**Module 5 Reflection and Pseudocode**

Jermaine Wiggins

Southern New Hampshire University

CS300 Analysis and Design

Prof. Sathish Gopalakrishnan

Feb 9, 2025

**Module 5 Reflection and Pseudocode**

This week’s assignment was another challenge as I have never worked with binary search trees (BST), but I understand the why they are preferred because they provide fast search, insertion and deletion if sorted.

The purpose of this program is to load a collection of bids into a BST, from there the user can display all bids, find a bid, or delete a bid.

The first and second fix-me were simple as I just needed to complete the constructor and destructor methods.

**Default Constructor Pseudocode**

**SET** root to null

**Destructor Pseudocode**

**CREATE** current and last processed node

**SET** current to root and last processed to null

**CREATE** stack

**WHILE** current isn’t null, or stack isn’t empty traverse tree

**IF** current isn’t null

**ADD** to stack current moves left

**ELSE** current null check right side of node

**IF** top of stack right is null or last processed is equal to top of stack right

**SET** last processed to top of stack

**REMOVE** from stack and delete node

**ELSE** move current right

**AFTER** deleting allset root to null

Next, implement the in order, post order and pre order public method, which traverse the list and print in a specific order. Additionally, we also implement an insert and remove method public method. For these we simply call the private methods with the root as the parameter.

**Pre-order pseudocode: Root → Left → Right**

**CALL** pre order method root parameter

**In-order pseudocode: Left → Root → Right**

**CALL** In order method root parameter

**Post-order pseudocode: Left → Right → Root**

**CALL** post order method root parameter

**Insert pseudocode**

**CALL** remove node root and bid id parameter

**Remove pseudocode**

**CALL** add node root and bid id parameter

The search fix-me searches the tree for a matching bid id and returns that bid.

**Search pseudocode**

**CREATE** new node set to root of tree

**WHILE** new node isn’t null traverse tree

**IF** matching id found return bid

**ELSE IF**

bid id less than new node bid id new node moves right

**ELSE**

new node moves right

return empty bid if not found

The add node method inserts a new bid into the tree in the correct position, so the bids added are in order based on bid id.

**Add node Pseudocode**

**IF** root is null

**CREATE** a new node and set to root

**CREATE** current node set to root

**TRAVERSE** tree to find correct position for insertion

**IF** bid id smaller than current node bid id and current left is null

**CREATE** new node and add to current left

**ELSE** current node continues left

**ELSE** bid id larger than current node bid id and current right is null

**CREATE** new node and add to current right

**ELSE** current node continues right

The in order, post order, and pre order each traverse the entire tree in a particular order and print the bid info along the way.

**In-order pseudocode: Left → Root → Right**

**IF** node is null return

**CALL** in order method on node **→** left to recursively visit left side of tree

**PRINT** bid info

**CALL** in order method on node **→** right to recursively visit right side of tree

**Post-order pseudocode: Left → Right → Root**

**IF** node is null return

**CALL** in order method on node **→** left to recursively visit left side of tree

**CALL** in order method on node **→** right to recursively visit right side of tree

**PRINT** bid info

**Pre-order pseudocode: Root → Left → Right**

**IF** node is null return

**PRINT** bid info

**CALL** in order method on node **→** left to recursively visit left side of tree

**CALL** in order method on node **→** right to recursively visit right side of tree

The remove node function removes a node recursively, it searches for the correct bid id and handles removal of the node depending on if the node is a leaf, has one child to the right, one child to the left or two children.

**Remove node pseudocode**

**IF** the current node is null return the node

**IF** the bid id is smaller than the current node's bid id

**RECURSE** to the left part of the tree

**ELSE IF** the bid id is larger than the current node's bid id

**RECURSE** to the right part of the tree

**ELSE** the bid id matches the current node's bid id:

**IF** the node has no children

**REMOVE** the node

**IF** the node has only one child:

**REPLACE** the node with its child

**IF** the node has two children

**FIND** the smallest value in the right part of the tree

**COPY** that value to the current node

**REMOVE** the node that had the smallest value

**RETURN** the node