# 4-2 Milestone Three: Enhancement Two: Data Structures and Algorithms

Brandon Goucher

CS 499 Computer Science Capstone

Prof Brooke

10/18/2024

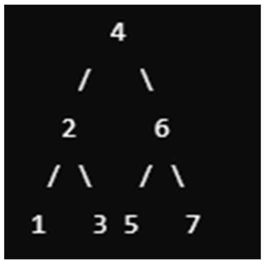
Introduction:

I chose my CS 360 Inventory Mobile Application for the data structures and algorithms. This was another project that, upon completion in that class, I knew that there was more that I wanted to do with this app, and fortunately, my next part fits exactly for part of this final. The whole idea of this app was to implement some user authentication. Then, once a user has signed in, they will be able to add, remove, and update items that any user wants to be able to store inside the database. Although I did have some complications with this project, it was certainly the most fun project I had gotten to work on, so continuing it was super exciting.

My goal for this project was to rearrange the main screen to incorporate a search bar so that a user could search for certain items within their own dataset, and then, if the item existed, it would take them to a new page with the item's specific details. This search bar needs some sort of algorithm to improve the overall speed of the query and future-proofing for if/when there are hundreds, if not thousands, of items inside of the database.

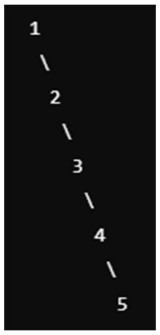
Creating a Binary Search Tree algorithm allowed this project to have a time complexity of O(log n) for the best-case scenario or O(n) for the worst-case scenario. For the time complexity to be the best-case scenario, it would only happen if the item numbers inside the database flowed in without being sequential, like 1, 2, 3, 4, 5, etc. The BST works in this case because it takes the number at the top of the database and works down from there no matter who puts that specific item number in. Using this logic, we can assume that it is almost impossible for every user to make it a perfect sequence; therefore, we are more than likely to see the O(log n) time instead of O(n) time.

To explain with images, this would be the BST if it happened to work out in random order:



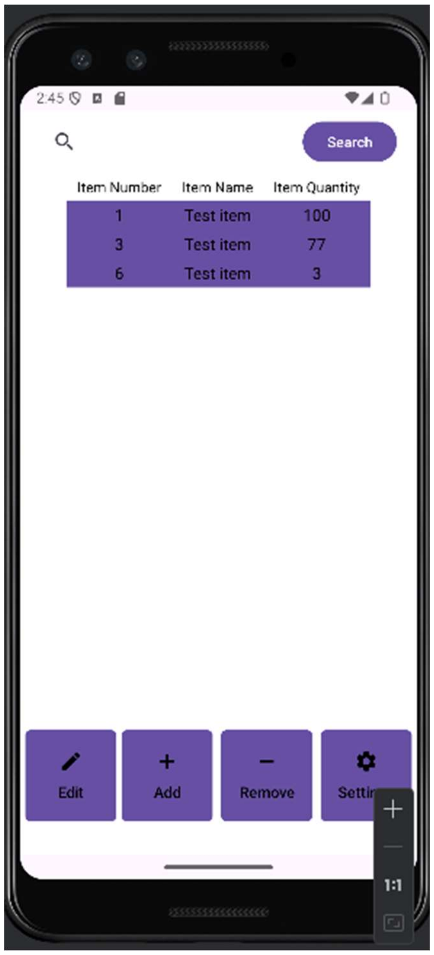
```
      4
    /   \
   2     6
  / \   / \
 1   3 5   7
```

With this, the time is cut down quite a bit. Let's say that the user is looking for the number 5. This works by checking to see if the starting number “4” is less than 5 or larger than 5. In this case, 4 is smaller, so it will go to the right, then it checks the same thing with 6, which will now go left and is only left with the last integer being 5.

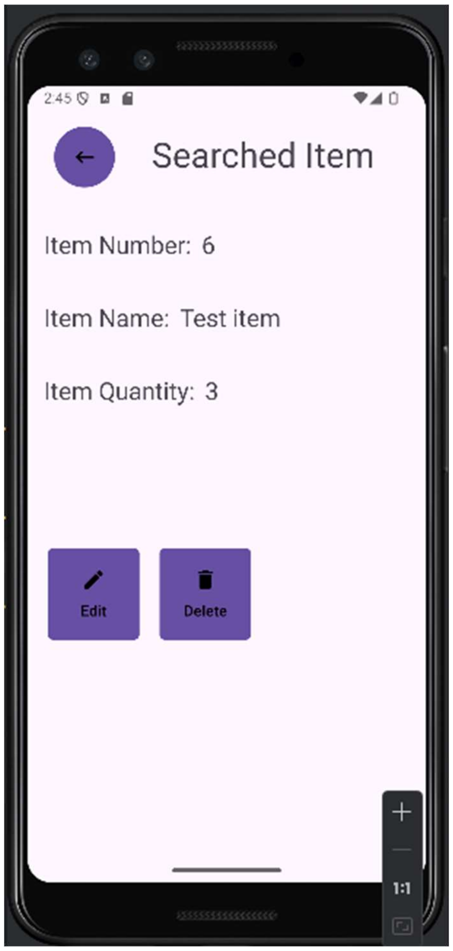


```
1
 \
  2
   \
    3
     \
      4
       \
        5
```

Here, if we were once again looking for 5, it would have to check every single number going down almost like a Linked List, which would result in a time complexity of O(n), which is, as I said, the worst possible case.



| Item Number | Item Name | Item Quantity |
|---|---|---|
| 1 | Test item | 100 |
| 3 | Test item | 77 |
| 6 | Test item | 3 |

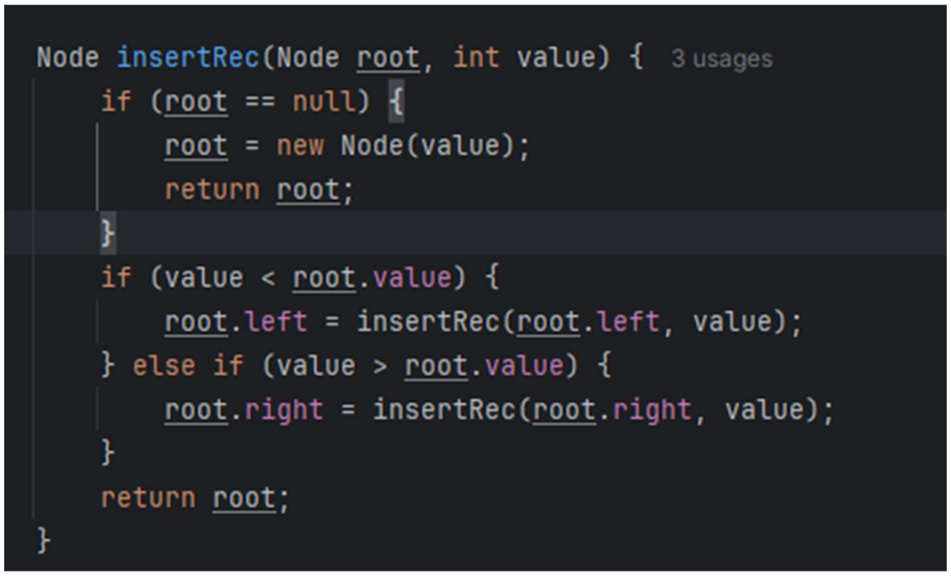


Searched Item

Item Number: 6

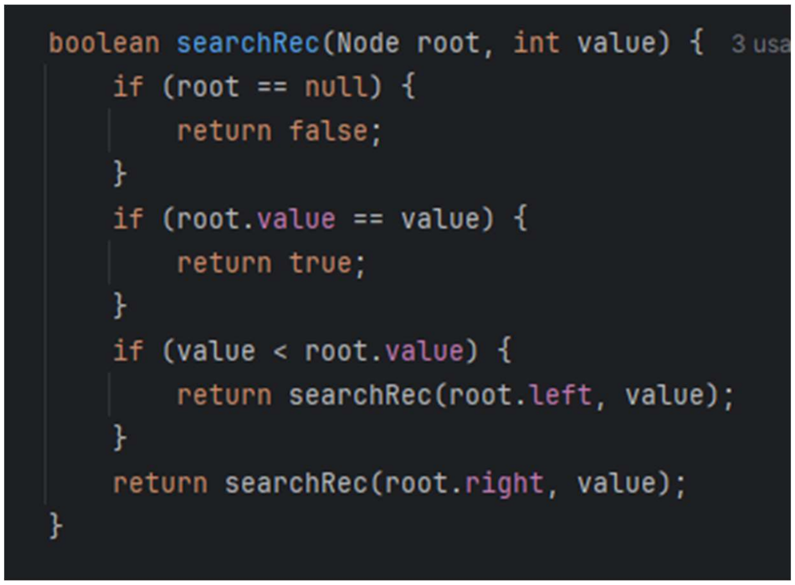
Item Name: Test item

Item Quantity: 3



```
Node insertRec(Node root, int value) {  3 usages
    if (root == null) {
        root = new Node(value);
        return root;
    }
    if (value < root.value) {
        root.left = insertRec(root.left, value);
    } else if (value > root.value) {
        root.right = insertRec(root.right, value);
    }
    return root;
}
```

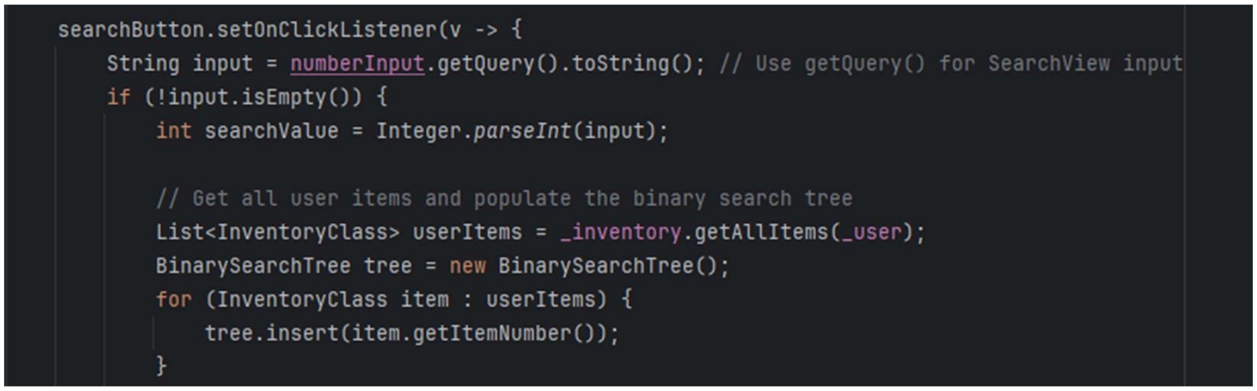
This is how the item numbers are put into the BST before it gets down to the searching aspect. It takes in a new node and checks to see if the value is less than the root value, and if it is, then the number is put on the left. If not, then check if it is larger than the root. If so, then it will go to the right. Then, the root is returned.



```
boolean searchRec(Node root, int value) {  3 usa
    if (root == null) {
        return false;
    }
    if (root.value == value) {
        return true;
    }
    if (value < root.value) {
        return searchRec(root.left, value);
    }
    return searchRec(root.right, value);
}
```

Then, for the search, it's a very similar process, but nothing is being input into the BST, only checked, so it first checks to see if the root is null; if it is, then it returns false, which then displays text on the screen saying the number doesn't exist. Then it checks over and over again until the number is found. In the best-case scenario, the root value is the value we are looking for, but that is extremely unlikely when the database gets many items put into it.

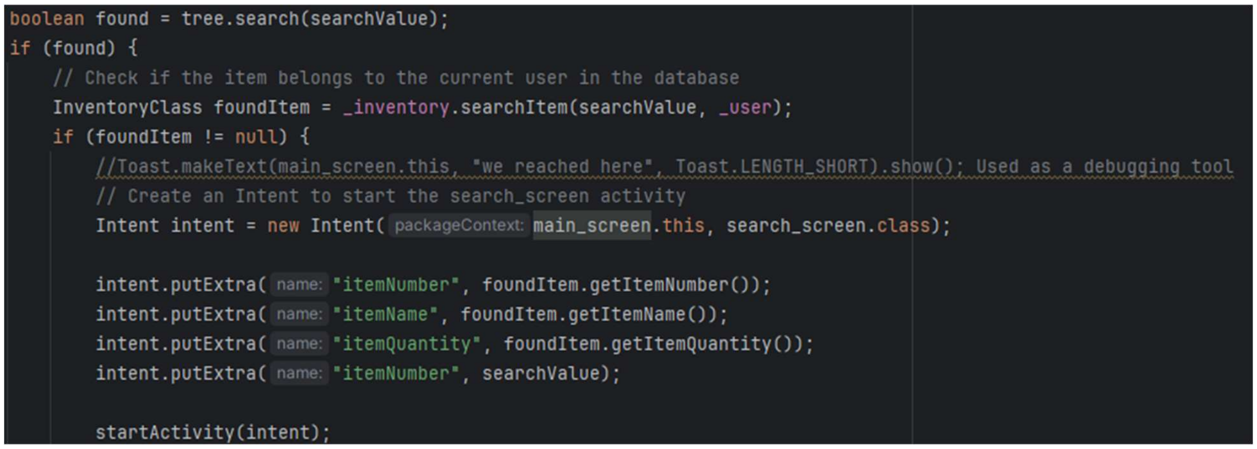
As for how this is all started, we have this code from the main screen:



```java
    searchButton.setOnClickListener(v -> {
        String input = numberInput.getQuery().toString(); // Use getQuery() for SearchView input
        if (!input.isEmpty()) {
            int searchValue = Integer.parseInt(input);

            // Get all user items and populate the binary search tree
            List<InventoryClass> userItems = _inventory.getAllItems(_user);
            BinarySearchTree tree = new BinarySearchTree();
            for (InventoryClass item : userItems) {
                tree.insert(item.getItemNumber());
            }
```

This checks to see if the search bar is empty after the button is clicked, if it is then all of the next code is skipped. It then takes the input and stores it as "searchValue." Then grabs all of the items from the database, then it calls the BinarySearchTree java file and starts the process of inserting each item into the tree.



```java
boolean found = tree.search(searchValue);
if (found) {
    // Check if the item belongs to the current user in the database
    InventoryClass foundItem = _inventory.searchItem(searchValue, _user);
    if (foundItem != null) {
        //Toast.makeText(main_screen.this, "we reached here", Toast.LENGTH_SHORT).show(); Used as a debugging tool
        // Create an Intent to start the search_screen activity
        Intent intent = new Intent( packageContext: main_screen.this, search_screen.class);

        intent.putExtra( name: "itemNumber", foundItem.getItemNumber());
        intent.putExtra( name: "itemName", foundItem.getItemName());
        intent.putExtra( name: "itemQuantity", foundItem.getItemQuantity());
        intent.putExtra( name: "itemNumber", searchValue);

        startActivity(intent);
```

If the item is found with the search, it returns true, allowing the next part to start, which then makes a query in the InventoryDB.java file that searches for that item number where the user is the current one that is logged in. Once grabbed, that data is then sent into the next file, search_screen.java, where it gets displayed for the user to see super easily.

The course outcomes that I achieved with this project are:

- Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices

This is applied since it is all about implementing algorithmic principles to solve a problem. Although at this stage of the application, it may not be a “problem” in the future if this app were to become very popular, it would be mandatory to have not only a search bar but also a way of searching for those items super quickly without causing mass use to the database.

- Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts

This course outcome also applies to this one and the others because of how well-documented each of the files is. Almost anyone with or without coding experience could go through this file and explain to some degree what is going on. There is nothing more that can be done to make these files any more professional quality; my use of comments inside of the files, as well as the headers in each of them, represents how this is professional quality.

To finish this off, this was more than just adding in an algorithm. It was restructuring and revising most of how this project looked. It was far from “visually appealing.” So, I really had two large goals for this project: revising the visual appeal and then working on the logic behind the search for items. I achieved both, everything from the color to the overall structure of app feeling, working, and looking much better.