Austin Antles

CS 499

July 24 2025

### Artifact Enhancement Narrative: Algorithms and Data Structures

**1. Artifact Description**

The original artifact I am enhancing is the data retrieval logic from my Android inventory management application, created in CS 360. In its initial form, the application's approach to data structures and algorithms was rudimentary. The primary data operation was a single, inflexible method, getAllInventoryItems, which executed a hardcoded SELECT \* SQL query. The result was loaded into a Java ArrayList, a basic linear data structure. This meant that to display or find any information, the entire dataset had to be loaded into the client device's memory first, with no efficient means of searching or sorting provided by the backend logic.

**2. Justification for Inclusion and Enhancement**

I selected this artifact because its initial implementation represents a common but inefficient approach to data handling that is not scalable. It provides a clear opportunity to demonstrate my understanding of algorithmic efficiency and the importance of server-side data processing. The original method's O(n) complexity for any search operation (where 'n' represents the total number of items) and its high memory usage make it a prime candidate for significant algorithmic enhancement.

The artifact was improved by implementing a sophisticated, server-side dynamic search and sorting algorithm within the Go backend. This new algorithm, exposed via the /api/v1/items/search endpoint, showcases several key skills:

* **Algorithmic Design:** The core of the enhancement is the searchItemsHandler function. This algorithm intelligently constructs a SQL query string by dynamically appending WHERE and ORDER BY clauses based on optional user-provided parameters. This is a far more complex and efficient approach than fetching all the data at once.
* **Data Structure Manipulation:** The algorithm uses a slice of interfaces ([]interface{}) to collect and pass parameters to the database driver safely. This technique is crucial for creating secure prepared statements and preventing SQL injection vulnerabilities.
* **Efficiency and Scalability:** By offloading the work of filtering and sorting to the PostgreSQL database engine (which can use indexes for searches closer to O(log n) complexity), the algorithm ensures that the client application remains fast and responsive, regardless of the size of the inventory.

**3. Course Outcome Alignment**

This enhancement directly meets the course outcome I planned in Module One:

* **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 enhancement is a direct application of the algorithmic tenets to solve the problem of efficiently finding specific data within a large dataset. I evaluated the original client-side iteration approach and identified its poor performance as a key trade-off. My new server-side algorithm is a standard computer science practice that strikes the correct balance: it increases backend complexity in exchange for a significant improvement in client-side performance, network efficiency, and overall application scalability.

**4. Reflection on the Process**

Enhancing this artifact was a challenging and highly educational process that moved me from simply retrieving data to intelligently processing it. The biggest challenge was determining how to construct a SQL query dynamically in a manner that was both flexible and secure. My initial thought was to use simple string concatenation to build the entire query, but I quickly learned through research that this method is wide open to SQL injection attacks.

The key thing I learned was the importance of prepared statements and how to use them with a dynamic number of parameters. The solution was to build the query structure with placeholders (such as $1, $2) and separately create a list (a slice in Go) of the actual values. This separation ensures that user input is treated as data, not as executable code, which was a significant "aha!" moment for me in understanding backend security. I also faced the challenge of handling multiple optional parameters gracefully. The series of if statements in my searchItemsHandler, which checks if a parameter exists before adding it to the query and the arguments list, is a direct result of that challenge. This process taught me that designing a good algorithm isn't just about the logic; it's also about anticipating edge cases and writing defensive, secure code.