Intro to Algorithms, COMP-160, Homework #9
Benjamin Tanen, 04/06/2016

2. Let A and B be two given binary search trees storing the same n values.

   (a) **Outline an algorithm that transforms A into B only using rotations. Your algorithm should use $O(n)$ rotations in the worst case.**

   Our algorithm uses the principal that any tree can be transformed into a "line" all-right (or all-left) tree. In order to use this, take tree A and start at the root. From here, use the following until the algorithm halts:

   i. If the current node has a left-subtree, right rotate the node and go to the node's new parent (ex: if we were at the root before the rotation, go back to the root)
   ii. If the current node has no left-subtree, go to the right child
   iii. If the current node has no left-subtree and no right-subtree (is a leaf), end the algorithm

   At this point, we have an all-right tree A' that comes originally from tree A. Now repeat this operation for tree B and take note of all of the rotations necessary to transform tree B into tree B'. Since both tree A' and tree B' are "line" all-right trees, they are the same in shape now so tree A' = tree B'. With all of the B rotations noted, perform these rotations in reverse-order but do left-rotations instead of right-notations. This translates tree $A' = B'$ into tree B.

   Since tree A has n nodes in it, in the worst case, turning tree A into an all-right tree would require a rotation for every node, which would take $O(n)$ rotations. We then have to transform tree A' into tree B, which again takes $O(n)$ rotations in the worst case. Thus it takes $O(n) + O(n) = O(n)$ rotations in the worst case to go from tree A to tree B.

   (b) **Show that any algorithm that uses rotations for such a transformation must use $\Omega(n)$ rotations in the worst case. In other words, give an example of two trees that require a linear number of rotations, no matter what rotation-based algorithm is used. Explain why the bound holds.**

   Consider tree A that is a "line" all-left tree and tree B that is a "line" all-right tree. To transform tree A into tree B, we must right rotate every single node in A from root to leaf. Since there are n nodes, it takes $n-1$ rotations to go from tree A to tree B. These $n-1$ rotations are necessary regardless of the algorithm used (we ultimately will always need to right-rotate all $n-1$ nodes). Thus, this shows that at least $n-1$ moves are necessary for any algorithm in our worst-case scenario, showing that any algorithm takes $\Omega(n)$ rotations in the worst-case scenario.