CAB302 – Assignment 2 Empirical Algorithm Analysis

1. Understand the experiment’s purpose

The purpose of this empirical algorithm test is to test the implementation of the ToArray function present in the membersCollection abstract data type (ADT) to verify the accuracy of a theoretical analysis of the algorithm’s efficiency.

1. The efficiency metric to be measured in this empirical study

The chosen metric for the empirical analysis of the ToArray function is the execution time of the program against the input size e.i the number of nodes.

1. The basic operation of the ToArray function

The input to the ToArray function is a binary search tree (BST), and the basic operation is the insertion of each node into the array. As each node must be visited exactly once, the number of operations grows linearly with the number of nodes. Thus, the problem size is defined by the number of nodes in the BST, and the function has a theoretical time complexity of

1. Prepare a program implementing the algorithm (or algorithms) for  
   the experimentation

The algorithm was implemented in C#, and the count variable present in the ADT is used to keep track of the amount of nodes in the BST. *Appendix A* contains the implementation of the ToArray function.

1. Generate a sample of inputs

A C# program was written that random generated the number of nodes in the BST between 0 and 100

1. Run the algorithm(s) on the sample’s inputs and record the data  
   observed
2. Analyse the data obtained

***Appendix A***

**ALGORITHM** *IsEmpty()*

// Check if this member collection a binary search tree is empty

//

**If** *root = null*

**return *true***

**else**

**return *false***

**ALGORITHM** *InOrderTraverse(root, members, index)*

//Traverses a binary search tree and fills the array 'members' with sorted member elements.

// Input: root, members, index

// root: The first node in the binary search tree (BTreeNode).

// members: The array in which nodes will be placed, sorted alphabetically by their names.

// index: A variable that tracks the current position in the members array (passed by reference).

//Output: An array that contains sorted list of IMembers by their names.

**If** *root null* **do**

*InOrderTraverse(root.lchild, members, index)*

*members[index] ← root.member*

*index ← index + 1*

*InOrderTraverse(root.rchild, members, index)*

**ALGORITHM** *ToArray()*

// Returns an array that contains all the members in this member collection.

// The members in the array are sorted in alphabetical order by full name.

**If** *IsEmpty()*

***return*** *null*

MemberArray 🡨 IMember [*0…count - 1]*

*Index 🡨 0*

*InOrderTraverse(root, MemberArray, index)*

**return** memberArray[]