Elijah Harris

# Algorithms and Data Structure

## Original Code Overview

The original implementation of the Mass Tracker App relied on an ArrayList to store and manage mass logs. While functional, this approach was inefficient for operations like searching, sorting, and displaying data dynamically. The reliance on ArrayList made lookups and complex operations computationally expensive, with linear time complexity, which could negatively impact performance as the dataset grew. Furthermore, the user interface lacked advanced features such as pagination, sorting, and searching, limiting its usability and scalability.

To address these shortcomings, I enhanced the application by introducing a more efficient data structure, implementing a search algorithm, adding pagination for better data navigation, and enabling sorting by multiple criteria. These enhancements significantly improved the system's performance, flexibility, and usability.

## Enhancement Objectives

The main goals of the refactor were:

1. **Optimize Data Storage:** Replace the ArrayList with a HashMap to improve lookup efficiency and facilitate advanced operations.
2. **Add Search and Sorting Algorithms:** Implement a search algorithm for finding logs nearest to a given date or mass and enable dynamic sorting by date or mass in ascending or descending order.
3. **Increase Efficiency and Scalability:** Introduce pagination for better data navigation and performance, especially for larger datasets.
4. **Enhance User Experience:** Ensure the user interface dynamically adapts to the user's needs, providing sorted, searchable, and paginated views of mass logs.

# Key Enhancements

Replaced ArrayList with HashMap

* + **Original Issue:** The use of ArrayList resulted in linear time complexity for lookups and lacked the ability to store data in a structured, easily searchable format.
  + **Enhancement**: I replaced the ArrayList with a HashMap, where dates are used as keys and masses as values.
  + **Impact**: The HashMap structure allows for quick retrieval of data, enabling efficient implementation of advanced features like nearest-value searches.

### Pagination

* + **Original Issue**: The entire dataset was displayed at once, which was cumbersome for users when navigating large datasets.
  + **Enhancement**: Implemented a pagination mechanism that divides the logs into smaller, manageable chunks.
  + **Impact**: This feature improves usability and performance by limiting the amount of data displayed at any given time, making the application more responsive.

Dynamic Sorting

* + **Original Issue**: The system lacked sorting functionality, forcing users to manually search for specific entries.
  + **Enhancement**: Added dynamic sorting functionality, allowing users to sort mass logs by date or mass in ascending or descending order. Sorting was implemented using comparator-based logic for flexibility and performance.
  + **Impact**: Sorting enables users to organize data according to their preferences.

Nearest-Value Search Algorithm

* + **Original Issue**: Users could not efficiently find logs close to a specific date or mass.
  + **Enhancement**: Implemented algorithms to find the nearest log by date or mass:
    - **Nearest Mass**: Utilizes absolute difference comparisons to locate the closest value.
    - **Nearest Date**: Compares date strings to identify the closest match.
  + **Impact**: These algorithms allow users to locate specific logs quickly, improving their ability to analyze and retrieve relevant data.

## Alignment with Course Outcomes

**1. Employing Strategies for Collaboration**

* The refactored data handling makes the codebase more modular and easier for team members to understand and extend. The well-documented search and sorting algorithms facilitate collaboration by providing clear, reusable components.

**2. Designing Professional-Quality Communications**

* The enhancements communicate their functionality effectively through clear code comments and a well-structured implementation. Pagination, sorting, and searching provide users with a cohesive and intuitive experience, showcasing professional-quality design.

**3. Designing and Evaluating Computing Solutions**

* The decision to adopt a HashMap and implement algorithms for advanced features demonstrates the ability to evaluate and implement efficient solutions to complex problems while managing trade-offs such as memory usage versus performance.

**4. Demonstrating Techniques and Tools**

* The sorting algorithms and nearest-value search highlight the use of innovative techniques to solve practical problems, while the pagination feature demonstrates the ability to design scalable solutions for large datasets.

**5. Developing a Security Mindset**

* Centralizing data operations through well-defined methods enables better validation of user inputs and prevents inconsistencies or vulnerabilities in data manipulation.

## Reflection on the Process

Refactoring the data structure and implementing advanced algorithms significantly enhanced the application’s efficiency and usability. One challenge was ensuring that the HashMap structure supported all required functionalities, such as dynamic sorting and nearest-value searches. This was addressed by designing helper methods to extract and process data from the HashMap when necessary. Another challenge was ensuring that pagination worked seamlessly with sorting and searching, requiring careful integration of these features.

These enhancements reinforced the importance of data structures in software design and the value of modular, reusable algorithms in creating efficient and scalable systems. By optimizing the application’s data handling, I demonstrated proficiency in designing practical solutions that align with industry standards and user needs.

This process not only improved the app’s functionality but also deepened my understanding of algorithms and data structures, preparing me for future challenges in the field of computer science.