Course: Data Structures, Algorithms, and Their Applications – CPSC 319
Assignment 3 Complexity Analysis

Instructor: Leonard Manzara
T.A.: Xi Wang
Tutorial Section: T04

Student Name: Danielle Jourdain
UCID: 30114566

Question 1

If the records are inserted into the tree in a random order, the height of the tree will be $O(\log n)$. In all cases, the root of the tree will be chosen as the first element of the input data file. Since this case deals with insertion in a random order, it is safe to assume approximately half of the data will be greater than the root data, and approximately half of it will be smaller. There may be some cases where this is not true, but they will be discussed in Question 2.

When searching, after checking each node, the program will either move left or right depending on if the desired value is less than or greater than the current node data. Each time a decision is made to move to a new branch, approximately half the data is removed as a possibility. When at the root node, there are n potential spots where the data could be. After moving past the root, approximately $\frac{n}{2}$ possible spots exist for the data to be. Moving onto the next node gives $\frac{n}{4}$ possible spots. This pattern continues until the desired node is found or a leaf node is reached – meaning the desired data does not exist in the tree. This gives the overall complexity of $O(\log n)$.

Question 2

The worst-case complexity of the tree would occur when the tree devolves into a simple Linked List. This happens when most or all the data is either in alphabetical order or reverse alphabetical order. This means when searching, the worst case would require searching through n elements to find the desired one. This gives a worst-case complexity of $O(n)$. In the case of the assignment, the worst case would happen when the root of the tree is chosen as a last name starting with a letter near the start or near the end of the alphabet.

Question 3

Part 1 – Depth First, In Order

For a depth-first, in-order Binary Search Tree traversal, there is no dependence on the number of records in the tree or how they are organized within the tree. This means that checking each record in the tree the complexity will be $O(1)$. Since the depth-first, in order traversal is a recursive method, a new copy of the function will be created for each level of the tree. This means there will be $O(h)$ copies of the function open at one time where h is the height of the tree. Using the product rule on the two complexities found above gives an overall space complexity of $O(h)$.

Part 2 – Breadth First

Breadth-first traversal of a Binary Search Tree is different from the depth-first traversal. The breadth-first method uses a queue to keep track of the nodes at each level of the tree. This queue will be the largest when at the end of the tree, with all the leaf nodes. The number of leaf nodes will be 2^h when the queue is the fullest, where h is once again the height of the tree. At all other times, there will be less nodes in the queue since there are fewer nodes at each level as you move up the tree. Therefore, the space complexity for a breadth-first traversal of a Binary Search tree is $O(2^h)$.

Part 3 – Comparison

By looking at the Big-O space complexities of the two traversal methods, we can easily see that the depth-first, in order traversal is much more efficient since it only has $O(h)$ complexity while the breadth-first traversal has $O(2^h)$ complexity. Thus, the depth-first, in order traversal is much better for memory usage.