Conder Shou
cs3544
Data Structures in Java

Homework 3: Written

1.)

Prove that *n* full nodes plus 1 = number of leaves in a binary tree

0 full nodes + 1 = 1 leaf

This is true, no full nodes means it's either a degenerate tree (a line of numbers), or a single node. In both cases, there is only one leaf in the tree.

Assume true for k = 1 to *n* full nodes

n full nodes + 1 = *n* + 1 leaves

(*n* + 1) full nodes + 1 = *n* + 1 + 1 leaves

n + 2 = *n* + 2 leaves

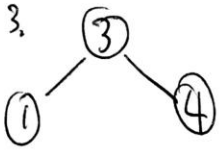
As shown above, this also holds true for *n + 1* full nodes.

2.) 4.9

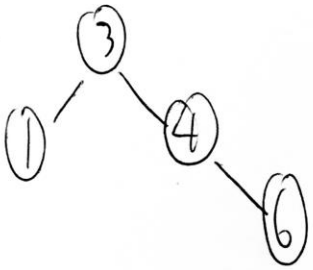
a) 1.

2.

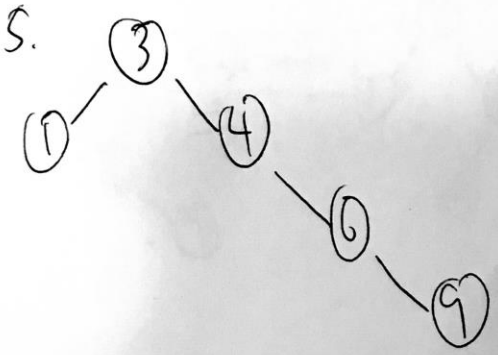
3.



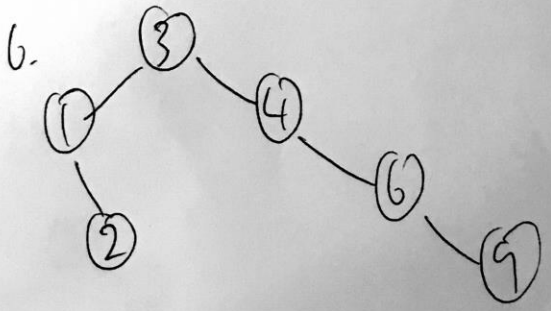
4.



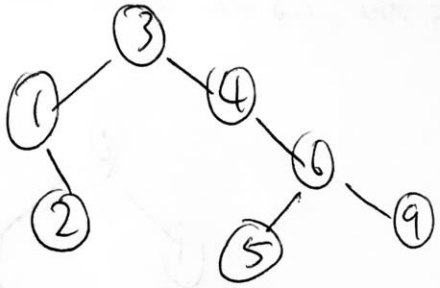
5.



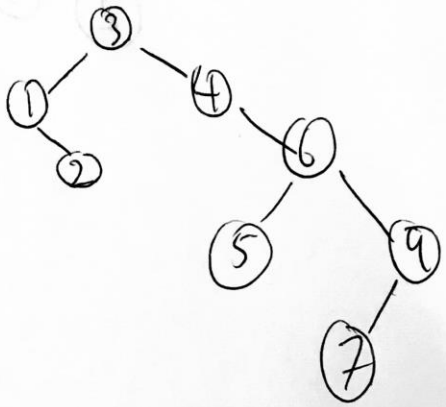
6.



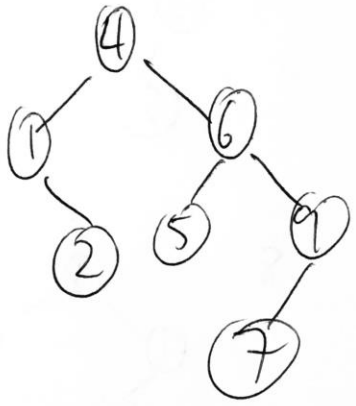
7.



8.



(b) We replace the root node with the smallest node in the right subtree, and then delete that node (from the right subtree).

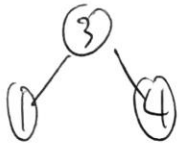


3.)

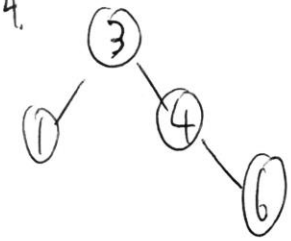
1.

2.

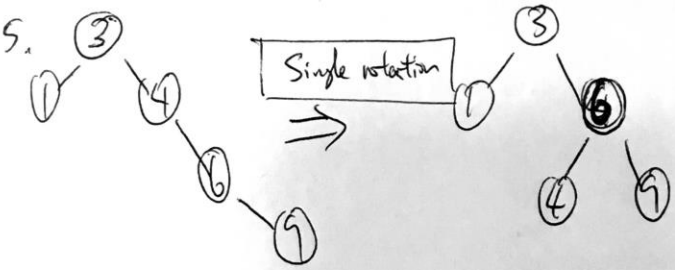
3.



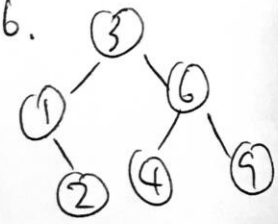
4.



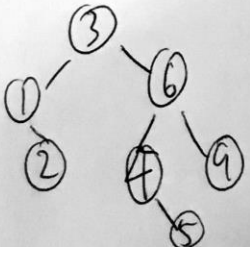
5.



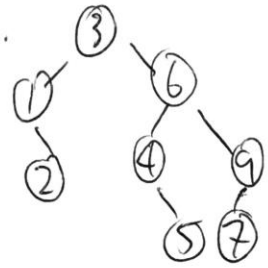
6.



7.



8.



4.)

Pre-Order Traversal: 1, 2

Post-Order Traversal: 2, 1

Pre-Order Traversal: 1, 2

Post-Order Traversal: 2, 1

Both of these different trees have values that only occur once, yet both trees have the same pre-order and post-order traversals.

Therefore, it is proven that given both the preorder and postorder traversal of some binary tree t. The information **IS NOT** necessarily sufficient to reconstruct t uniquely, even if each value in t occurs only once.

5.)

In order to implement lazy deletion, I would add a fourth parameter to the BinaryNode class, perhaps called “deleteMk”, and that would be assigned to the instance Boolean variable “deleteMark”.

When creating a node without any left or right subtrees, we would then automatically set this fourth parameter to “false” in “this(theElement, null, null, false)”.

In “findMin”, we would then have the following code:

```
private BinaryNode<AnyType> findMin( BinaryNode<AnyType> t)
{

        if ( t == null )
        {
                return null;
        }

        if (t.left != null)
        {
                return findMin(t.left);
        }

        if (t.left == null && !t.deleteMark)
        {
                return t;
        }

        // this is basically else if (t.left == null && t.deleteMark)

        return findMin(t.right);

}
```