## Insertion:

Code for insertion :

Node \*temp = **new** Item(title, description, price, quantity, maxDiscount);

current->Add(temp);

Pseudo code:

Create new node object of item/category type

Add the new node to the parent node:

Push the new node into the list variable of the parent node

Create a new node class/object, using either category or item derived (children classes). This uses inheritance data structure to create and store different categories and items within a list in the tree for pointer to the children. The “.add()” function adds a node to the current categories list if it is a category.

Complexity analysis: O(1)

This is because we are already within the category or subcategory when the add/insert function is called, and so don’t have to traverse the tree at all just add the new node to the list of children in the parents’ category.

## Deletion:

Code for deletion:

current->GetParent()->Remove(current);

children\_.remove(component);

component->SetParent(**nullptr**);

Pseudo code:

Remove node from the list in parents’ variables

Set parent variable in node to NULL

Delete node to free up memory

Just remove the node from the parents list of children, and then free up memory using “delete” command.

Complexity analysis: O(1)

This is because we are already within the category or subcategory when the add/insert function is called, and so don’t have to traverse the tree at all just remove the node from the list of children in the parents’ category.

## Update:

Code for update function:

current->setTitle(newTitle);

current->setQuantity(newQuantity);

current->setPrice(newPrice);

current->setDescription(newDescription);

current->setMaxDiscount(newMaxDiscount);

Pseudo code:

Receive new variables for update

Override variables in node to new variable data

Read in all values for updated fields and then use predefined functions to update fields for that node. Same process fort category update command but just uses title.

Complexity analysis: O(1)

This is because we are already within the category or subcategory when the update function is called, and so don’t have to traverse the tree at all just update the fields within the node.

## Search algorithm:

**for** (**const** Node \*c : children\_)

{

found = c->Operation(find);

**if** (found == **true**) {

**return** **true**;

}

}

Pseudo code:

Use depth first traversal and recursion:

Recursively traverse the list of children

Compare the title of the node with the searched string

If match then

Print item/category to user

Else

Print no items found

The code basically recursively traverses through the list and searches for items and categories matching the string entered.

Complexity analysis: O(n)

We must traverse the whole tree because we must search for all items and categories that match not only the first one found. Therefore, complexity is O(n) with ‘n’ being the number of nodes in the tree.

## Justification:

I used inheritance with base Node class and two (item and category) derived classes to store data and this is the most efficient method as no duplicates are used when creating the nodes. This saves a lot of space on the computer especially when the tree may have large amounts of data stored within it.