|  |
| --- |
| **Programming Assignment 3 Warmup: size-balanced trees** |

|  |
| --- |
| **Definition (**size-balance property for a node) Consider a node *v i*n a binary tree with nodes in its left subtree and nodes in its right subtree; we say that *v*is *size-balanced* if and only if:  + 1  (so roughly, an imbalance of up to ⅓ - ⅔ is allowed)  **Definition:**  (size-balance property for a tree). We say that a binary tree *t*  **is size-balanced** if and only if ***all*** nodes *v* in *t* are size-balanced |
| Your implementation/modification of the BST implementation of the Dictionary ADT will enforce this size-balance property during insertions and deletions.  The rules are as follows: insertions and deletions are performed as usual except:  When a violation of the size-balance rule is detected, the subtree rooted at the violating node closest to the root is completely rebalanced (to be as balanced as possible). |
| **EXERCISES**: Get out pencil and paper and work with a neighbor.  Simulate an insertion sequence into an initially empty BST by a sequence of drawings. After each insertion:   * determine if a violation occurs. * if so,   + identify ***all*** nodes at which there a violation and   + re-balance the violating node closest to the root.   + (and draw the resulting tree).   **Exercise 1:** do the above for the following insertion sequence:  100, 50, 75, 60, 20, 90, 55, 70, 110, 65  **Exercise 2:** do the above for the following insertion sequence:  1, 2, 3, 4, 5, 6, 7, 8, 9, 10  **Exercise 3:** Build (and draw) a size balanced tree with 10 nodes which is as "tall" as possible (i.e., among all size-balanced trees with 10 nodes, construct one with maximum height). |