ENSF 593 Lab 11 - Bryce Shaw

How to run project

cd /BinarySearchTree/src

1. javac Main.java Record.java Tree.java TreeOps.java
2. java Main ../a3input1.txt ../output1.txt ../output2.txt

Picture of Resulting Tree

The output of the breadth-wise traverse (output2.txt) is a poorly formatted picture of the tree. It has all nodes printed line-by-line corresponding to their depth in the tree, but it lack justification to keep the structure of the tree (everything is squashed to the left side of the screen)

Complexity Analysis

1. Given that a binary tree splits into two node at each branch, the number of nodes (n) in a tree of height (h) would be n = 2^h. Inverting this expression, the height of a tree is O(log2(n))
2. A worst case input to this tree would be ascending or descending input. This would create a linked list with a height of O(n).
3. The depth first traversal recursively calls print functions for each node's right and left branches. This will result in a call stack in memory equal to the height of tree (O(log2(n)). At worst case this will be O(n) calls in the stack (for an ascending input).

   For the breadth first traversal, we will add right and left branches to the queue, while popping nodes from the front of the queue. In the case of an ascending input (tree is height O(n)), the queue will not exceed a size of 1 element... In the case of a perfect tree, the stack will pop all elements from the current level and the queue will only contain the elements in the next level down. This is at worst, a space complexity of ~O(n/2) ... This is because the summation of all the previous levels 1, 2, 4, 8, 16, 32 ... will always equal 1- 2^h where h is the bottom of the tree.

   The time complexity of both traverses is the same O(n) because we visit the same number of