Ramaneek Gill   1000005754
Sept/29/2014

CSC263 Problem Set 3

A)
If keys match I will be inserting them as the right child.

```
TREEINSERT(root, x):
    If root is NIL:
        BSTNode(x)     //x is a node (with NIL for children)
    Else if x.key < root.key
        root.left = TREEINSERT(root.left, x)
    Else if x.key > root.key
        root.right = TREEINSERT(root.right, x)
    Else  //x.key==root.key
        root.right = TREEINSERT(root.right, x)
    Return Root
```

The runtime for TREEINSERT is normally Θ(h) where h is the height of the tree. However what we are doing in this case is almost like inserting a linked list somewhere into the tree, the list will have n identical keys and will be always the right child. The total time taken by all n calls to TREEINSERT is the sum of Θ(h + i) for i=1 to n.

Therefore the total time taken by this algorithm to insert n identical pairs is Θ(h(n(n+1))/2).

B)

```
TREEINSERT(root, x, bool_go_left):
    If root is NIL:
        BSTNode(x)     //x is a node (with NIL for children)

    Else if x.key < root.key
        root.left = TREEINSERT(root.left, x, bool_go_left)
    Else if x.key > root.key
        root.right = TREEINSERT(root.right, x, bool_go_left)

    Else //x.key == root.key
        if bool_go_left == False:
            root.right = TREEINSERT(root.right, x, !bool_go_left)
        Else:
            root.left = TREEINSERT(root.left, x, !bool_go_left)
    Return Root
```

When the user first calls this function they will pass a boolean called bool_go_left, this boolean will determine whether a key of identical value should be inserted as the left child or the right child. Since it's value is alternated on each call it will place identical keys as evenly as possible as potential left or right child nodes.

For every two elements inserted into the tree the height of it is increased by one. Since TREEINSERT normally runs at $\Theta(h)$ time where h is the height of the tree this algorithm will have a total running time of the sum of $\Theta(h+i/2)$ for i=1 to n.

Therefore the total time taken by this algorithm to insert n identical pairs is $\Theta(h(n/2(n/2+1))/2)$

C)

The keys for each node will be implemented in a linked list, this way one node can have multiple keys and this will prevent the tree from increasing in height. Root and X are nodes, root.head is the node holding the linked list of keys. This linked list will always be sorted since the elements are all identical.

```
TREEINSERT(root, x):
    If root is NIL:
        BSTNode(x)     //x is a node (with NIL for children)
    Else if x.key.value < root.key.value
        root.left = TREEINSERT(root.left, x)
    Else if x.key.value > root.key.value
        root.right = TREEINSERT(root.right, x)
    Else  //x.key==root.key
        temp_node = root.key.head  //makes x the new head of root.key and appends
the rest of the list to x
        root.key.head = x.key
        root.key.next = temp_node
    Return Root
```

WORST CASE ANALYSIS:

TREEINSERT will run at $\Theta(h)$ where h is the height of the tree. Since for every identical key inserted in to the tree the tree's height will never change TREEINSERT will run at $\Theta(h + c)$ where c is the constant time it takes to update the head of the linked
list. So the worst case analysis of TREEINSERT will be $\Theta(h)$ since the height will never change for identical key insertions.