**Data Structures and Algorithms Narrative**

Jordan Taylor

The CS-360 Mobil Architecture and Programming project provides a perfect starting point for application enhancement. This application provides proof of concept so users can see what an inventory app can do. The application also uses a database and could benefit from an algorithmic touch when dealing with the main list data structure that it uses. For the application to be useful, it needs some enhancements such as a publicly accessible database and search functionality.

Adding search functionality warrants the inclusion of another list. Now, the view that used the existing list in the application uses the filtered list. The list gets filtered based on text that is entered into the search box that I included in the menu area of the main screen. While accomplishing this, I learned a lot about how different components of an application communicate with each other. There are three components that manage the list, and control what the view presents to the user. All of them needed to be modified to incorporate the new filtered list.

I also wanted the list to be sorted. As the list of inventory items grows, if a user wants to scroll through the list of items instead of using the search functionality to filter them, an unsorted list is not a scalable option. I made use of Java’s List.sort() method to do this. Through my research I learned that this method used the TimSort algorithm which can sort a list in O(n log n) average time complexity which is the best average time complexity for mostly unsorted data. This worked out well but presented new problems. What to do when items are added or updated. Originally the application simply added the item to the end of the list. Which is not suitable for a list that is sorted. If I were to re-sort the list, the TimSort algorithm handles the sorting of items that are mostly sorted efficiently with a time complexity that can reach O(n) linear time complexity. I still though that this might not be good enough.

I chose to create a method that is a variant of the binary search algorithm that will return the index of the item that is lexicographically one larger than the item to be inserted. This algorithm can find the index to return from a sorted list by reducing the number of items that it needs to sort through by half for every iteration that it runs. In simpler terms, it runs in O(log n) logarithmic time complexity. The list still needs to be shuffled when the item is inserted, but I believe this sequence of events achieves the most effective solution for the data structure that I chose to use.

Ultimately, I am happy with the solution that I came up with. It gave me more practice with considering time complexity in an application. If the application ever started to have performance issues due to the size of the list, a new data structure could be used such as a hash map or a binary tree. Operations on these two data structures can result in better time complexity for similar operations. With these enhancements I have achieved the course outcomes related to Algorithms and Data Structures.

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen with text

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.