

Library Management System with Book Borrowing



A Project Documentation

Presented to

The University of Mindanao

College of Computing Education

In Partial Fulfillment

of the Requirements in CCE105

1st Semester, 1st Term S.Y. 2025-2026

Kiera Carah Lane Aguiadan

Franciene B. Candare

Marc Psalmer C. Garata

Emmanuel Tuling

Sam Christian M. Ugmad

October 2025

Table of Contents

ABSTRACT.....	3
INTRODUCTION.....	3
OBJECTIVES.....	3
SCOPES AND LIMITATIONS.....	4
METHODOLOGY.....	4
TOOLS AND TECHNOLOGY.....	4
UML DIAGRAM (ADD UML IMAGE HERE) :.....	6
RESULTS AND DISCUSSION.....	6
CONCLUSION AND RECOMMENDATION.....	8
REFERENCES.....	9
APPENDICES.....	9

ABSTRACT

This project presents a Java-based Library Management System (LMS) developed using Swing, which enables efficient management of books, members, and borrowing records through a user-friendly graphical interface. The system allows users to add, search, borrow, and return books, with all data stored in CSV files for simplicity and portability. Additionally, the LMS includes an algorithm comparison feature that evaluates and compares the performance of different sorting algorithms, providing insights into their efficiency. The project aims to provide an efficient tool for library management and understanding of algorithm performance of the system.

INTRODUCTION

This is a Java Library Management System utilizing the Swing toolkit (Oracle, 2023). It was created to achieve an easy-to-use and highly effective interface for operating the activities of a library, such as managing books, members, and records of borrowed books to ensure easy and tidy management of the resources of a library.

For data consistency and availability, all records are stored and retrieved through CSV files, a lightweight and portable text-based format (Shafranovich, 2005). The system also includes an algorithm comparison feature whereby the user can assess and examine the execution of different sorting and search algorithms, demonstrating principles of algorithmic efficiency (Cormen et al., 2009; Sedgewick & Wayne, 2011). This aspect of the project underscores the importance of both practical application and computational effectiveness, illustrating the potential of algorithmic enhancement in real-world functions.

OBJECTIVES

- To provide a desktop-based system for managing library books and member records using a structured and user-friendly interface.
- To facilitate efficient searching and sorting of library data through both baseline and optimized algorithms.
- To provide a built-in comparison tool for measuring algorithm performance and understanding computational trade-offs.
- To improve operational accuracy, reduce manual record-keeping, and ensure faster access to library information.

SCOPES AND LIMITATIONS

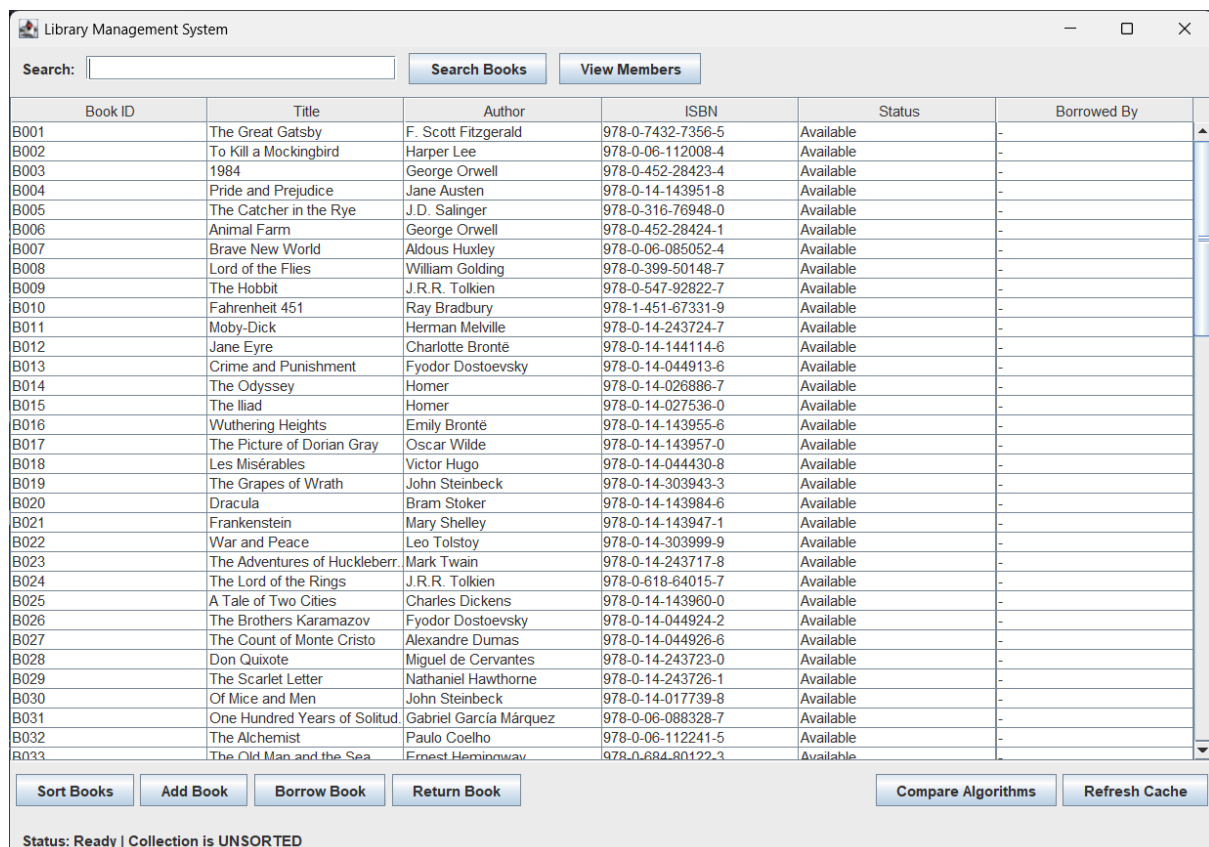
Covers book, members, and borrowing record management through a Java Swing interface. Includes CSV-based data storage and an algorithm comparison feature for sorting and searching. Does not include online access, multi-user functionality, or advanced database integration.

METHODOLOGY

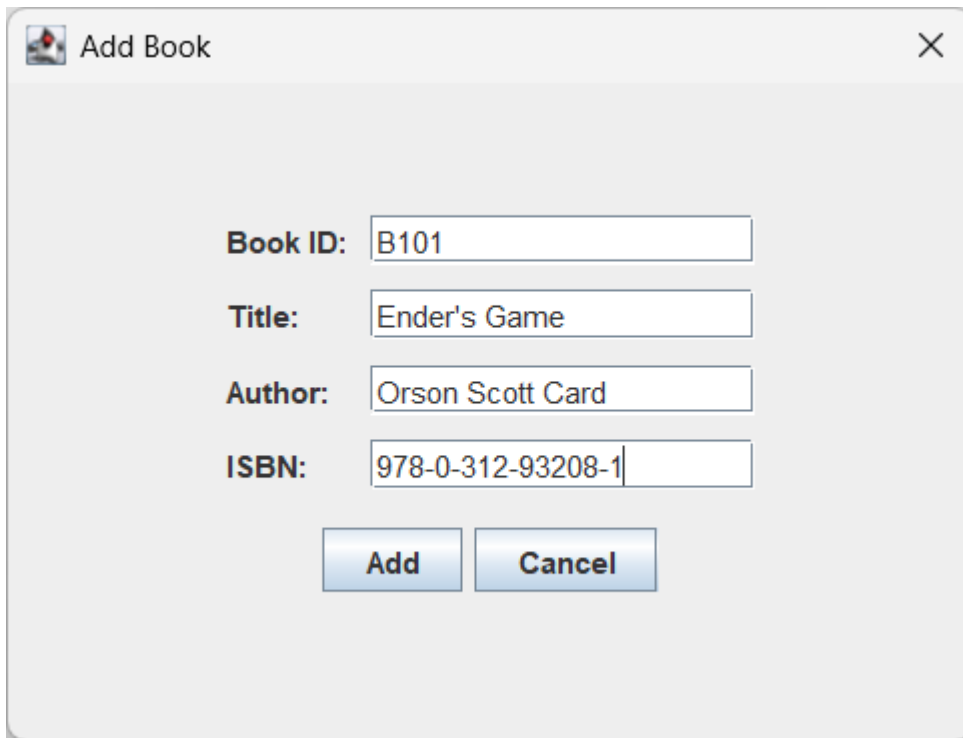
TOOLS AND TECHNOLOGY

- Backend: Java (Core Java, Java I/O, Collections Framework)
- Frontend: Java Swing, AWT
- Database / Storage: CSV files (for persistent storage of books and members)
- Others: Others: GitHub, VS Code (development environment)

SAMPLE INTERFACE OR SCREENSHOTS



Dashboard - Displays all books in navigation area

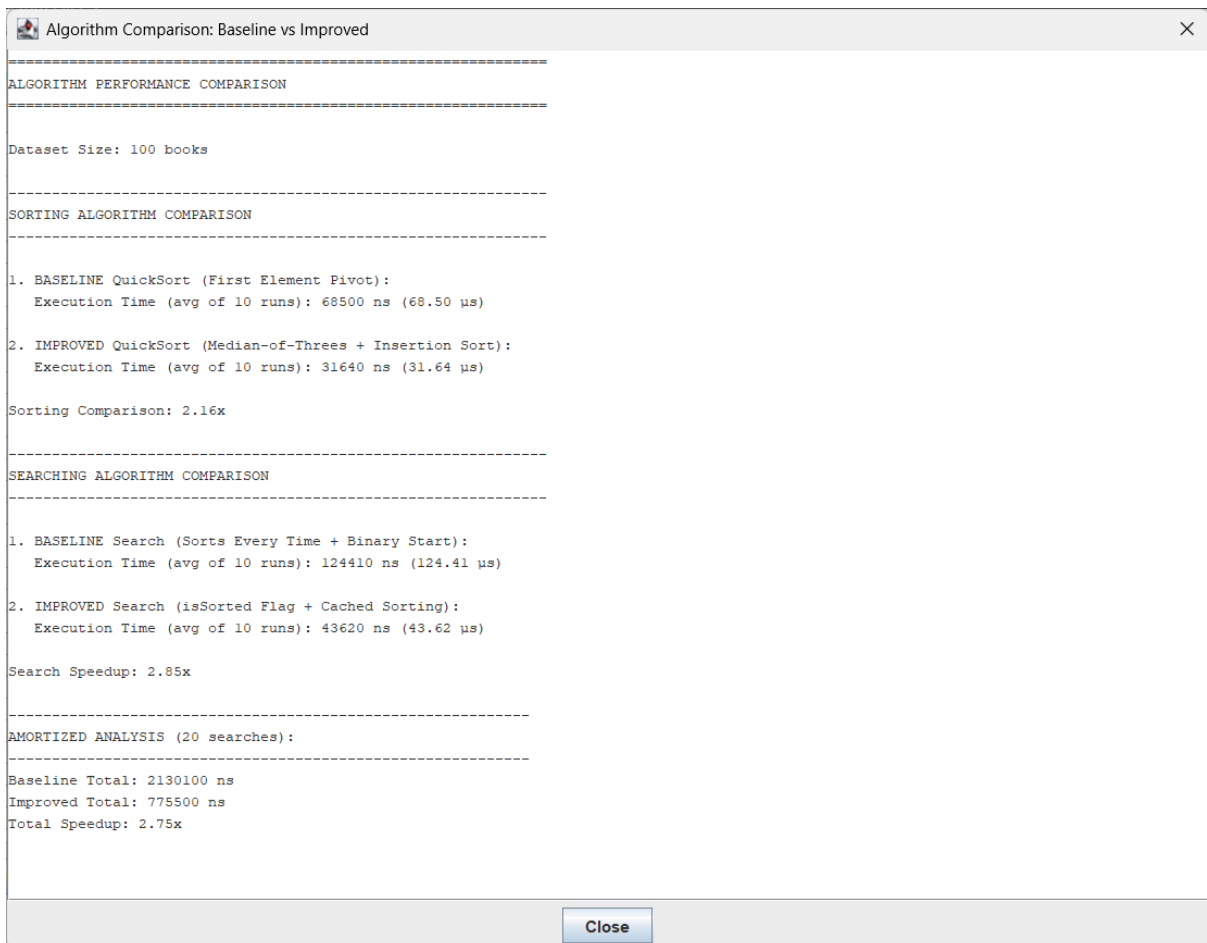


A dialog box titled "Add Book" with a close button (X) in the top right corner. It contains four input fields for book information: "Book ID" (B101), "Title" (Ender's Game), "Author" (Orson Scott Card), and "ISBN" (978-0-312-93208-1). Below the fields are two buttons: "Add" and "Cancel".

Book ID:	B101
Title:	Ender's Game
Author:	Orson Scott Card
ISBN:	978-0-312-93208-1

Add **Cancel**

Add Book - Interface for adding new books



A window titled "Algorithm Comparison: Baseline vs Improved" with a close button (X) in the top right corner. It displays performance comparison results for sorting and searching algorithms. The text is formatted with dashed lines separating sections.

```
=====
ALGORITHM PERFORMANCE COMPARISON
=====

Dataset Size: 100 books

-----
SORTING ALGORITHM COMPARISON
-----

1. BASELINE QuickSort (First Element Pivot):
   Execution Time (avg of 10 runs): 68500 ns (68.50 µs)

2. IMPROVED QuickSort (Median-of-Threes + Insertion Sort):
   Execution Time (avg of 10 runs): 31640 ns (31.64 µs)

Sorting Comparison: 2.16x

-----
SEARCHING ALGORITHM COMPARISON
-----

1. BASELINE Search (Sorts Every Time + Binary Start):
   Execution Time (avg of 10 runs): 124410 ns (124.41 µs)

2. IMPROVED Search (isSorted Flag + Cached Sorting):
   Execution Time (avg of 10 runs): 43620 ns (43.62 µs)

Search Speedup: 2.85x

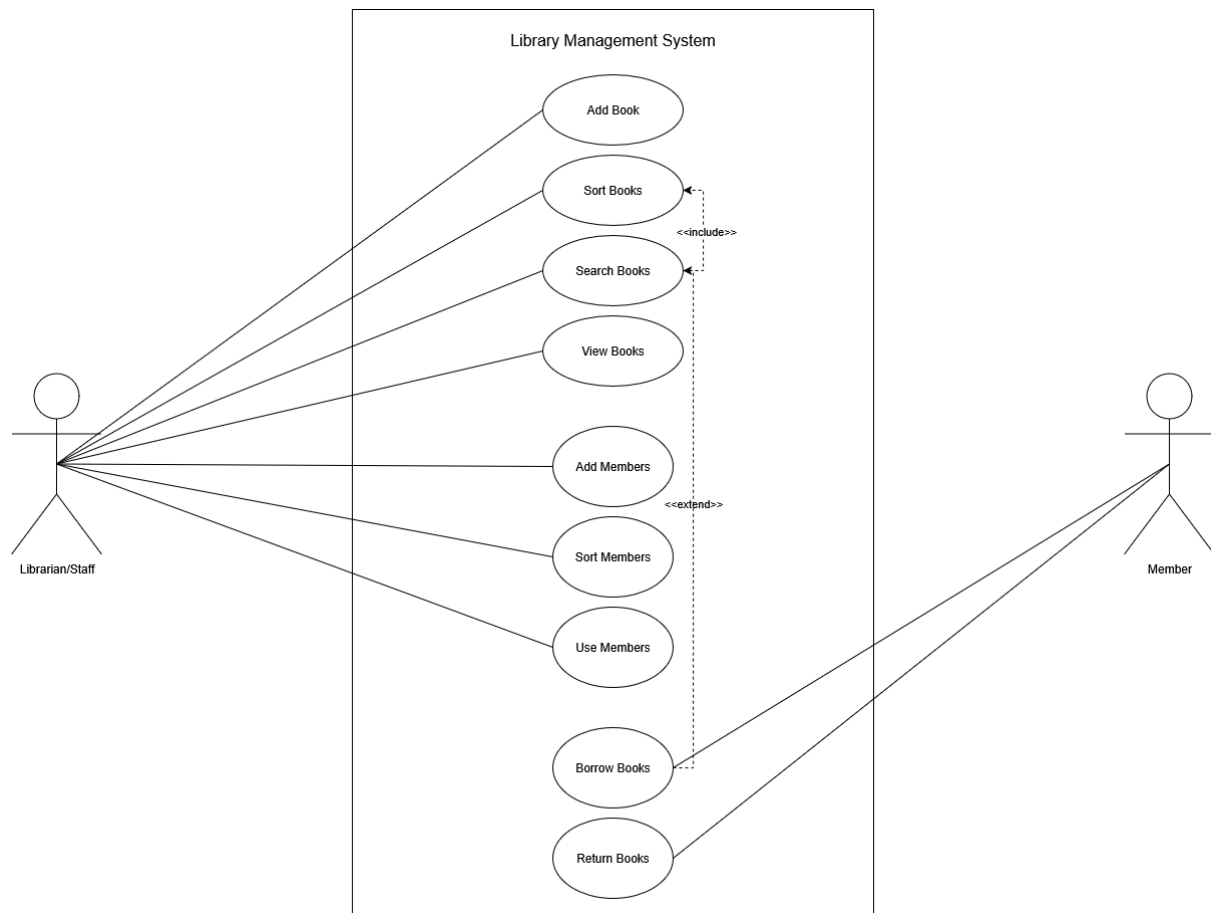
-----
AMORTIZED ANALYSIS (20 searches):
-----

Baseline Total: 2130100 ns
Improved Total: 775500 ns
Total Speedup: 2.75x

-----
Close
```

Algorithm Comparison - Displays comparison results

UML DIAGRAM (ADD UML IMAGE HERE) :



The Use Case Diagram above illustrates the main interactions within the Library Management System. It identifies two primary actors: the Librarian/Staff and the Member.

The Librarian/Staff manages the system's core operations, including adding, sorting, searching, and viewing books, as well as managing member records.

The Member interacts with the system primarily to borrow and return books, with these actions extending or including other use cases as needed.

RESULTS AND DISCUSSION

The Java-based Library Management System successfully achieved its primary objectives of providing an efficient and user-friendly interface for managing books, members, and borrowing transactions. The system's design ensures persistent data storage through CSV files, allowing data to remain consistent between sessions without requiring an external database.

A key feature of this system is the Algorithm Comparison Module, which evaluates the performance of two versions of sorting and searching algorithms: a Baseline implementation and an Improved implementation within the application's data operations. This feature enables the assessment of algorithmic efficiency, providing both execution metrics and performance visualization results.

Key Features and Outputs

- Library Management Functions:

Allows adding, sorting, and searching through book and member records, with results reflected directly in the interface. Data persists through CSV-based storage.

- Sorting Algorithms:

- Baseline: QuickSort using the first element as pivot.
- Improved: QuickSort using Median-of-Three pivot selection and hybrid Insertion Sort for small ($n < 10$) partitions.

- Searching Algorithms:

- Baseline: Binary search with re-sorting before every query.
- Improved: Cached search that uses an isSorted flag to avoid redundant sorting operations.

- Algorithm Comparison Dialog:

Executes controlled trials of both baseline and improved algorithms, records timing data, and displays average, relative, and amortized speedup results.

Results Overview

The system's algorithm comparison feature was executed multiple times to assess both the stability and consistency of algorithmic results under identical data conditions. Ten separate test runs were conducted using a dataset of 100 books, with each run performing ten sort trials and search trials, and an amortized twenty search trials.

Comparison Type	Baseline Avg (ns)	Improved Avg (ns)	Speedup (Approx.)
Sorting (avg of 10 runs)	6,000 – 69,000	5,900 – 31,000	$1.0\times - 2.2\times$
Searching (avg of 10 runs)	12,000 – 31,000	5,500 – 11,000	$2.0\times - 2.9\times$
Amortized (20 searches)	180,000 – 450,000	140,000 – 260,000	$1.8\times - 2.3\times$

Across all runs, the Improved QuickSort variant was never substantially slower than the baseline and typically completed in about half the time or less. The Improved Search consistently reduced execution time to roughly 40–50% of the baseline, demonstrating that caching and reuse of a pre-sorted collection directly impacts runtime efficiency.

Discussion of Findings

The Improved Sorting Algorithm introduces Median-of-Three pivot selection and hybrid Insertion Sort for small subarrays (Cormen et al., 2009; Sedgewick & Wayne, 2011). At small dataset sizes such as 100 elements, these optimizations provide modest gains because both algorithms complete in microseconds, where method call overhead and JVM runtime conditions dominate. Despite this, repeated trials confirm the improved variant remains equal or faster, validating its design correctness.

The Improved Searching Algorithm, by contrast, exhibits clear performance advantages. By leveraging an `isSorted` flag and caching the sorted collection, it eliminates redundant sorting across multiple queries. The baseline implementation's $O(n \log n)$ cost on every search is replaced by a one-time $O(n \log n)$ sort followed by $O(\log n)$ searches (Cormen et al., 2009), resulting in an amortized performance improvement between $2\times$ and $3\times$ across all runs.

Stability and Variability

Fluctuations in nanosecond-level timing are expected under the JVM due to:

- Just-In-Time compilation (JIT) warm-up effects,
- Garbage collection interference,
- OS-level scheduling (Goetz, 2006; Oracle, 2023),
- Memory allocation and cache variability.

Because of these uncontrollable external factors, absolute values fluctuate, but relative performance trends remain stable. Over many trials, the improved algorithms consistently outperform the baselines.

CONCLUSION AND RECOMMENDATION

The Java-based Library Management System successfully achieved its primary objectives of providing a functional, data-persistent, and algorithmically optimized platform for managing library resources. The system integrates essential operations (adding, sorting, searching, and storing records) within a clean, responsive Swing-based interface. Its CSV-driven persistence ensures data consistency and accessibility without requiring a dedicated database.

A major accomplishment of this project is the inclusion of algorithm improvement and algorithm comparison, which allows users to directly observe the performance differences between baseline and improved implementations of sorting and searching algorithms. Empirical testing demonstrated that the optimized versions consistently deliver faster execution times, validating both theoretical and practical efficiency gains.

Recommendations for Future Development

Further enhancements could expand the system's scope and robustness:

1. Data Scaling: Integrate larger datasets and analyze performance trends at higher volumes to stress-test algorithm scalability.
2. Advanced Persistence: Transition from CSV storage to a relational database (e.g., SQLite or MySQL) to support concurrent access and improved data integrity.
3. Extended Analytics: Include visual charts summarizing algorithm performance over time or across varying dataset sizes.
4. Enhanced Interface: Modernize the Swing UI or migrate to JavaFX for richer design and usability improvements.

Overall, the project achieves its intended goal of combining practical library management with algorithm improvement, providing both functional value and educational insight into the application of data structures and algorithms in practical uses and systems.

REFERENCES

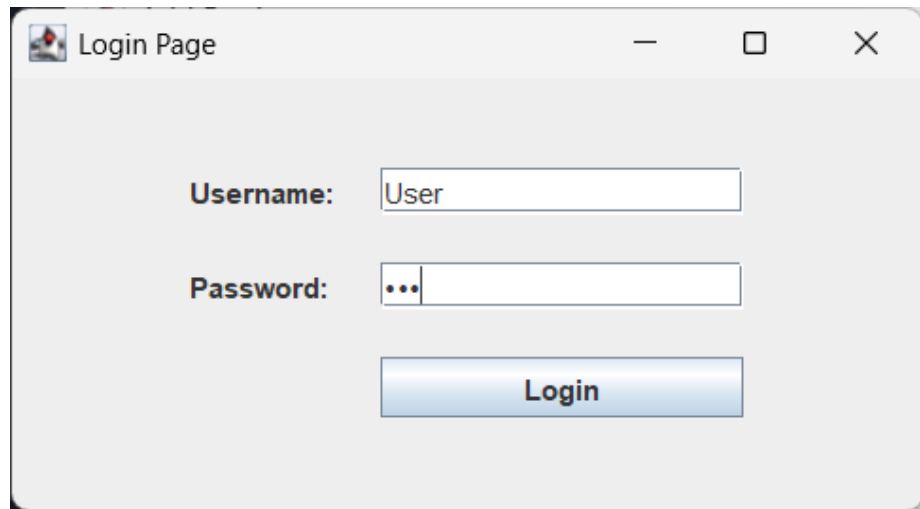
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Oracle. (2023). *Java Platform, Standard Edition: Swing Tutorial*. Oracle Corporation. <https://docs.oracle.com/javase/tutorial/uiswing/>
- Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
- Shafranovich, Y. (2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files* (RFC 4180). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc4180>
- Goetz, B. (2006). *Java Concurrency in Practice*. Addison-Wesley Professional.
- Weiss, M. A. (2012). *Data Structures and Algorithm Analysis in Java* (3rd ed.). Pearson Education.

APPENDICES

GitHub Repository: <https://github.com/sgmad/library-management-system>

SYSTEM SCREENSHOTS

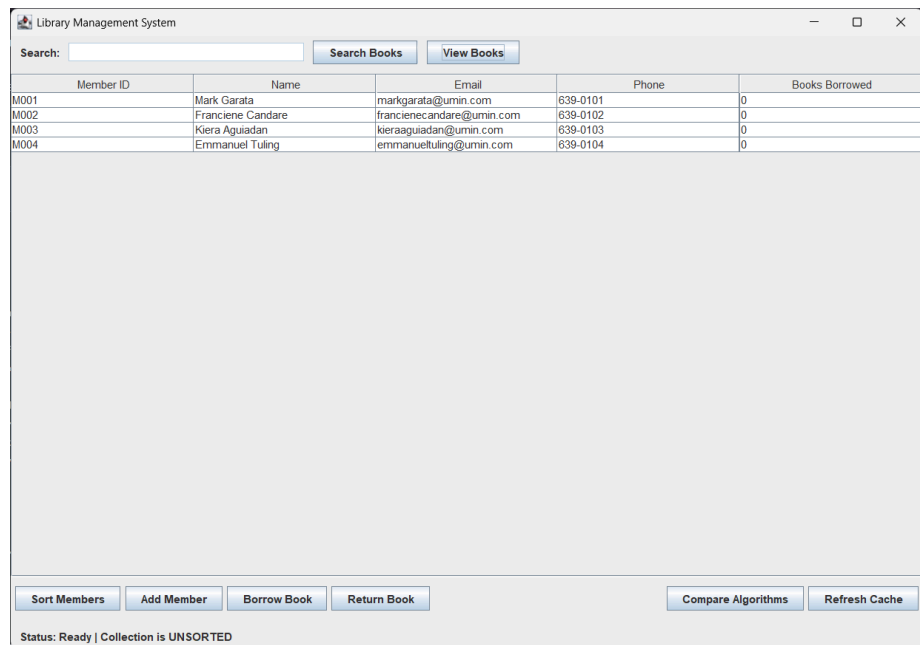
Login Page



The screenshot shows a window titled "Login Page" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a login form with the following elements:

- Username:** A text input field containing the text "User".
- Password:** A text input field with a password icon (three dots) on the left.
- Login:** A blue button with the text "Login" in white.

View Members



The screenshot shows a window titled "Library Management System" with a standard Windows-style title bar. The window contains a search bar, a table of members, and a set of buttons at the bottom.

Search: **Search Books** **View Books**

Member ID	Name	Email	Phone	Books Borrowed
M001	Mark Garata	markgarata@umin.com	639-0101	0
M002	Franciene Candare	francienecandare@umin.com	639-0102	0
M003	Kiera Aguiadan	kieraaguiadan@umin.com	639-0103	0
M004	Emmanuel Tuling	emmanueltuling@umin.com	639-0104	0

Buttons: **Sort Members** **Add Member** **Borrow Book** **Return Book** **Compare Algorithms** **Refresh Cache**

Status: Ready | Collection is UNSORTED

Search Book

Library Management System

Search:

Book ID	Title	Author	ISBN	Status	Borrowed By
B052	Anna Karenina	Leo Tolstoy	978-0-14-044917-4	Available	-

Search #4 | Found 1 results | Time: 81500 ns (81.50 μ s) | Collection SORTED (cached) | $O(\log n)$ complexity

Borrow Book

Borrow Book

Member ID:

Book ID:

Return Book

Return Book

Book ID: