Milestone Three Narrative

CS 499: Capstone

April 2, 2025

Cole Flournoy

The original artifact is an Android mobile app which has features to support a mobile inventory management system. This was the final project for CS 360 - Mobile Architecture and Programming and was written with Java and SQLite. The artifact that I will be submitting for my final ePortfolio is an adapted version of that original app that has been ported over to a completely new tech stack. The new app is built with the MERN stack, which leverages MongoDB, Express, React, and Node to create a full stack web application with dynamic features that expand upon the original functionality required in the mobile app.

I am using the same original artifact for all three of my enhancements, and I actually selected it for this Data Structures and Algorithms category because I thought it *wasn’t* a very good representation of my abilities in this area. For that reason, I wanted to add features to the inventory management app that were a better demonstration of what I’ve learned in this program and my ability to make effective decisions regarding efficiency tradeoffs. The two main features that I added to the web application are search and sort on the primary inventory table. While Javascript does come with a standard Array.sort function that makes some internal efficiency decisions based on the data passed in, I thought it would be more fun (and more interesting for future employers) to implement my data sorting from scratch. With that in mind, I wrote my own version of three commonly used sorting algorithms (merge sort, quick sort, and bubble sort) and created a UI for users to select their preferred algorithm when sorting. Implementations of the same sorting algorithm can vary significantly, so I tried to write mine in a way that maximizes the time and space efficiency of each algorithm. For table search, the method of searching isn’t particularly unique (filtering an array of objects), but there were some interesting tradeoffs I had to consider during the process. For example, my search isn’t as efficient as possible because I decided to prioritize writing the algorithm in a way that it could easily be reused for additional tables in the future. I also made an efficiency sacrifice in order to provide what I think is a better user experience, because I decided to allow the search to look through all values on an object (rather than only searching the item name, for example).

I believe I met my planned course outcome (which was #3) with this enhancement because I used algorithmic principles to provide value to users in a way that effectively weighs the tradeoffs associated with my design choices and implementations. I don’t have any updates to my outcome coverage plans.

Diving into sorting algorithms and big O notation for this enhancement was really fun. Visualizing each step of the process was really important for me when researching and implementing these sorting algorithms, especially for the ones that can be implemented with recursion. Recursion itself isn’t a complicated concept, but sometimes I have trouble implementing recursive functions. I’m so used to the inputs and outputs of a function being separate that I can forget to structure my functions in a way that allows them to pass data recursively. The biggest challenge for me with this enhancement was actually state management. When I had a simple table receiving rows from a parent component, things were relatively simple, but once I introduced sorting and then later search the state management got much more complicated. The fact that these features interact, meaning that the search-filtered table entries should also be updated if the sorting selection changes, lead to frequent questions about what piece of state should be the source of truth for the table row data and how can everything be efficiently updated. I went through a lot of iterations that included unnecessary state objects before I arrived at what is hopefully a pretty streamlined implementation.