Ramaneek Gill 1000005754

1)

I will modify the Insert and Delete algorithm structure to make the AVL tree always have an instance variable that keeps track of the minimum key in the tree so returning it will take constant time.

```
Min(S):
        return S.min

Insert(root, x):
        If root is NIL:
                Root = Node(x)

//////////////////////////////////////////////////
//Changed code:
                if(min_node == NIL or root.key < min.key):
                        min = root
//////////////////////////////////////////////////

        Else if x.key < root.item.key:
                Root.left = INSERT(root.left, x)
                If root.left.height > root.right.height + 1:
                        Root = REBALANCE−LEFT(root)
                Else:
                        Root.height = 1 + max(root.left.height, root.right.height)

        Else if x.key > root.item.key:
                Root.right = INSERT(root.right, x)
                If root.right.height > root.left.height + 1:
                        Root = REBALANCE−RIGHT(root)
                Else:
                        Root.height = 1 + max(root.right.height, root.left.height)
        Else:
                Root.item = x
        Return root
```

It follows a similar method in the DELETE process. When deleting a node check if it is the minimum node. If it is the minimum node we make the new minimum node the minimum of either the parent of the node being deleted or the right child of the parent node. Since the tree is an AVL tree we know that the smallest key in the tree exists in the node that is furthest to the left. If it is being deleted we know that the second smallest node is it's parent or the parent's right child if a rotation has to occur.

The new augmented AVL data structure can now return the minimum node in the tree in constant time, however the insert and delete operations will now to slightly longer to run but since we are only focused on big theta notation their run time is not affected by the augmentation.