Solution to CSE143X Section #17 Problems

1. 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);
    }
}
```

2. One possible solution appears below.

```
public void tighten() {
    overallRoot = tighten(overallRoot);
}

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

3. One possible solution appears below.

```
public void readTree(Scanner input) {
    overallRoot = readTreeHelper(input);
}

private IntTreeNode readTreeHelper(Scanner input) {
    int type = input.nextInt();
    int data = input.nextInt();
    IntTreeNode root = new IntTreeNode(data);
    if (type == 1 || type == 3) {
        root.left = readTreeHelper(input);
    }
    if (type == 2 || type == 3) {
        root.right = readTreeHelper(input);
    }
    return root;
}
```

4. One possible solution appears below.

```
public void limitPathSum(int max) {
    overallRoot = limitPathSum(overallRoot, max, 0);
}

private IntTreeNode limitPathSum(IntTreeNode root, int max,
                                 int current) {
    if (root != null) {
        current += root.data;
        if (current > max) {
            root = null;
        } else {
            root.left = limitPathSum(root.left, max, current);
            root.right = limitPathSum(root.right, max, current);
        }
    }
    return root;
}
```

5. One possible solution appears below.

```
public void construct(int n) {
    if (n < 0) {
        throw new IllegalArgumentException();
    }
    overallRoot = construct(0, n - 1);
}

private IntTreeNode construct(int low, int high) {
    if (low > high) {
        return null;
    } else {
        int mid = (low + high) / 2;
        return new IntTreeNode(mid, construct(low, mid - 1),
                               construct(mid + 1, high));
    }
}
```

6. One possible solution appears below.

```
public void removeLeaves() {
    overallRoot = removeLeaves(overallRoot);
}

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

7. One possible solution appears below.

```
public void construct(int[] data) {
    overallRoot = construct(data, 0, data.length - 1);
}

private IntTreeNode construct(int[] data, int start, int stop) {
    if (start > stop) {
        return null;
    } else {
        int mid = (start + stop + 1) / 2;
        return new IntTreeNode(data[mid],
                               construct(data, start, mid - 1),
                               construct(data, mid + 1, stop));
    }
}
```

8. One possible solution appears below.

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

private IntTreeNode complete(IntTreeNode root, int target, int level) {
    if (level <= target) {
        if (root == null) {
            root = new IntTreeNode(-1);
        }
        root.left = complete(root.left, target, level + 1);
        root.right = complete(root.right, target, level + 1);
    }
    return root;
}
```

9. One possible solution appears below.

```
public IntTree combineWith(IntTree other) {
    IntTree result = new IntTree();
    result.overallRoot = combine(overallRoot, other.overallRoot);
    return result;
}

private IntTreeNode combine(IntTreeNode root1, IntTreeNode root2) {
    if (root1 == null) {
        if (root2 == null) {
            return null;
        } else { // root2 != null
            return new IntTreeNode(2, combine(null, root2.left),
                                   combine(null, root2.right));
        }
    } else { // root1 != null
        if (root2 == null) {
            return new IntTreeNode(1, combine(root1.left, null),
                                   combine(root1.right, null));
        } else {
            return new IntTreeNode(3, combine(root1.left, root2.left),
                                   combine(root1.right, root2.right));
        }
    }
}
```