Chapter 8

1. The following questions refer to the tree of Figure 8.1.3
   1. Which node is the root?
      1. /user/rt/courses
   2. What are the internal nodes?
      1. /user/rt/courses
      2. CS016/
      3. Homeworks/
      4. Programs/
      5. CS252/
      6. Projects/
      7. Papers/
      8. Demos/
   3. How many descendants does node cs016/ have?
      1. 10
   4. How many ancestors does node cs016/ have?
      1. 1
   5. What are the siblings od node homeworks/
      1. Grades
      2. Programs/
   6. Which nodes are in the subtree rooted at node projects/?
      1. Projects/
      2. Papers/
      3. Buylow
      4. Sellhigh
      5. Demos/
      6. Market
   7. What is the depth of node papers/?
      1. 3
   8. What is the height of the tree?
      1. 4

5) Describe an algorithm, relying only on BinaryTree Operations, that counts the number of leaves in a binary tree that are the left child of their respective parent.

Function int leftCheck(Node root):

Int count = 0

If(root.left == null AND root.right == null)

Return 0;

If root.left is not null AND root.left.left is null AND root.left.right is null

Count = 1;

If root.left is not null

Count = count + leftCheck(root.left)

/\* We could also adjust this so that this function doesn’t

Happen if the above is true because that would mean

That inherently a left lead doesn’t have further left

Nodes to check. But for simplicity, I did not implement\*/

If root.right is not null

Count = count + leftCheck(root.right)

Return count

This algorithm should recursively call down the tree and add 1 to the total every time there is a left leaf node. It will then add up all the left leaf nodes that were under both the right and left node up once it gets back to the first call.

6) Let