Project 3 Part B Analysis

Trevor Parsons

Part B Results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | N = 100 | N = 500 | N = 1000 |
| t = 5 | 11 | 19 | 21 |
| t = 10 | 12 | 18 | 21 |
| t = 15 | 12 | 17 | 21 |

Analysis

In part B, the *insert* function that was used to construct the random-value BSTs inserted duplicate values in the right subtree. Given random values, is unlikely that placing duplicate values in the right subtree made any difference in the average height of the BST compared to the option of placing duplicates in the left subtree.

The theoretical efficiency of the *treeHeight* function is based on the structure of the binary tree. In the best-case scenario, a random-value BST is as balanced as possible for a given number of nodes, making the minimum height = O(floor(lg(n))). In the worst-case, a random-value BST has nconnected nodes and none of the nodes have two children, making the maximum height = O(n – 1).

According to our book, the expected (average) height of a random-value BST is O(lg(n)). However, the experimental results from the program do not support this, as the average height for all BSTs with N-values of random nodes should be significantly lower. It is possible that achieving the theoretical average height of O(lg(n)) for a random-value BST (that places duplicates in the right subtree) is not reasonable in practice. Using an insert function that handles duplicates in a more efficient manner may lower the average height in practice. Still, the average heights from the program are much close to the expected values than the worst-case values, which was anticipated.

Example output from the program is on the next page, for values N = 100 and t = 5. The inorder traversal of the first BST (t = 1) is displayed to show that each node has a random value between 1-10,000. After the traversal, the heights of randomly generated BSTs are calculated and the average is taken of those heights.



