1 Tree-versal

a) What is the pre-order traversal of the tree?

b) What is the post-order traversal of the tree?

c) What is the in-order traversal of the tree?

d) What is the level-order traversal of the tree?
2 Runtime Questions

Provide the best case and worst case runtimes in theta notation in terms of \( N \), and a brief justification for the following operations on a binary search tree. Assume \( N \) to be the number of nodes in the tree. Additionally, each node correctly maintains the size of the subtree rooted at it. [Taken from Final Summer 2016]

boolean contains(T o); // Returns true if the object is in the tree

Best: \( \Theta(\ ) \) Justification:
Worst: \( \Theta(\ ) \) Justification:

void insert(T o); // Inserts the given object.

Best: \( \Theta(\ ) \) Justification:
Worst: \( \Theta(\ ) \) Justification:

T getElement(int i); // Returns the ith smallest object in the tree.

Best: \( \Theta(\ ) \) Justification:
Worst: \( \Theta(\ ) \) Justification:
3  Is This a BST?

The following code should check if a given binary tree is a BST. However, for some
trees, it returns the wrong answer. Give an example of a binary tree for which
brokenIsBST fails.

```java
public static boolean brokenIsBST(TreeNode T) {
    if (T == null) {
        return true;
    } else if (T.left != null && T.left.val > T.val) {
        return false;
    } else if (T.right != null && T.right.val < T.val) {
        return false;
    } else {
        return brokenIsBST(T.left) && brokenIsBST(T.right);
    }
}
```

Now, write isBST that fixes the error encountered in part (a).

*Hint:* You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful.

```java
public static boolean isBST(TreeNode T) {
    return isBSTHelper();
}
```

```java
public static boolean isBSTHelper() {
}
```
Pruning Trees

Assume we have some binary search tree, and we want to prune it so that all values in the tree are between \( L \) and \( R \), inclusive. Pruning simply means removing certain items and adjusting the tree so that it is still a BST. Fill out the method below that takes in a BST, as well as \( L \) and \( R \), and returns the pruned tree. Note that the root of the original tree might not be between \( L \) and \( R \), so make sure you return the root of the new pruned tree.

```java
class BST {
    int label;
    BST left; // null if no left child
    BST right; // null if no right child
}

public BST pruneBST(BST root, int L, int R) {
    if (_______________) {
        return ________;
    } else if (____________________) {
        return pruneBST(____________, _____, _____);
    } else if (____________________) {
        return pruneBST(____________, _____, _____);
    }
    ______________ = pruneBST(____________, _____, _____);
    ______________ = pruneBST(____________, _____, _____);
    return _______;  
}
```
5 BTree Motivation

1. Why does a binary search tree have a worst case runtime of $\theta(n)$ for `contains`?

2. Give a sequence of operations, such that if they were inserted in the order they appear, would result in a "poor" binary search tree.

3. Examine this B-tree with order 3. Mark the paths taken when the user calls `contains(40)`.

4. Now call `insert(35)`, and draw the resulting tree.

5. What property of a B-tree rectifies problems of binary search trees, such as the one in 1.1? Why would you not use a B-tree?