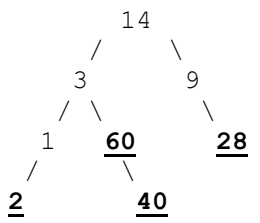
**1)** (10 pts) DSN (Binary Trees)

The goal of a function named *legacyCount()* is to take the root of a binary tree (*root*) and return the number of nodes that contain a value greater than at least one of their ancestors. For example, this function would return **4** for the following tree, since **60** is greater than both of its ancestors (3 and 14), **40** is greater than two of its ancestors (3 and 14) (even though 40 isn't greater than its parent!), **28** is greater than both of its ancestors (9 and 14), and **2** is greater than one of its ancestors (1).




Our node struct is as follows:

```
typedef struct node {
   int data;
   struct node *left;
   struct node *right;
} node;
```

To make the code work, *legacyCount*() is a wrapper function for a recursive function called *legacyHelper*(). Included below is the code for legacyCount() as well as the function signature for *legacyHelper*(). Write all of the code for the *legacyHelper*() function. Note: If *root* is NULL, you should return 0.

```
int legacyCount(node *root) {
    if (root == NULL) return 0;
    return legacyHelper(root->left, root->data) +
           legacyHelper(root->right, root->data);
}

int legacyHelper(node* root, int minAncestor) {

      if (root == NULL)
            return 0;

      if (root->data > minAncestor)
            return 1 + legacyHelper(root->left, minAncestor) +
                       legacyHelper(root->right, minAncestor);

      return legacyHelper(root->left, root->data) +
             legacyHelper(root->right, root->data);

}
```

**Grading**:

+ 2 pt for correct root == NULL base case in recursive function
+1 for if checking root->data vs. smallest Ancestor
+3 for returning the right result in this case (1 pt for 1, 1 pt for each rec call)
+4 for return in other case, 1 pt for each rec call and 1 pt for updating the second parameter in both calls.