# CS 11 Data Structures and Algorithms

## Assignment 13: Binary Search Trees 1

**Skip to Main Content**

### Assignment 13.1 [0 points, optional]

No documentation is required on this assignment.

This assignment will not be graded for style issues. If your code works correctly, you'll get 15 points. Style will be graded next week.

Start with the binaryTree class provided in lesson 23 and make the following changes. Don't make any changes other than the additions listed here. Do not add the big-3 yet. We'll work on that next week. (Adding private helper functions is also ok.)

1. **mSize**: Add a data member to store the size of the tree. Call it "mSize" to distinguish it from the "size()" function. You'll need to update this new data member in all of the appropriate places in the class implementation. Change the size() function so that it returns this data member instead of calculating the size.

2. **numPrimes()**: Add a "numPrimes()" function to the class that returns the number of nodes in the tree that contain prime integers. Good decomposition dictates that you will want a helper function that determines whether its parameter is prime. Don't worry about making this efficient. Just test all of the numbers less than the parameter and if any of them divide evenly into the parameter, then it's not prime.

3. **toLL()**: Add a "toLL()" function to the class that converts the calling binary tree object into an LL object. The items in the LL object should be in increasing order. The created LL object should be returned. Here's a sample client to illustrate how this function might be used:

```
#include <iostream>
#include "LL.h"
#include "binarytree.h"
using namespace std;

int main() {
    binarytree t;
    for (int i = 0; i < 20; i++) {
        t.insert(rand() % 50);
    }

    cout << "The binary tree: ";
    t.print();
    cout << endl;

    LL<int> l;
    l = t.toLL();

    cout << "The linked list: ";

    for (LL<int>::iterator i = l.begin(); i != l.end(); i++) {
        cout << *i << " ";
    }
    cout << endl;

    cout << "The original binary tree still intact: ";
    t.print();

    cout << endl;
}
```

The hard part of the toLL_aux() function is making sure that not only do you call toLL_aux() recursively on the left and right subtrees, but you link up the results. You can't just call toLL_aux(left) and then call push_front() and then call toLL_aux(right). You have to make sure you link them all up. To put this another way, you need to think of the toLL_aux() function not as a function that converts a binarytree to an LL, but as a function that appends a binarytree to an already existing LL. To do this, I passed an LL iterator as one of my arguments, and toLL_aux() places the binarytree object argument in the LL object after this iterator. I used push_front() only when the LL iterator is nullptr (so the LL object is being created from scratch, not added to an already existing LL object). In other cases, I used insert_after(). Also, my toLL_aux() is void.

I'm sure there are other approaches. This is just how I approached things.

## Submit Your Work

Use the Assignment Submission link to submit your binarytree.h and binarytree.cpp files. No client file or output is required. When you submit your assignment there will be a text field in which you can add a note to me (called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the class works as required.