Solution to CSE143 Section #15 Problems

1. One possible solution appears below.

```
    public int numNodes() {
        return numNodes(overallRoot);
    }

    private int numNodes(IntTreeNode root) {
        if (root == null) {
            return 0;
        } else {
            return 1 + numNodes(root.left) + numNodes(root.right);
        }
    }
```

2. One possible solution appears below.

```
    public int numLeaves() {
        return numLeaves(overallRoot);
    }

    private int numLeaves(IntTreeNode root) {
        if (root == null) {
            return 0;
        } else if (root.left == null && root.right == null) {
            return 1;
        } else {
            return numLeaves(root.left) + numLeaves(root.right);
        }
    }
```

3. One possible solution appears below.

```
    public boolean equals(IntTree other) {
        return equals(overallRoot, other.overallRoot);
    }

    private boolean equals(IntTreeNode root1, IntTreeNode root2) {
        if (root1 == null || root2 == null) {
            return root1 == null && root2 == null;
        } else {
            return root1.data == root2.data
                && equals(root1.left, root2.left)
                && equals(root1.right, root2.right);
        }
    }
```

4. One possible solution appears below.

```
      public int depthSum() {
          return depthSum(overallRoot, 1);
      }
```

```
    private int depthSum(IntTreeNode root, int n) {
        if (root == null) {
            return 0;
        } else {
            return n * root.data + depthSum(root.left, n + 1)
                   + depthSum(root.right, n + 1);
        }
    }
```

5. One possible solution appears below.

```
    public int height() {
        return height(overallRoot);
    }

    private int height(IntTreeNode root) {
        if (root == null) {
            return 0;
        } else {
            return 1 + Math.max(height(root.left), height(root.right));
        }
    }
```

6. One possible solution appears below.

```
    public boolean isFull() {
        return (overallRoot == null) || isFull(overallRoot);
    }

    private boolean isFull(IntTreeNode root) {
        if (root.left == null && root.right == null) {
            return true;
        } else {
            return root.left != null && root.right != null
                && isFull(root.left) && isFull(root.right);
        }
    }
```

7. One possible solution appears below.

```
    public boolean contains(int value) {
        return contains(overallRoot, value);
    }

    private boolean contains(IntTreeNode root, int value) {
        if (root == null) {
            return false;
        } else if (root.data == value) {
            return true;
        } else {
            return contains(root.left, value) || contains(root.right, value);
        }
    }
```

8. One possible solution appears below.

```
    public void printLevel(int target) {
        if (target < 1) {
            throw new IllegalArgumentException();
        }
        printLevel(overallRoot, target, 1);
    }

    private void printLevel(IntTreeNode root, int target, int level) {
        if (root != null) {
            if (level == target) {
                System.out.println(root.data);
            } else {
                printLevel(root.left, target, level + 1);
                printLevel(root.right, target, level + 1);
            }
        }
    }
```

9. One possible solution appears below.

```
    public boolean hasPathSum(int sum) {
        return hasPathSum(overallRoot, sum);
    }

    private boolean hasPathSum(IntTreeNode root, int sum) {
        if (root == null) {
            return false;
        } else if (root.data == sum) {
            return true;
        } else {
            int sum2 = sum - root.data;
            return hasPathSum(root.left, sum2) ||
                   hasPathSum(root.right, sum2);
        }
    }
```

10. One possible solution appears below.

```
    public void writeTree() {
        writeTree(overallRoot);
    }

    private void writeTree(IntTreeNode root) {
        if (root != null) {
            int type = 0;
            if (root.left != null) {
                type++;
            }
            if (root.right != null) {
```

```
            type += 2;
        }
        System.out.println(type + " " + root.data);
        writeTree(root.left);
        writeTree(root.right);
    }
}
```