Methods in BinarySearchTree class:

```
int countSiblingDiffGreaterThan(int threshold) {
    return root.countSiblingDiffGreaterThan(threshold);
}

ArrayList<Integer> buildSearchPathList(int item) {
    BooleanContainer bc = new BooleanContainer();
    ArrayList<Integer> list = new ArrayList<>();
    root.buildSearchPathList(item, list, bc);
    if (!bc.found) {
        list.clear();
    }
    return list;
}

class BooleanContainer {
    public boolean found = false;
}


void sproutToDepth(int depth) {
    root = root.sproutToDepth(depth, 0);
}
```

Methods in BinaryNode class:

```
public int countSiblingDiffGreaterThan(int threshold) {
    if (this == NULL_NODE) {
        return 0;
    }
    int count = 0;
    if (this.left != NULL_NODE && right != NULL_NODE
            && right.data - left.data > threshold) {
        count++;
    }
    return count
            + left.countSiblingDiffGreaterThan(threshold)
            + right.countSiblingDiffGreaterThan(threshold);
}


public void buildSearchPathList(int item, ArrayList<Integer> list, BooleanContainer bc) {
    if (this == NULL_NODE) {
        bc.found = false;
        // NOTE: Clearing the list here means a BooleanContainer isn't really needed
        // on this problem; I just left it in so you'd have an example of the pattern.
        list.clear();
        return;
    }
    list.add(data);
    if (item < data) {
        left.buildSearchPathList(item, list, bc);
    } else if (item > data) {
        right.buildSearchPathList(item, list, bc);
    } else {
        bc.found = true;
    }
}


public BinaryNode sproutToDepth(int depth, int value) {
    if (this == NULL_NODE) {
        if (depth >= 0) {
            return new BinaryNode(value);
        }
        return NULL_NODE;
    }
    left = left.sproutToDepth(depth - 1, this.data);
    right = right.sproutToDepth(depth - 1, this.data);
    return this;
}
```