# [CS302-Data Structures] Homework 4: Binary Search Trees

*Instructor:* Kostas Alexis
*Teaching Assistants:* Heyang Qin, Hemanta Sapkota, Huan Nguyen
Fall 2020 Semester

**Section 1.** Exercise 1: Binary Search Tree and Inorder, Preorder, Postorder Traversal

## Section 1. Exercise 1: Binary Search Tree and Inorder, Preorder, Postorder Traversal

Randomly generate 100 unique values in the range of [1-200] and insert them into a binary search tree (BST). Print height and inorder, preorder and postoder output of the BST tree. To implement your tree make sure you follow the interface code provided below. Deliver source code and a test file that shows the result of printing height (10%) and inorder (30%), preorder (30%), and postorder (30%) traversal.

```
/** Link-based implementation of the ADT binary search tree.
 @file BinarySearchTree.h */

#ifndef BINARY_SEARCH_TREE_
#define BINARY_SEARCH_TREE_

#include "BinaryTreeInterface.h"
#include "BinaryNode.h"
#include "BinaryNodeTree.h"
#include "NotFoundException.h"
#include "PrecondViolatedExcept.h"
#include <memory>

template<class ItemType>
class BinarySearchTree : public BinaryNodeTree<ItemType>
{
private:
   std::shared_ptr<BinaryNode<ItemType>> rootPtr;

protected:
   //------------------------------------------------------------
   //    Protected Utility Methods Section:
   //    Recursive helper methods for the public methods.
   //------------------------------------------------------------
   // Places a given new node at its proper position in this binary
```

```
    // search tree.
    auto placeNode(std::shared_ptr<BinaryNode<ItemType>> subTreePtr,
                   std::shared_ptr<BinaryNode<ItemType>> newNode);

    // Removes the given target value from the tree while maintaining a
    // binary search tree.
    auto removeValue(std::shared_ptr<BinaryNode<ItemType>> subTreePtr,
                     const ItemType target,
                     bool& isSuccessful) override;

    // Removes a given node from a tree while maintaining a binary search tree.
    auto removeNode(std::shared_ptr<BinaryNode<ItemType>> nodePtr);

    // Removes the leftmost node in the left subtree of the node
    // pointed to by nodePtr.
    // Sets inorderSuccessor to the value in this node.
    // Returns a pointer to the revised subtree.
    auto removeLeftmostNode(std::shared_ptr<BinaryNode<ItemType>>subTreePtr,
                            ItemType& inorderSuccessor);

    // Returns a pointer to the node containing the given value,
    // or nullptr if not found.
    auto findNode(std::shared_ptr<BinaryNode<ItemType>> treePtr,
                  const ItemType& target) const;

public:
    //------------------------------------------------------------
    // Constructor and Destructor Section.
    //------------------------------------------------------------
    BinarySearchTree();
    BinarySearchTree(const ItemType& rootItem);
    BinarySearchTree(const BinarySearchTree<ItemType>& tree);
    virtual ~BinarySearchTree();

    //------------------------------------------------------------
    // Public Methods Section.
    //------------------------------------------------------------
    bool isEmpty() const;
    int getHeight() const;
    int getNumberOfNodes() const;

    ItemType getRootData() const throw(PrecondViolatedExcept);
    void setRootData(const ItemType& newData);

    bool add(const ItemType& newEntry);
```

```
   bool remove(const ItemType& target);
   void clear();

   ItemType getEntry(const ItemType& anEntry) const throw(NotFoundException);
   bool contains(const ItemType& anEntry) const;

   //------------------------------------------------------------
   // Public Traversals Section.
   //------------------------------------------------------------
   void preorderTraverse(void visit(ItemType&)) const;
   void inorderTraverse(void visit(ItemType&)) const;
   void postorderTraverse(void visit(ItemType&)) const;

   //------------------------------------------------------------
   // Overloaded Operator Section.
   //------------------------------------------------------------
   BinarySearchTree<ItemType>&
   operator=(const BinarySearchTree<ItemType>& rightHandSide);
}; // end BinarySearchTree

#include "BinarySearchTree.cpp"
#endif
```