Description and Big O analysis

The binary tree search algorithm has two basic sections. The first is a component that builds the binary tree by prompting the user to enter an integer. The tree will be searched to find the appropriate place to insert the new item into. The algorithm is designed to exit the build tree functionality when a value of 0 is entered and then prompts the user to enter the search value. A binary search is conducted to find the search value. Because this is a binary search, the maximum number of search iterations will be equal to the height of the tree. I carefully watched the animation of Jeliot and found that the greatest height of the tree was 4. You can tell this by the number of addNode instances that are instantiated when adding a new integer.

Because the height of the tree will tend to be limited, the Big O of the binary search tree will tend on average for large n to be restricted to Big O(log n). The worst case will be O(n). In the use case provided we have an n = 14 and our binary search found the search value in 4 iterations. If the binary tree were to become unbalanced, then the average time would tend more towards O(n).

This Algorithm uses a linked list structure (although an array would work just as well) to implement the binary tree structure.