CSE143 Lecture Questions for Wednesday, 5/19/21

Question	Answer
You mention that every interesting binary tree problem that we encounter will need a public and private pair. How do we know if the problem does not require it? What should we look out for? That answers my question, thank you!	You're not going to encounter one where you don't need it unless it's something really trivial like isEmpty. If you have to traverse the tree, then you'll want a public/private pair to at least have a root parameter.
If you had instead done (value < root.data) goes to the left and (value >= root.data) goes to the right, would the output still be in alphabetical order when you do an in order search? In other words, does it matter which one is <= or >=? Ok thank you.	It doesn't matter whether duplicates go to the left or the right as long as you are consistent.
I'm not sure if you go into this later in the lecture, but what happens if you do preorder or postorder rather than inorder? Would the only thing that changed be that Twyla would move in the order? Ah ok. It wouldn't be alphabetical anymore. Thank you!	There is nothing special you'd see with a preorder or postorder traversal of a binary search tree. No, it would affect the entire sequence because we'd be doing the same thing at every level, not just at the top of the three. That's right...it wouldn't be in alphabetical order.
Seeing as LinkedLists could be traversed using loops, is a bad solution to BinaryTree problems to use while() loops and a reference to the current node as well? I see. Keeping track of traversed values wouldn't be possible. Thanks!	Trying to write loop-based solutions for binary trees tends to lead to very complicated code. The recursive solutions we are encouraging are much simpler in many cases. You'd also need an additional data structure like a stack to solve them without recursion.
Could we just do root.left = new IntTreeNode to get around the x assign change x problem? Or would that be looking too far ahead for a recursive method? So it'd be better to just use the recursive method with the x assign change x instead of trying to get around the issue?	You could add code like that, but it would have to be duplicated three times because sometimes you are resetting overallRoot, sometimes you are resetting root.left, and sometimes you are resetting root.right. That's a lot of redundancy and when you do something three times, it increases the odds that you'll introduce a bug in one of them. Yes, using x=change(x) with recursion leads to much simpler code that is easier to verify.