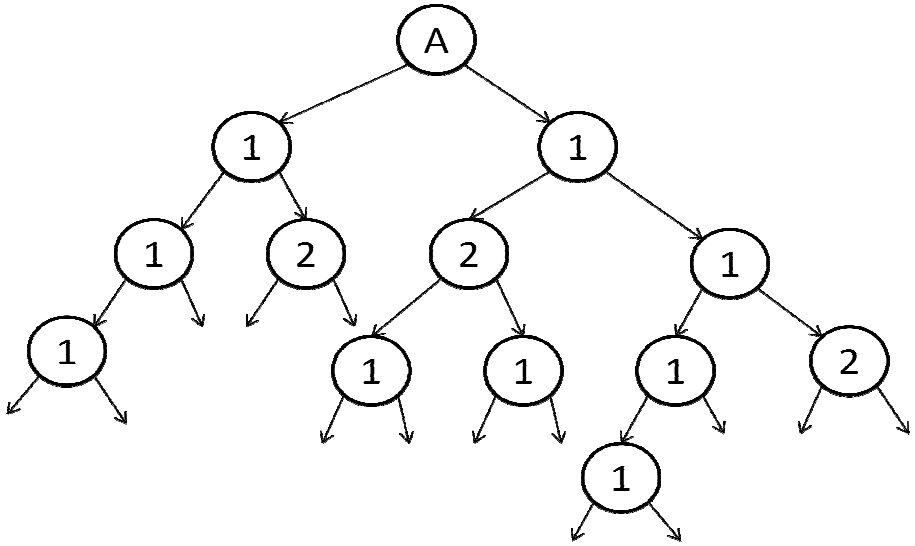
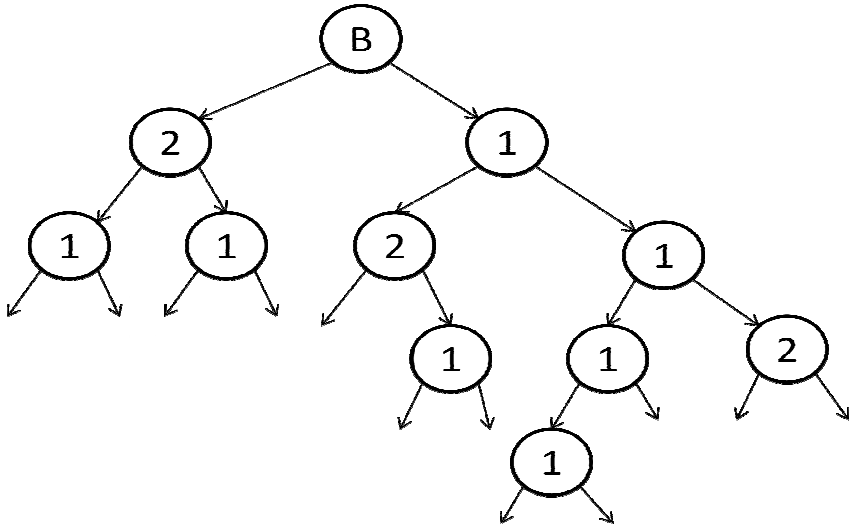
COS 423 Problem Set 2 Due Wednesday March 2, 2011 11 AM
Spring 2011
Tarjan
Collaboration allowed on Problem 3 only

1. The trees with roots A and B below are alleged to be AVL trees, with the numbers inside the nodes being alleged rank differences. (For simplicity, node items are omitted; A and B merely identify the trees.)
(a) Is tree A a valid AVL tree? If not, can it be made into a valid AVL tree by giving the nodes valid rank differences? If so how; if not, why not?
(b) Is tree B a valid AVL tree? If not, can it be made into a valid AVL tree by giving the nodes valid rank differences? If so how; if not, why not?





2. (Maintaining node sizes in binary search trees) Suppose we add to each node of a binary search a field containing its *size*, namely the number of nodes in its subtree including itself.
(a) Describe how to update sizes after an insertion or deletion in O(d + 1) time, where d is the depth of the inserted or deleted node. (In a deletion, if a swap occurs, the depth is the depth of the node to be deleted after the swap, i.e., the original depth of the node with which it is swapped.)
(b) Describe how to update sizes after a rotation in O(1) time.
(c) Using (a) and (b), describe how to maintain sizes in a balanced search tree such as an AVL tree so that the time for an insertion or deletion increases by at most a constant factor.
(d) Devise an algorithm that, given an item, uses sizes to find the number of items in an AVL tree smaller than the given item in $O(1 + \lg n)$ time. Note: a solution to this problem can be used to solve Problem 3 on Problem Set 1.

3. (a) Devise an algorithm to implement *split*(T, x) which, given a ravl tree T and an item x that may or may not be in T, splits T into two trees, T_1 and T_2, such that T_1 contains all items in T less than or equal to x and T_2 contains all items in T strictly greater than x, in $O(d + 1)$ time, where d is the depth of T. Verify the correctness of the algorithm and the time bound. Your algorithm is allowed to destroy T and reuse parts of it in the process of buiding T_1 and T_2.
(b) Consider a sequence of m intermixed access, insert, and split operations on a set of ravl trees, initially empty, where splits are done using your algorithm for part (a). Prove that each operation takes $O(\lg m)$ time, worst-case.

4. (Completion of the proof in Lecture 5 that an insertion into an n-node splay tree takes $O(1 + \lg n)$ amortized time) If x is a node in a binary search tree T, we define $\Phi(x) = \lg(s(x))$, where $s(x)$ is the *size* of x, the number of nodes in the subtree rooted at x. We define the potential of the entire tree, $\Phi(T)$, to be the sum of the potentials of its nodes. Suppose we do an insertion of a new item into T, without doing any rotations. This replaces a null child of one of the leaves of T by a new node containing the newly inserted item. The new node has a size of one, and the size of every other node along the path from the root to the new node increases by one. Prove that adding the new node increases the potential of T by $O(1 + \lg n)$, where n is the number of nodes in T after the insertion.