Assessed Exercise 2

**Before attempting the tasks, please check the “hints” you can find below each task.**

**Task A: Binary Tree Traversal (Display)**

Implement a Console Application for C#. Add the classes **Node** and **BinTree**. Add to the BinTree class InsertItem and the following methods to traverse the tree: InOrder, PostOrder, PreOrder.

Call (in the Main() method) the InOrder, PostOrder and PreOrder methods on the tree below (constructed using InsertItem) and display the result of the methods.

*Hints: We have completed the InOrder and InsertItem method in the Lecture and Lab. The methods PostOrder and PreOrder need to be completed following the pseudo-code that can be found on the associated slides. Note the similarities in the pseudo-code between the three visit methods. On the lab slides there is also an example on how to construct a tree and how to call a visit method in the Main() method and display the result of the visit.*

**Task B : Binary Tree Contains Method**

Add a **Contains** Method to your Binary Tree. As you are now adding items via your InsertItem method the tree is an ordered Binary Tree (a Binary Search Tree):

public Boolean Contains(int item)

The method should allow an item to be searched for, returning true if it’s present in the tree and false otherwise. Add at least two tests to your application (Main) to show this working, one where the item is not present and one where it is present (at a leaf of your non-trivial tree). In Main, you can use if statements to display a suitable message to the Console.

*Hints: We need a public and private version of the method. The private version will have some similarities to insertItem as that method also searches for the correct empty place to add a new item. Here we can search either left or right subtrees, depending on whether the item is smaller or larger than the data in the current node of the tree.*

**Task C: Algorithms on Trees**

Implement a binary tree that stores *strings* as data (take a copy and modify, the classes Node and BinTree implemented in the lab).

Implement in the class BinTree (and systematically test in Main()):

* A method **Longest()** that returns the longest string (containing the most characters) present in the tree.
* A method called **Ancestor()** that takes as arguments two strings and (if they are both present in the tree) returns the string associated with the first common ancestor node in the tree (see example below). This method is extensively used in many applications from bioinformatics (e.g. phylogenetic trees) to artificial intelligence.

*Hints: For the method Longest, your solution should use the recursive structure of the tree (use the same idea as we did for the method to count the number of nodes) and search the entire tree. String.Length is a property (see* [*https://learn.microsoft.com/en-us/dotnet/api/system.string.length?view=net-7.0*](https://learn.microsoft.com/en-us/dotnet/api/system.string.length?view=net-7.0)*). For the Ancestor method, search the ordered Binary tree making use of the Contains method you wrote in part B. A good solution should be independent of the order of the arguments (the two strings being search for).*

Example:

The method Ancestor that takes as arguments the string “idea” and “waif” should return “join” which is the string associated with the first common ancestor.

The node with string “hello” is also a common ancestor but not the first (i.e., “closest”) one.