You will submit your solution to this assignment to the Curator System (as `HW01`). Your solution must be either a plain text file (e.g., NotePad++) or a typed MS Word document; submissions in other formats will not be graded.

Partial credit will only be given if you show relevant work.

For questions 1 through 4, refer to the BST generic interface provided for Minor Project 2. Strive for the most efficient solution you can devise. You may not use any Java collections, including ones you implement yourself, to store any additional information (like copying the data elements from the BST into an array).

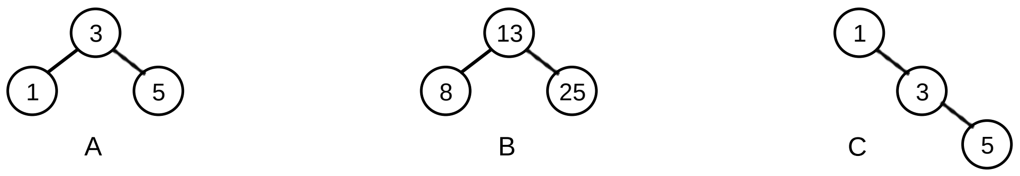
1. **[20 points]** Design an algorithm to determine whether a given binary tree contains any *single-child* nodes; that is, nodes that have exactly one child. Express your solution using Java code as if it were to be implemented as a public member function (with private helper) belonging to the BST generic specified in Minor Project 2; your answer (which will include both the public and private functions) must conform to the public interface shown below:

```java
// Pre:      none
// Post:     the BST is unchanged
// Returns:  true iff the BST has at least one node with a single child
//
public boolean hasSingleChild( ) {

   // up to you. . .
}
```

2. **[20 points]** This is a modification of question 1. Design an algorithm to count the number of *single-child* nodes in a binary search tree; that is, nodes that have exactly one child. Express your solution using Java code as if it were to be implemented as a public member function (with private helper) belonging to the BST generic specified in Minor Project 2; your answer (which will include both the public and private functions) must conform to the public interface shown below:

```java
// Pre:      none
// Post:     the BST is unchanged
// Returns:  the number of nodes in the tree that have a single child
//
public int numSingleChildNodes( ) {

   // up to you. . .
}
```

3. **[20 points]** We say that two binary trees are *congruent* if and only if the two trees have exactly the same structure (arrangement of their nodes), without regard for the data values stored in the trees. For example, trees A and B below are congruent, but trees A and C are not:



Design an algorithm to determine whether two BSTs are *congruent*. Express your solution using Java code as if it were to be implemented as a public member function (with private helper) of some class belonging to same package as the BST implementation; your answer (which will include both the public and private functions) must conform to the public interface shown below:

```
// Pre:      none
// Post:     the BST is unchanged
// Returns:  true iff the two trees are congruent
//
public boolean areCongruent( BST<Integer> A, BST<Integer> B) {

   // up to you. . .
}
```

4. Assume that an implementation of the BST generic includes three functions as described below:

```
// Writes the data elements in the tree (using their toString() method)
// to the given output stream, using a preorder traversal.
//
// Pre:      out is opened on a file
// Post:     the BST is unchanged
//
public void writePreorder( FileWriter out ) { . . . }

// Writes the data elements in the tree (using their toString() method)
// to the given output stream, using a postorder traversal.
//
// Pre:      out is opened on a file
// Post:     the BST is unchanged
//
public void writePostorder( FileWriter out ) { . . . }

// Writes the data elements in the tree (using their toString() method)
// to the given output stream, using an inorder traversal.
//
// Pre:      out is opened on a file
// Post:     the BST is unchanged
//
public void writeInorder( FileWriter out ) { . . . }
```

a) **[10 points]** If `writePreorder()` and `writeInorder()` yield exactly the same output for a given nonempty BST, what must be true about that BST? Explain.

b) **[10 points]** If `writePreorder()` and `writePostorder()` yield exactly the same output for a given nonempty BST, what must be true about that BST? Explain.

5. **[20 points]** Use Induction to prove the following theorem about binary trees:

For every nonempty binary tree, the number of nodes that have two children is one less than the number of leaf nodes.

Note: this is similar to the first part of the Full Binary Tree theorem, but different, since this does not assume anything special about the binary tree, other than that it is nonempty.