**Danny Forte**

**CS-300 Analysis and Design**

**April 05, 2025**

**BST: Code Reflection**

This assignment continues to build upon our previous works and once again the note in the assignment for us to look for pieces of code that we can reuse with adjustments if needed was helpful. This saves lots of time especially when we can reuse the code for printing out our results and only need to make small adjustment to it. We once again load a CSV file and ask for it to display all bids, we asked for it to find a bid and then to be able to delete a bid. The main part of this assignment was our use of Binary Search Trees. A binary search tree (BST) is a data structure that can organize its data in a sorted manner. This allows for quick and efficient searching, insertion and deletion of items.

The part that was challenging for me with the BST structure was understanding the breakdown of its sub trees. I am fairly confident in my understanding of the concept of parent and child nodes, and the levels of the tree. The issue that I ran into as I was working through the zybooks activities and the coding assignment was understanding the order of traversal. This idea at first was very confusing, but as I spent more time reading about it and practicing it, the concept became clearer.

**PsuedoCode**

While choice IS NOT equal to 9

Main

Display user menu:

1. Load Bids

2. Display All Bids

3. Find Bid

4. Remove Bid

5. Insert Bid

9. Exit

Ask for User input and store choice for menu selection

If Choice = 1. Load Bids

Start Clock and store ticks

Call loadBids to open and store CSV data

Output number of records found in CSV file

Stop the clock

Display time needed to read CSV file

Break

If Choice = 2. Display All Bids

Call bst inOrder

Display Bids

Break

If Choice = 3. Find Bid

Start Clock and store ticks

Call search()

Search will complete

Search(bidId)

Set current node as root

While current node is not null

If bidId matches current nodes bid

Return the bid

Else if bidId is smaller

Traverse left subtree

Else

Traverse right subtree

Return empty bid (not found)

Stop the clock

Display time needed to read CSV file

Break

If choice = 4. Remove bid

Call Remove() method with passed bidId;

Remove() method will complete

removeNode(node, bidId)

If node is null

Return null

If bidId is smaller than current nodes bid

Recursively removeNode in left subtree

Else if bidId is larger than current nodes bid

Recursively removeNode in right subtree

Else (node to be removed found)

If node has no children

Delete node, return null

Else if node has one child

Replace node with child, delete node

Else node has two children

Find min bid in right subtree

Replace nodes bid with min bid

Recursively remove min bid in right subtree

Return updated node

Recursive Method: inOrder(node)

If node is not null

Recursively call inOrder on left child

Print nodes bid detail

Recursively call inOrder on right child

Recursive Method: postOrder(node)

If node is not null

Recursively call postOrder on left child

Recursively call postOrder on right child

Print nodes bid detail

Recursive Method: preOrder(node)

If node is not null

Print nodes bid detail

Recursively call preOrder on left child

Recursively call preOrder on right child

Method: addNode(node, bid)

If bid is smaller than current nodes bid

If left child is null

Create new Node with bid as left child

Else

Recursively call addNode on left child

Else

If right child is null

Create new Node with bid as right child

Else

Recursively call addNode on right child