**Sam Lazrak**

**CS 303 Algorithms and Data Structures**

**Homework Assignment 7**

**2/27/18**

1. Work the following Exercises from Chapter 12 of the text:
   1. (2 points) Exercise 12.1-4 on page 289.

* PREORDER-TREE-WALK(x)

if x NIL

then print key[x]

PREORDER-TREE-WALK(left[x])

PREORDER-TREE-WALK(right[x])

* POSTORDER-TREE-WALK(x)

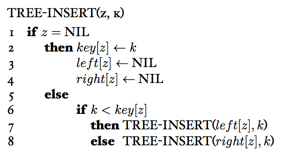
if x NIL

then POSTORDER-TREE-WALK(left[x])

POSTORDER-TREE-WALK(right[x])

print key[x]

* 1. (2 points) Exercise 12.3-1 on page 299.



(Gzc)

1. Solve the following problems from Chapter 12 (not in the book):

As suggested in class, here is the pseudocode to balance an “unbushy” tree:

TREE-BALANCE (T)

1. create array A(N–1)

2. for i = 0 to N–1

3. x = T.root

4. A[i] = x

5. TREE-DELETE(T, x)

6. for j = 0 to N–1

7. TREE-INSERT(T, A[j])

Assume that the tree, though a little unbalanced, is “not that bad,” such that at the end of the for loop in lines 2-4 the values in the array are pretty well randomly distributed.

1. (4 points) Perform an asymptotic analysis of TREE-BALANCE to determine its running time in terms of N. Write out each step in detail. What is the cost of TREE-BALANCE in additional storage space?

TREE-BALANCE (T)

1. create array A(N–1) c1 1

2. for i = 0 to N–1 c2 n-1

3. x = T.root c3 1

4. A[i] = x c4 1

5. TREE-DELETE(T, x) c5 lgn

6. for j = 0 to N–1 c6 n-1

7. TREE-INSERT(T, A[j]) c7 lgn

T(n) = c1+ c2(n-1) + c3 + c4 + c5 (lg(n))+ c6 (n-1) + c7 (lg(n))

T(n) = (c1+ c3 + c4+ c2 + c6) \* (c2+ c6)n \* (c5+ c7)(lg(n))

T(n) = (n\*lg(n)) + (n\*lg(n))

T(n) = O(nlg(n))

In additional storage space, the cost of TREE-BALANCE will be O(n).

1. (2 points) What is the worst case of a Binary Search Tree? What happens to the running time performance of TREE-BALANCE in this case? What is the overall outcome of the shape of the tree in this case?

* The worst-case of binary search tree is when the tree is already balanced which means the worst-case time complexity is O(n) since TREE-BALANCE will run equal to the height of the tree. The running time performance of TREE-BALANCE will be O(n) and the shape of tree will have the left side of the tree empty and the right side of the tree will be balanced.

1. (5 points) In the discussion of QUICKSORT we saw an alternative PARTITION function, called RANDOMIZED-PARTION. Thinking about this, how could you devise a helper function to insert between lines 5 and 6 in the above pseudocode that would help address the problem of really badly balanced trees? What would this do to the overall running time of TREE-BALANCE?

* A helper function can randomize the order of the elements to avoid the problem of running into the worst-case running time of O(n). A randomizer helper function will make it nearly impossible to get the worst-case. The overall running time of TREE-BALANCE will still be O(nlg(n)) but the worst-case will now be avoided.

1. (5 points) Devise another approach to TREE-BALANCE to guarantee a very well balanced binary search tree as the result. Analyze the running time performance of this approach to Tree-Balance.

* Another approach is to use recursion and when inserting elements, you are to insert it at the median place. So every time you insert an element, find the median and insert the element, and recursively call the method for the part before and after the index value.  The running time analysis will be the same as TREE-BALANCE except the TREE-INSERT method will run in linear time which does not change the running time of O(nlg(n)).

1. Work the following Exercises from Chapter 13 of the text:
   1. (2 points) Exercise 13.2-1 on page 314.

* RIGHT-ROTATE (T, x)

y = x.left //set y

x.left = y.right // turn y’s right subtree into x’s left subtree

if y.right ≠ NIL

y.right.p = x

y.p = x.p // link x’s parent to y

if x.p == NIL

T.root = y

else if x == x.p.right

x.p.right = y

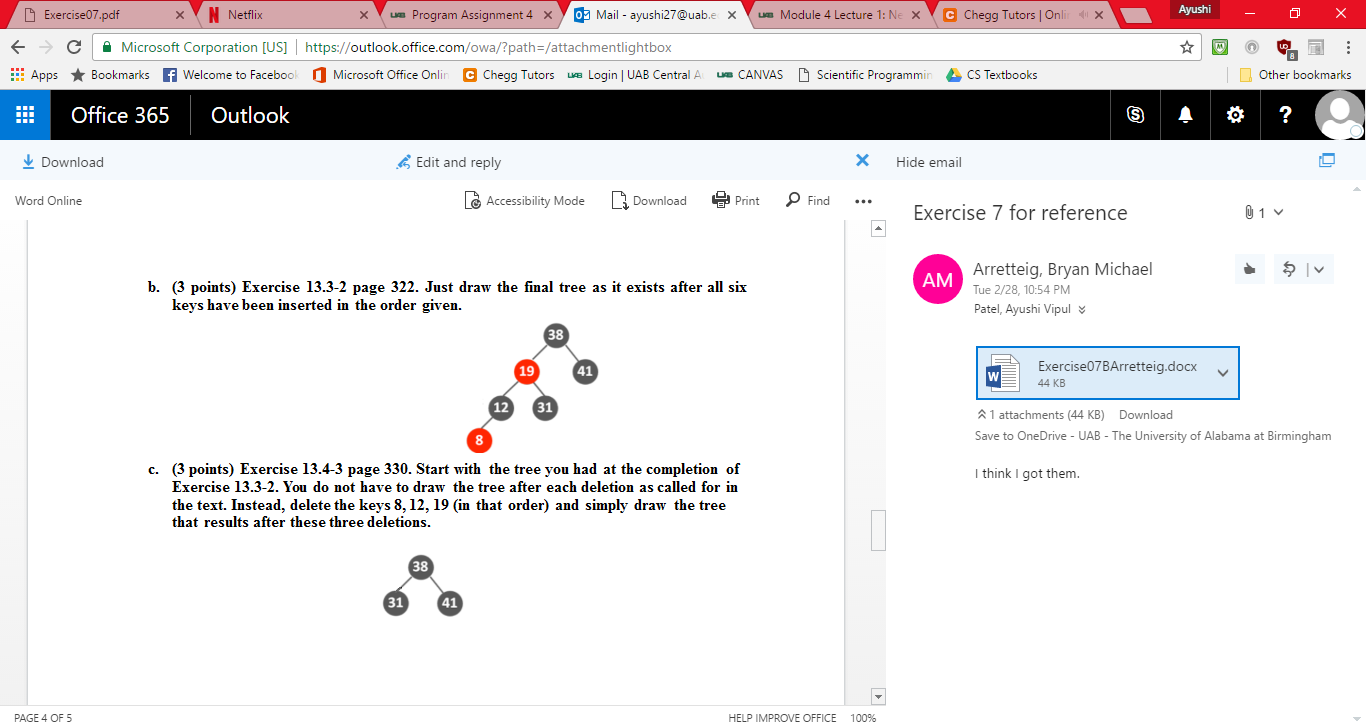
else

x.p.left = y

y.right = x //put x on y’s right

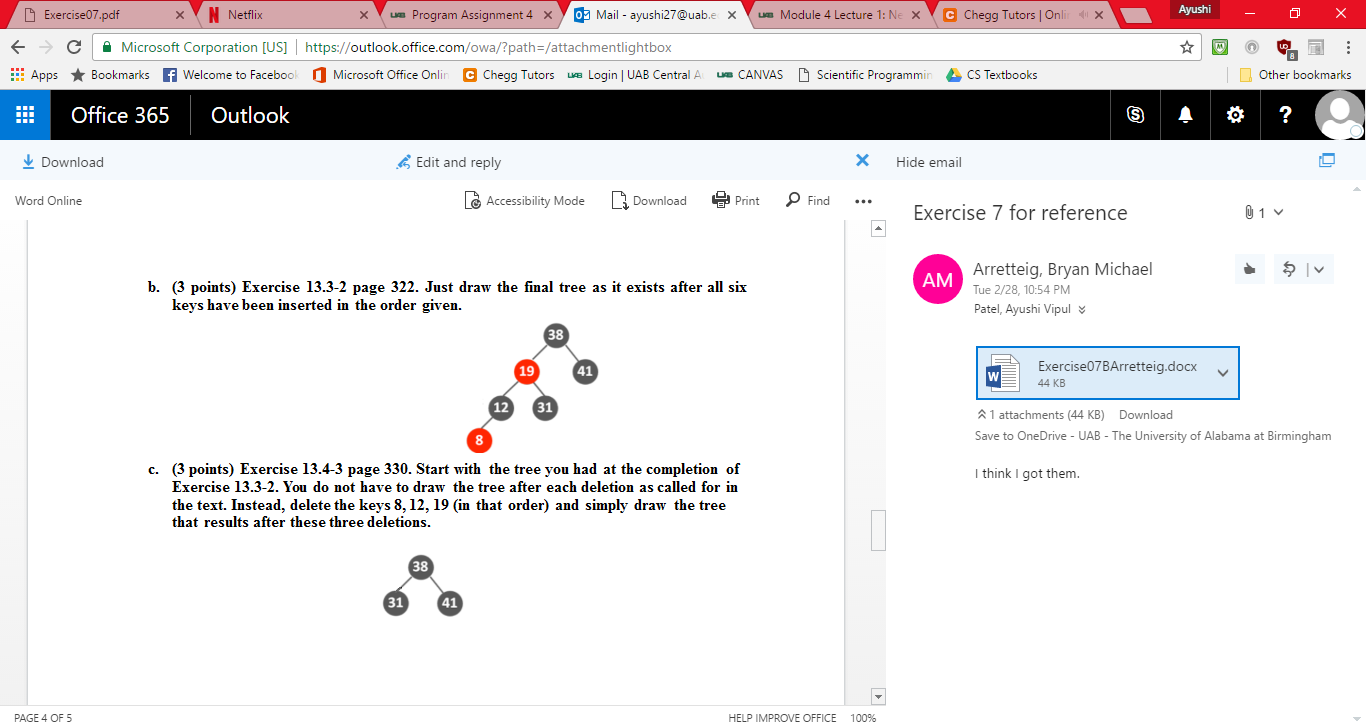
x.p = y

* 1. (3 points) Exercise 13.3-2 page 322.

Just draw the final tree as it exists after all six keys have been inserted in the order given.

* 1. (3 points) Exercise 13.4-3 page 330.

Start with the tree you had at the completion of Exercise 13.3-2. You do not have to draw the tree after each deletion as called for in the text. Instead, delete the keys 8, 12, 19 (in that order) and simply draw the tree that results after these three deletions.



Works Cited

Gzc. " C12-Binary-Search-Trees." *GitHub*. N.p., n.d.