**Brandon Hobbs**

**Jan. 30, 2022**

**CS-300**

**Module 5 BinarySearchTree**

The code is broken down into the following functions/structs/classes:

* Class BinarySearchTree
  + Class definition of the BinarySearchTree objects that hold
    - Private members: size, root, and inOrder()
    - Public methods: Constructor, destructor, InOrder(), Insert(), Remove(); Search(); Size()
* BinarySearchTree()
  + Default constructor that sets the root to a null pointer
* BinarySearchTree::Insert(Bid)
  + A Bid is passed
  + The function then adds the passed Bid as a new Node in the tree (nodes)
* BinarySearchTree::InOrder()
  + Function loops through the BinarySearchTree starting at the root and outputs, to the console, 4 values from the Bid struct
* BinarySearchTree::Remove(String)
  + Function starts at the root and searches for the String
  + Upon finding the String the Node containing it is freed from memory
  + The tree is updated accordingly
* BinarySearchTree::Search(String)
  + Function starts at the root and searches for the passed String
  + Upon finding the String the Node containing it is returned
* BinarySearchTree::Size()
  + A get method used to access the private member, size
* strToDouble
  + Used to convert the CSV file data into useable value
* Bid
  + Struct containing the data
  + Used with the vector that will be sorted
* Node
  + Struct containing the Node data
  + Consists of a Bid and 2 pointers: left and right
* loadBids
  + Function used to read in the csv data
  + Can read the csv path in from arguments or use a default path
  + Add parts of the data into the Bid structure and then adds that Bid to the BinarySearchTree
* main
  + *main* is the primary driver for the application
  + *main* has a menu to allow a user to enter a bid, load the data, view the data, and delete a node and then exit the application
  + *main* also reports the timing each algorithm takes to perform the sort using the *time.h* library

The code was straight forward, especially since the parser was delivered to me.

Pseudocode:

**Main** Function()

**Read** cmd arguments

**Store** argument as CSV file path

**If** no cmd arguments load default CSV file path

**Loop** while choice is not equal to ‘9’

**Output** menu

**Get** user input; Store in choice

**Validate** user input

**If** choice is not 1-4 or 9 throw an error

**If** choice equals ‘1’

**Start** the clock and **store** in ticks

**Call** loadBids and store CSV data in BinarySearchTree *bst*

**Output** number of records in the CSV file

**Stop** the clock

**Output** the elapsed time needed to read in the CSV file

**If** choice equals ‘2’

**Call** inOrder()

**If** choice equals ‘3’

**Start** the clock and store in ticks

**Call** Search() passing a *bidKey* to search for

**Stop** the clock

**Output** the elapsed time needed to find the *bidKey*

**If** choice equals ‘4’

**Call** Remove() passing *bidKey*

**If** Choice equals ‘9’

**Exit** the application

**Output** ‘Good bye’

**End**

**BinarySearchTree::Insert(*Bid*)**

**Create** a new Node pointer called *currentNode* and set to the *root*

**Check** if the *root* is Null

**If** the root is Null **Create** a new Node containing *Bid*

**If** *root* is not Null

**Loop** until *currentNode* is Null

**If** bidId within Bid is less than *currentNode’s* bidId

**If** the *currentNode’s* left pointer is Null

**Set** currentNode left pointer to a new Node contain Bid

**Set** currentNode to Null to break

**Else** set currentNode to Node at the left pointer

**Else**

**If** the *currentNode’s* right pointer is Null

**Set** *currentNode* right pointer to a new Node contain Bid

**Set** currentNode to Null to break

**Else** set currentNode to Node at the right pointer

**Increment** size

**End**

**BinarySearchTree::InOrder(Node)**

**Check** if Node is null and if so return

**Call** via recursion Node’s left pointer which will find the left most Node

**Output** to console: *bidId, title, amount, fund*

**Call** via recursion Node’s right pointer which will find the right most Node

**End**

**BinarySearchTree::Search(*String*)**

**Create** a new *Node* pointer called *currentNode*

**Set** *currentNode* to the *root*

**Loop** until *curentNode* is NULL (end of the tree)

**If** the Node at currentNode contains a *bidId* equal than to *String*

**Return** *currentNode’s* Bid

**If** the Node at currentNode contains a *bidId* less than to *String*

**Set** *currentNode* equal to the left Node

**If** the Node at currentNode contains a *bidId* greater than to *String*

**Set** *currentNode* equal to the right Node

**Return** an empty Bid

**End**

**BinarySearchTree::Remove(String)**

**Create** a new *Node* pointer called *curr* and set to *root*

**Create** a new *Node* pointer called *par* and set to Null

**Loop** until *curr* is Null

**If** *curr* containing a BidId equal to String

**Check** if there are leaves

**If** there are leaves assign them to *Par*

**Create** a new Node called *suc*

**Set** *suc* to *curr’s* right pointer

**Loop** until *suc’s* left pointer is null

Set *suc* to its left pointer

**Create** a new Node called *successorData*

**Set** *successorData* to a node containing *suc*

**Call** via recursion Remove() passing *suc’s* bidId

**Set** *curr’s* bid equal to *successorData’s* bid

**If** *curr’s* bidId is less than String

**Set** *par* equal to *curr*

**Set** *curr* equal to *curr’s* right pointer

**Else**

**Set** *par* equal to *curr*

**Set** *curr* equal to *curr’s* left pointer

**End**