Homework 3

Due Monday, July 23rd at 11:59pm

Name: Zachary McNulty
Student ID: 1636402

Problem 3: Binary Search Trees

(i)
In-order traversal: a,b,c,d,e,f,g,h,i

(ii)
pre-order traversal: f,c,b,a,e,d,h,g,i

(iii)
post-order traversal: a,b,d,e,c,g,i,h,f

Interesting Property: The in-order traversal of a BST returns the elements in sorted order (from lowest to highest)

b)

Firstly, add the field "position" to the IntTreeNode class. This data field keeps track of the position of the node if the tree were to be sorted. For example, if a node has position value 3, that means the data in that node is the 3rd smallest element within the tree. Maintaining and updating this field to keep it accurate does not change the complexity of the add() or remove() method. Whenever you are placing a new node, if you move down a node's left subtree, increment that node's position by 1 (because that node now has now another node to the left of it, and thus smaller). You do not have to change the position if you move down a node's right subtree. When you reach the bottom of the tree, assign the node being added the parent's position minus one if it is the parent's left child, and the parent's position plus one if it is the right child. I accomplished this by adding a new parameter to the add() method, position, which keeps track of the node's theoretical position if it were to be placed at that given time. As the only new operation is incrementing an int, a constant time operation, there is no change in complexity. A similar strategy can be implemented in remove as well

Once you have this extra data field implemented, traversing the tree is much more organized. If we are looking for the kth smallest element, simply observe the current node's position. If it is equal to k, we are done: return the data. Else if the current node's position is less than k, move down the right subtree as the number we are looking for must be higher than it. Else if the current node's position is greater than k, move down the left subtree as the number we are looking for must be below it. Then just repeat until the kth smallest element is found.

As we only move down only a single subtree at each level, we search through at most h nodes, where h is the height of the tree. As we are only doing constant time comparisons at each recursive level, the overall complexity of this algorithm is $O(h)$.