Starting the program, I defined a check to see if it is a binary tree. I then defined predicates for the empty binary tree, and the non-empty binary tree. For the non-empty binary tree, I implemented comparisons that would ensure that the elements of the left subtree are less than/equal to the root node, and that the elements of the right subtree are greater than the root node. I then moved on to implement the preorder traversal. To do so, start by visiting the root node, then visiting each node of the left subtree, and then each node of the right subtree. As the tree is traversed, every node that is visited gets appended to a list *L*. I then moved on to implementing the inorder traversal. To do so, start by visiting each node of the left subtree, then the root node, then each node of the right subtree, and append each node to a list *L*. I then implemented the postorder traversal by visiting the left subtree, then the right subtree, then the root node, appending each visited node to a list *L* as the tree is traversed. After implementing the traversals, I implemented the search. This is achieved by recursively checking the node, left subtree and right subtree for the element *I*. Following this is the height implementation. I did this by first checking if the tree was empty, and if it was asserting the height to 0. I then checked for if the tree contained 1 element, and if so, I asserted the height to 1. Otherwise, the height of the binary tree can be obtained by getting the maximum height of the left or right branches and adding the root node. I then implemented the insert by checking if the new element *I* is less than or equal to the root node. If it is, we insert it into the left tree, otherwise we insert it into the right tree.