**Binary Search Tree Reflection**

This code implements a binary search tree (BST) to store and manage auction bid data efficiently. The main goal was to create a structure that allows fast insertion, deletion, and searching of bids using their unique IDs as keys. I used recursive methods for most operations since BSTs naturally lead themselves to recursion—each subtree is itself a BST. The trickiest part was handling all the edge cases in node removal, especially when dealing with nodes that have two children. I overcame these by carefully reading the pseudocode instructions and reading text book to make sure the logic is correct.

**Pseudocode for Insert() :**

// if root equarl to null ptr

// root is equal to new node bid

// else

// add Node root and bid

**Pseudocode for Remove() :**

// remove node root bidID

**Pseudocode for Search() :**

// set current node equal to root

// keep looping downwards until bottom reached or matching bidId found

// if match found, return current bid

// if bid is smaller than current node then traverse left

// else larger so traverse right

Bid bid;

return bid;

**Pseudocode for addNote() :**

// if node is larger then add to left

// if no left node

// this node becomes left

// else recurse down the left node

// else

// if no right node

// this node becomes right

//else

// recurse down the left node

**Pseudocode for inOrder() :**

//if node is not equal to null ptr

//InOrder not left

//output bidID, title, amount, fund

//InOder right

**Pseudocode for postOrder() :**

//if node is not equal to null ptr

//postOrder left

//postOrder right

//output bidID, title, amount, fund

**Pseudocode for preOrder() :**

//if node is not equal to null ptr

//output bidID, title, amount, fund

//postOrder left

//postOrder right

**Pseudocode for removeNode() :**

// if node = nullptr return node

// (otherwise recurse down the left subtree)

// check for match and if so, remove left node using recursive call

// (otherwise recurse down the right subtree)

// check for match and if so, remove right node using recursive call

// (otherwise no children so node is a leaf node)

// if left node = nullptr && right node = nullptr delete node

// (otherwise check one child to the left)

// if left node != nullptr && right node = nullptr delete node

// (otherwise check one child to the right)

// if left node = nullptr && right node != nullptr delete node

// (otherwise more than one child so find the minimum)

// create temp node to right

// while left node is not nullptr keep moving temp left

// make node bid (right) equal to temp bid (left)

// remove right node using recursive call

// return node