**CS303 Lab 8 – Binary Search Tree**

**Problem Specification:**

The problem given to us is to create the code for a binary search tree and run it against the given input and data files. We had to specifically test out two different functions. The first was the inorder traversal and the second being the searching function.

**Program Design:**

I have three classes: Node.java, BST.java, and Driver.java. The driver class just reads in all the files and creates nodes for the BST file to use. The BST file has three functions: TreeInsert(), inOrderTreeWalk(), and search(). The TreeInsert() function essentially creates the tree by appending a node to the previous node starting from the root each time. The first node to be added to the tree instantly becomes the root. The inOrderTreeWalk function starts at the root of the function and goes down the left and right side of each node and prints out the key. Finally, the search function goes through the entire tree and finds every element that is listed within the input.dat or KEYs.dat file. After finding the element, it prints out the time it took to find it and the key and description of the node. The final class I have is the Node class, which is essentially a class that holds the data for anything that is deemed to be a Node object. It holds the key and description of each node as well as its left and right node information.

**Testing Plan:**

In order to implement this, I created a system that allows the user to choose which file they would like to create a binary search tree out of. Then I run the inOrderTreeWalk and print out all the keys of the nodes within the tree. I have set inOrderTreeWalk to only run if you are using Data2.csv or Data3.csv due to Data1.csv and UPC.csv causing a stack overflow due to how many things it is printing. After it is done printing out all the keys, I ask the user if which input file they would like to use. The choices are the input.dat or the KEYs.csv. After calling the search on each of the keys within either file the program prints out how long it took to search for that key and prints out the key and its description.

**Text

Description automatically generatedTest Cases:**

**Text

Description automatically generated**

Both test cases show data set 1 being inputted into the tree and then both input file and key file used as search keys.

**Analysis/Conclusion:**

Based on the results I got for the first data file and the search keys, it seems logical for how long it took to get the results. The closer to the root, the smaller time it took to find and print out the key and its description. The higher the number, the longer time it took to find that key, due to the distance and number of checks that had to be completed to get to that node.

The time complexity of TreeInsert is only O(n) since there is only one loop. The time complexity is O(h), h being the height of the tree. The time complexity of search is O(n). The overall time complexity should be O(n) because it cannot take longer than all of the elements in the data files.

**References**:

The only references I must make are to the pseudocode and my previous lab in which I used a similar format for writing this report. Below I have added images of all my code for this lab.

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer screen

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated