CS163 Feedback for Lab #: 7 Page #1 Name:_____

Step	Measurement	Expected	Assessment
Step 1a	• Count the number of nodes with no children	• Base cases: ○ **if (root == NULL) return 0;** ○ **if (root->left == NULL && root->right == NULL) return 1;**	
Step 1b	• Solve for the next smaller subproblem	• Uses the returned value from the function **num = count(root->left) + count(root->right);**	
Step 1c	• Compare the values of left and right	• Returns 1 if root->left && root->right are both NULL	
Step 1d	• Return the counter	• Returns the sum of recursive calls via the variable num ○ **return num;**	

Test Case(s)	Expected Result	Verified? (yes/no)
1. Empty Tree	**Count should be zero**	Call count before inserting data into the tree
2. Only 1 node (root)	**Count should be 1**	Physically add just one node in a test function and call count
3. Larger tree, some nodes have 1 child, some nodes have 2 children	**Double check that the count is correct**	Use the tree provided and verify

Step 2a	• Calculates sum	• Base case: **if (root is NULL) return 0**	
Step 2b	• Solve for the next smaller subproblem	• Call the function recursively, **value = sum(root->left) + sum(root->right);**	
Step 2d	• Perform the operation	• Add the data to the value returned. You could say: **return root->data +sum(root->left) + sum(root->right);**	

Test Case(s)	Expected Result	Verified? (yes/no)
1. Empty Tree	**Sum should be zero**	Call sum before inserting data into the tree
2. Only 1 node (root)	**Sum should just be root's data**	Physically add just one node in a test function and call sum
3. Larger tree, some nodes have 1 child, some nodes have 2 children	**Double check that the sum is correct**	Use the tree provided and verify

Step 3a	• Determine the height of the tree	• Base case: if (root is NULL) height is zero	
Step 3b	• Solve for the next smaller subproblem	• Uses the returned value from the function **left_height = height(root->left);** **right_height = height(root->right);**	
Step 3c, 3d	• Computes the height by first finding out which height is larger	• It is the maximum of the left or right heights, • Add one for the current node **height = MAX(left_height, right_height) +1;**	
Step 3d	• Returns the height	• Returns the computed height	

Test Case(s)	Expected Result	Verified? (yes/no)
1. Empty Tree	**Height should be zero**	Call height before inserting data into the tree
2. Only 1 node (root)	**Height should be one**	Physically add just one node in a test function
3. Larger tree, but all nodes are on the right side of the root, no left children	**The height should be the same as the number of data items entered**	Insert the data in sorted order
4. Larger tree, some nodes have 1 child, some nodes have 2 children	**Double check that the height is correct**	Use the tree provided and verify

Step 4a	• Remove all	• Base case: **if (root is NULL) return 0**	
Step 4b	• Solve for the next smaller subproblem	• Call the function recursively, **value=remove_all(root->left)+remove_all(root->right);**	
Step 4c	• What should be done before the recursive call?	• Nothing beside the base case…	
Step 4d	• Delete after the recursive call	• Deallocate the memory for the node! **delete root; root=NULL;**	
Step 5a	• Makes a complete copy	• Base case: if (root is NULL) destination pointer should be set to NULL, returning 0;	
Step 5b	• Solve for the next smaller subproblem	• Call the function recursively, using the returned value **value = copy(destination->left, source->left);** **value += copy(destination->right, source->right);**	
Step 5c	• What should be done before the recursive call?	• Copy the node **destination = new node;** **destination->data = Source->data;** **destination->left=Destination->right=NULL;**	
Step 5d	• Returns the value	• Returns the value (add one if you also want to count the number of nodes!)	