1. [10 points] Suppose that a binary search tree was built by inserting the following values, in the order they are listed:

```plaintext
45  55  19  23  21  12  50  62  73  10  51
```

For each value below, list the data values that would be examined in a search of the binary search tree for that value.

a) 73

b) 51

**Inserting the given values in the given order yields the following BST:**

```
      45
     /  \
   19    55
  /  \\ /  \\ 
12  23  50  62
 / \   /   /   \
10 21 51 73
```

a) Searching for 73 would examine the data values: 45 55 62 73

b) Searching for 51 would examine the data values: 45 55 50 51
For the following questions, assume the following interfaces for binary node and BST templates:

```cpp
template <typename T> class BinNodeT {
public:
    T Element;
    BinNodeT<T>* Left;
    BinNodeT<T>* Right;

    BinNodeT();
    BinNodeT(T D, BinNodeT<T>* L = NULL, BinNodeT<T>* R = NULL);
    ~BinNodeT();
};
```

```cpp
template <typename T> class BST {
protected:
    BinNodeT<T>* Root;

    unsigned int HeightHelper(BinNodeT<T>* sRoot) const;
    ...
    T* const FindHelper(const T& toFind, BinNodeT<T>* sRoot);
    const T* const FindHelper(const T& toFind, BinNodeT<T>* sRoot) const;

public:
    BST (); // create empty BST
    BST (const BST<T>& Source); // deep copy support
    BST<T>& operator=(const BST<T>& Source);
    unsigned int Height() const; // number of levels in tree
    void Display(ostream& Out); // formatted inorder display
    void Clear(); // restore tree to empty state
    ...
    bool Insert(const T& D); // insert element
    bool Delete(const T& D); // delete element
    T* const Find(const T& D); // return access to D
    const T* const Find(const T& D) const; // return safe access to D
};
```

2. [10 points] Write an implementation for the `HeightHelper()` function.

```cpp
template <typename T>
unsigned int HeightHelper(BinNodeT<T>* sRoot) const {
    if (sRoot == NULL) return 0; // fallen off the tree, no nodes here

    // get the heights of both subtrees (possibly empty):
    unsigned int leftHeight = HeightHelper(sRoot->Left);
    unsigned int rightHeight = HeightHelper(sRoot->Right);

    // return the maximum subtree height plus 1 for the current level:
    if (leftHeight >= rightHeight)
        return (1 + leftHeight);
    else
        return (1 + rightHeight);
}
```
3. [10 points] Write an implementation for the recursive helper function for the new BST template member function, using the interface shown below, and satisfying the description of the function's behavior given in the header comment:

// This public function uses a recursive helper function to perform an inorder traversal of the BST and return a vector containing all the data values that are in leaf nodes of the BST.
//
// template <typename T> vector<T> BST<T>::logLeaves() const {
// vector to hold the data values
vector<T> loggedData;
logHelper(Root, loggedData); // call helper to do the work
return loggedData; // return the logged values
}

template <typename T>
void BST<T>::logHelper(BinNodeT<T>* sRoot, vector<T>& leafData) const {
if (sRoot == NULL) return; // no subtree, no data to log
logHelper(sRoot->Left, leafData); // process left subtree
if (sRoot->Left == NULL && sRoot->Right == NULL) { // at a leaf?
    leafData.push_back(sRoot->Element);
    return;
}
logHelper(sRoot->Right, leafData); // process right subtree
}
4. [10 points] Give a formal, mathematical proof of the following theorem (induction is logically required):

If T is a nonempty binary tree with n nodes, then T has n - 1 edges.

proof: Let S be the set of all positive integers n such that if T is a nonempty binary tree with n nodes, then T has n – 1 edges.

If T is a binary tree with one node, then that node must be a leaf, and T has no edges. So, 1 is in S.

Assume: there is a positive integer n in S. That is, for some n, if T is any binary tree with n nodes, then T has n – 1 edges.

Let T be an arbitrary binary tree with n + 1 nodes. Now, T has at least 2 nodes and at least one of its nodes must be a leaf. Select a leaf (doesn't matter which one), and remove that leaf and the corresponding edge from T. The resulting binary tree has n nodes, and so by the assumption above, it must have n – 1 edges. But T had one more edge than this "pruned" tree, so T must have had n edges, which is exactly what we needed to show. Therefore, n + 1 is in S.

So, by the Principle of Induction, S includes all positive integers.