Technical Question 4:

This question essentially asks, what is the first node (preferably below the root) that connects the two nodes given. In other words, the intersection of the two nodes. I believe an efficient way to handle this problem is to traverse each node level in succession. The way in which we will travel down those nodes is based on the value of the current node we are visiting. Since Binary Search Trees node values are always increasing in the right-hand direction, we can use this to speed up our search.

I am using the Node class from the class tutorial. I have created three methods: find\_left and find\_right are decorator methods for the find\_children method. The latter returns an array of the left and right child nodes of any given node. The wrapper methods narrow down the result to the specific direction based on the value returned (0 is left, 1 is right with None being the default).

We begin with the root node. From here we set a condition to keep looping until we run into a leaf (where find\_children returns None, None).

If we pass this base condition, we check to see if the value of node\_index (here set to the value of the current node being examined) is less than both the comparing nodes (n1 and n2). If so, we can then safely turn to the right and examine that node.

If instead, the node\_index is larger than both the comparing nodes (n1 and n2) and we can look at the child left node.

If neither of these conditions is true, it must mean that we have found our node. We return this node’s value as the LCA (least common ancestor) of the two nodes.

If the tree is balanced, we will have an efficiency of O(log(n)) since we will need to traverse the depth of the tree. If unbalanced, we will need to traverse each element, thus bringing the efficiency to O(n). At worst case, we will need to traverse the depth of the binary tree, thus at worst the efficiency would be O(log(n)). The time efficiency is O(log(n)). The space complexity is O(n) since we need to capture an array of length n.