The purpose of this code is to open and iterate through each line in a CSV file. These lines are then individually separated into their columns and to be assigned to a node and it's attributes. These nodes belong to a "binary search tree" data structure where the nodes location is based on the integer value each line associates with the bid ID. The root of the BST is the middle bid ID where there is approximately an equal amount of other bids on either side of it being either less than or greater than itself. This data structure has the advantage of sorting data in order and allowing for efficient search queries since it just has to compare the search parameter to the current node and determine if the parameter is greater than it or not. It then moves to the next node where is compares again and if it doesn't matches it repeats the same steps until nothing is found or the information matches.

This data structure was confusing to implement at first as it was difficult to keep track of where the current node was or will go when conditions are met. What helped was sketching out an example tree with each possible amount of children a node can have and varying depths until a leaf node is reached. I was able to draw lines between the nodes as kind of a bread crumb trail to track the scope each node has. This method made it much easier to create conditions the current node will encounter and what needs to be considered.

I used a lot of while loops and recursions which gets confusing especially when there is a lot of recursion. The tree sketches also helped with this because I could keep track of what recursion step was the one working and when it was exited. Otherwise I was able to refer back to the hash table and linked lists program files to fill in any gaps and avoid rewriting more code than needed as I already had code that functioned the way I needed it to requiring some minor medications to work with the variables like "root", "left", "right" versus "head", "tail", or "next".

BST::BST() { // default constructor

root = nullptr

}

BST::~BST() { //destructor

while root != nullptr

create pointer current type Node = root

while not a leaf node

if current->left != nullptr

set current = current-> left

continue

if current ->right != nullptr

set current = current->right

delete current

}

void BST::PostOrder() {

pass root through postOrder()

}

void BST::Insert(Bid bid) {

if root = nullptr

set root = pass bid through a new Node

else

pass root and bid through addNode()

}

void BST::Remove(string bidId) {

set root = root and bid passed through removeNode()

}

Bid BST::Search(string bidId) {

create pointer current type Node = root

while current != nullptr

if current bidId = bidId

return current bid

else if current bidId > bidId

set current = current->left

else

current = current->right

create empty bid

return bid

}

void BST::addNode(Node \*node, Bid bid) {

create pointer current type Node = node

while current != nullptr

if current bidId > bid's bidId

if current->left = nullptr

set current->left = bid passed through a new Node()

return

else

current = current->left

else

if current->right = nullptr

set current->right = bid passed through a new Node()

return

else

set current = current->right

}

void BST::inOrder(Node \*node) {

if node != nullptr

pass node->left through inOrder()

print node's bid information

pass node->right through inOrder()

}

void BST::postOrder(Node \*node) {

if node != nullptr

pass node->left through postOrder()

pass node->right through postOrder()

print node's bid information

}

void BST::preOrder(Node \*node) {

if node != nullptr

print node's bid information

pass node->left through preOrder()

pass node->right through preOrder()

}

Node \*BST::removeNode(Node \*node, string bidId) {

if node = nullptr

return node

if bidId < node's bidId

node->left = node->left and bidId passed through removeNode()

return node

else if bidId > node's bidId

node->right =node->right and bidId passed through removeNode()

if node->left = nullptr

create pointer temp type Node = node->right

delete node

return temp

else if node->right = nullptr

create pointer node temp type Node = node->left

delete node

return temp

create pointer tempNode type Node = node->right

while node->left != nullptr

set tempNode = tempNode->left

set node->left = tempNode->bid

set node->right=pass node->right and tempNode's bidId through removeNode()

return node

}