1. [25 points] Design an algorithm to count and return the number of nodes in a binary tree that have two children. Express your solution as a pair of Java functions (not BST member functions), which would be implemented in the same package as the BST generic specified in Minor Project 2:

```java
public int numFullNodes( BST<T> Tree ) {
    return nFNHelper(Tree.root);
}

private int nFNHelper(BinaryNode sRoot) {
    if ( sRoot == null ) return 0;
    int countThis = sRoot.left != null && sRoot.right != null ? 1 : 0;
    return countThis + nFNHelper(sRoot.left) + nFNHelper(sRoot.right);
}
```

(Of course, the function shown above should have a recursive helper function.)
2. [25 points] Design an algorithm to count and return the greatest lower bound (GLB) of a data object $X$. Your solution will assume that the following public method has been added to the interface for the BST given in Minor Project 2:

```java
// Pre: X is a valid object of type T
// Returns: reference to the unique object Y in the BST such that
//          Y = max { Z in tree | X.compareTo(Z) >= 0 }
//          or NULL if no such element exists in the BST
//
public T GLB(T X) {
    return GLBHelper(X, root);
}
```

Complete the implementation of the following private helper function, which would also be added to the given BST interface:

```java
private T GLBHelper(T X, BinaryNode sroot) {
    if (sroot == null) return null;

    int compare = X.compareTo(sroot.element);

    // Three cases:
    // X == sroot.element, that's the GLB
    // X < sroot.element, GLB can only be in left subtree
    // X > sroot.element, GLB can only be here or in right subtree
    if (compare == 0) {
        return sroot.element;
    }
    if (compare < 0) {
        return GLBHelper(X, sroot.left);
    }
    if (compare > 0) {
        T candidateFromRight = GLBHelper(X, sroot.right);
        if (candidateFromRight == null) {
            return sroot.element;
        } else {
            return candidateFromRight;
        }
    }
}
```

The "..." indicates you may use additional parameters if you find them useful or necessary. Your implementation should operate as efficiently as possible; that is, it should not examine any branch of the BST unless that branch could contain relevant data.
3. [25 points] Use Induction to prove the following fact: for every integer, \( h \geq 0 \), a full binary tree with height \( h \) can have at most \( 2^{h+1} - 1 \) nodes. (You may not use any of the tree theorems from the notes.)

**proof:** Let \( h = 0 \), then we have an empty tree, with 0 nodes, and \( 2^{h+1} - 1 = 2^1 - 1 = 1 \), so the bound holds for \( h = 0 \).

Now assume that for some \( k \geq 0 \), whenever we have a full binary tree with height less than or equal to \( k \), the tree has no more than \( 2^k - 1 \) nodes.

Suppose we have a full binary tree with \( k+1 \) nodes. Then the tree consists of a root node and two subtrees, which are also full binary trees, and note that each subtree has height less than or equal to \( k \). Thus, each subtree has no more than \( 2^k - 1 \) nodes. Therefore, the number of nodes in the entire tree can be no more than \( (2^{k+1} - 1) + (2^{k+1} - 1) + 1 = 2 \cdot 2^{k+1} - 1 = 2^{k+1} - 1 \).

Therefore, the result holds for all full binary trees.

(Note: the proof never uses the fact that the binary tree is full.)

4. [25 points] Use the result proved in question 3 to prove that: for every integer, \( N \geq 0 \), a full binary tree with \( N \) nodes must have at least \( \lceil \log(N+1) \rceil - 1 \) levels.

From problem 3, we know that we must have \( N \leq 2^{h+1} - 1 \).

Rearranging terms and taking the base-2 logarithm of both sides, we get that:

\[ h + 1 \geq \log(N + 1) \]

Since the left side is an integer value, this implies that:

\[ h + 1 \geq \lceil \log(N + 1) \rceil \]

And so,

\[ h \geq \lceil \log(N + 1) \rceil - 1 \]