Aaron Mcanerney
CS302
Dr. Harris
10/23/2017

Lab 09 Anaylsis

Analysis 1:

The height of the shortest binary is O(log(n)). This case occurs when the root node is the median of the data (and the data is evenly distributed on the left and right sides). For example, this occurs when the values: 9, 20, 5, 4, 6, 19, and 30 are inserted sequentially.
The resultant tree will be a full binary search tree with 7 items and a height of three.
Formally,

$$2^h - 1 = n \quad \textit{where n is the number of nodes in the tree and h is the height } (1 \textit{ based indexing})$$
$$2^h = n + 1$$
$$h = log_2(n) - 1$$
$$\textit{therefore, the height of a binary search tree is } O(log_2(n))$$

The worst case scenario is a height of O(n). This occurs when ordered data is entered sequentially. This is degenerative linked list case, where the link list exists to the right or the left of the root node. The length of a linked list is O(n), therefore the height of the tree is O(n), as well.

Analysis 2:

*This analysis assumes that the tree is the shortest possible height for all cases. If the tree was degenerated into a linked list, every operation would become order n.

Retrieve: O(log(h))

Retrieve is key based which means you are ignoring half of the tree on each movement downward. Anything involving division by two on each step is logarithmic.

Insert: O(log(h))

Insert is is key based which means you are ignoring half of the tree on each movement downward. Anything involving division by two on each step is logarithmic.

Remove O(log(h))

The way I have written remove (which is different than Harris's pseudocode) is O(log(h)). You find a node and then trace down to the bottom of tree with only one recursive call.

WriteKeys O(n)

In order to print every element in the tree you must “touch” every element in the tree. Every node is hit just once, therefore, it is of order n.