**Reflection**

This program is designed to implement a binary search tree (BST) for managing a collection of bids. The functions were created to allow efficient searching, insertion, and deletion of data. The bids are organized by a unique bid ID for quick lookups. The data is loaded from a CSV file, and a main menu was developed to allow the user to select which functions they want to execute.

The tree is designed with a focus on recursion and node manipulation. The recursive functions traverse the tree while maintaining its structure, which is critical for inserting nodes in the correct spot as well as preserving order when removing nodes. The biggest challenge was the remove function, as it presents a myriad of edge cases, including instances where the node to be deleted has two children. These cases need to be carefully considered and handled to avoid creating orphan nodes with no connection back to the root of the tree. Due to the binary structure, the average time complexity for a search is O(log N) since only half of the tree is usually searched, while the worst case is O(N), where every node is visited exactly once.

**Pseudocode**

FUNCTION Insert(Bid)

IF root is null

root = new Node(Bid)

ELSE

call addNode(root, Bid)

FUNCTION Search(bidId)

Traverse the tree from root:

IF match found, return Bid

ELSE move left or right

FUNCTION Remove(bidId)

root = removeNode(root, bidId)

FUNCTION InOrder()

call inOrder(root)

FUNCTION PreOrder()

call preOrder(root)

FUNCTION PostOrder()

call postOrder(root)

FUNCTION addNode(node, Bid)

Recursively find correct spot and insert

FUNCTION removeNode(node, bidId)

Locate node, then:

- Remove leaf

- Replace with child if exists

- Or replace with smallest value in right subtree

FUNCTION inOrder(node)

Traverse left → visit → right

FUNCTION preOrder(node)

Visit → left → right

FUNCTION postOrder(node)

Left → right → visit

DESTRUCTOR

Recursively delete all nodes