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 P. Tuling
Sam Christian M. Ugmad

October 2025

Table of Contents

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. It also introduces a custom Abstract Data Type (ADT) for managing book collections, providing controlled data handling and supporting the integration of sorting and searching algorithms. 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 a deeper understanding of algorithmic design and performance within structured data systems.

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 applications.

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

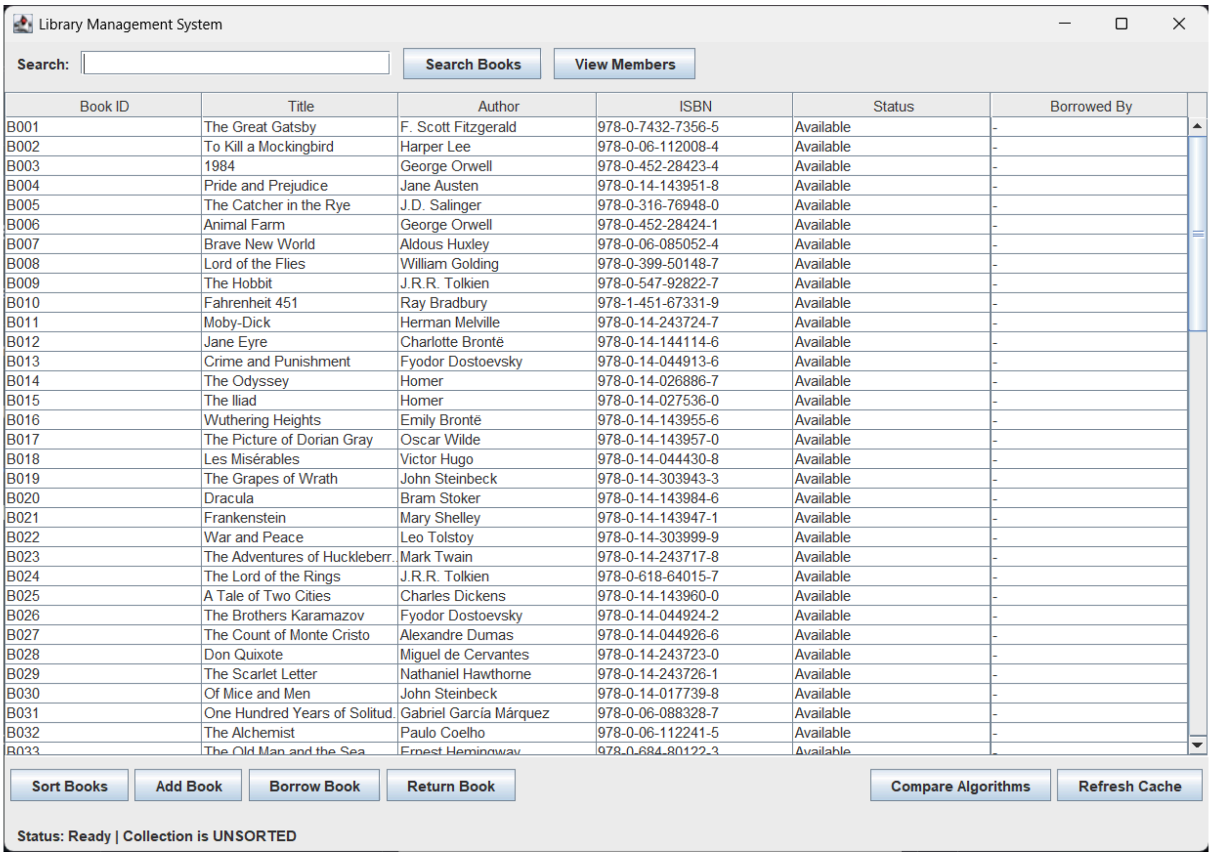
Covers book, members, and borrowing record management through a Java Swing interface. Includes CSV-based data storage and an algorithm comparison feature for sorting and searching. Does not include online access, multi-user functionality, or advanced database integration.

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: GitHub, VS Code (development environment)

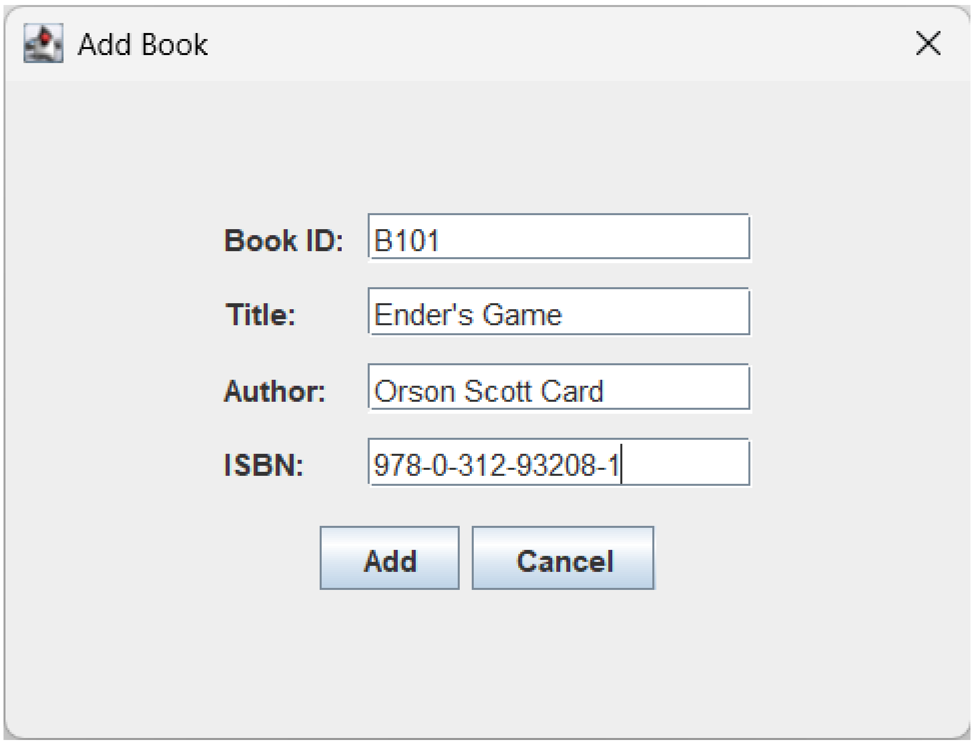
SAMPLE INTERFACE OR SCREENSHOTS



Book ID	Title	Author	ISBN	Status	Borrowed By
B001	The Great Gatsby	F. Scott Fitzgerald	978-0-7432-7356-5	Available	-
B002	To Kill a Mockingbird	Harper Lee	978-0-06-112008-4	Available	-
B003	1984	George Orwell	978-0-452-28423-4	Available	-
B004	Pride and Prejudice	Jane Austen	978-0-14-143951-8	Available	-
B005	The Catcher in the Rye	J.D. Salinger	978-0-316-76948-0	Available	-
B006	Animal Farm	George Orwell	978-0-452-28424-1	Available	-
B007	Brave New World	Aldous Huxley	978-0-06-085052-4	Available	-
B008	Lord of the Flies	William Golding	978-0-399-50148-7	Available	-
B009	The Hobbit	J.R.R. Tolkien	978-0-547-92822-7	Available	-
B010	Fahrenheit 451	Ray Bradbury	978-1-451-67331-9	Available	-
B011	Moby-Dick	Herman Melville	978-0-14-243724-7	Available	-
B012	Jane Eyre	Charlotte Brontë	978-0-14-144114-6	Available	-
B013	Crime and Punishment	Fyodor Dostoevsky	978-0-14-044913-6	Available	-
B014	The Odyssey	Homer	978-0-14-026886-7	Available	-
B015	The Iliad	Homer	978-0-14-027536-0	Available	-
B016	Wuthering Heights	Emily Brontë	978-0-14-143955-6	Available	-
B017	The Picture of Dorian Gray	Oscar Wilde	978-0-14-143957-0	Available	-
B018	Les Misérables	Victor Hugo	978-0-14-044430-8	Available	-
B019	The Grapes of Wrath	John Steinbeck	978-0-14-303943-3	Available	-
B020	Dracula	Bram Stoker	978-0-14-143984-6	Available	-
B021	Frankenstein	Mary Shelley	978-0-14-143947-1	Available	-
B022	War and Peace	Leo Tolstoy	978-0-14-303999-9	Available	-
B023	The Adventures of Huckleberr...	Mark Twain	978-0-14-243717-8	Available	-
B024	The Lord of the Rings	J.R.R. Tolkien	978-0-618-64015-7	Available	-
B025	A Tale of Two Cities	Charles Dickens	978-0-14-143960-0	Available	-
B026	The Brothers Karamazov	Fyodor Dostoevsky	978-0-14-044924-2	Available	-
B027	The Count of Monte Cristo	Alexandre Dumas	978-0-14-044926-6	Available	-
B028	Don Quixote	Miguel de Cervantes	978-0-14-243723-0	Available	-
B029	The Scarlet Letter	Nathaniel Hawthorne	978-0-14-243726-1	Available	-
B030	Of Mice and Men	John Steinbeck	978-0-14-017739-8	Available	-
B031	One Hundred Years of Solitud...	Gabriel García Márquez	978-0-06-088328-7	Available	-
B032	The Alchemist	Paulo Coelho	978-0-06-112241-5	Available	-
B033	The Old Man and the Sea	Ernest Hemingway	978-0-684-80122-3	Available	-

Status: Ready | Collection is UNSORTED

Dashboard - Displays all books in navigation area



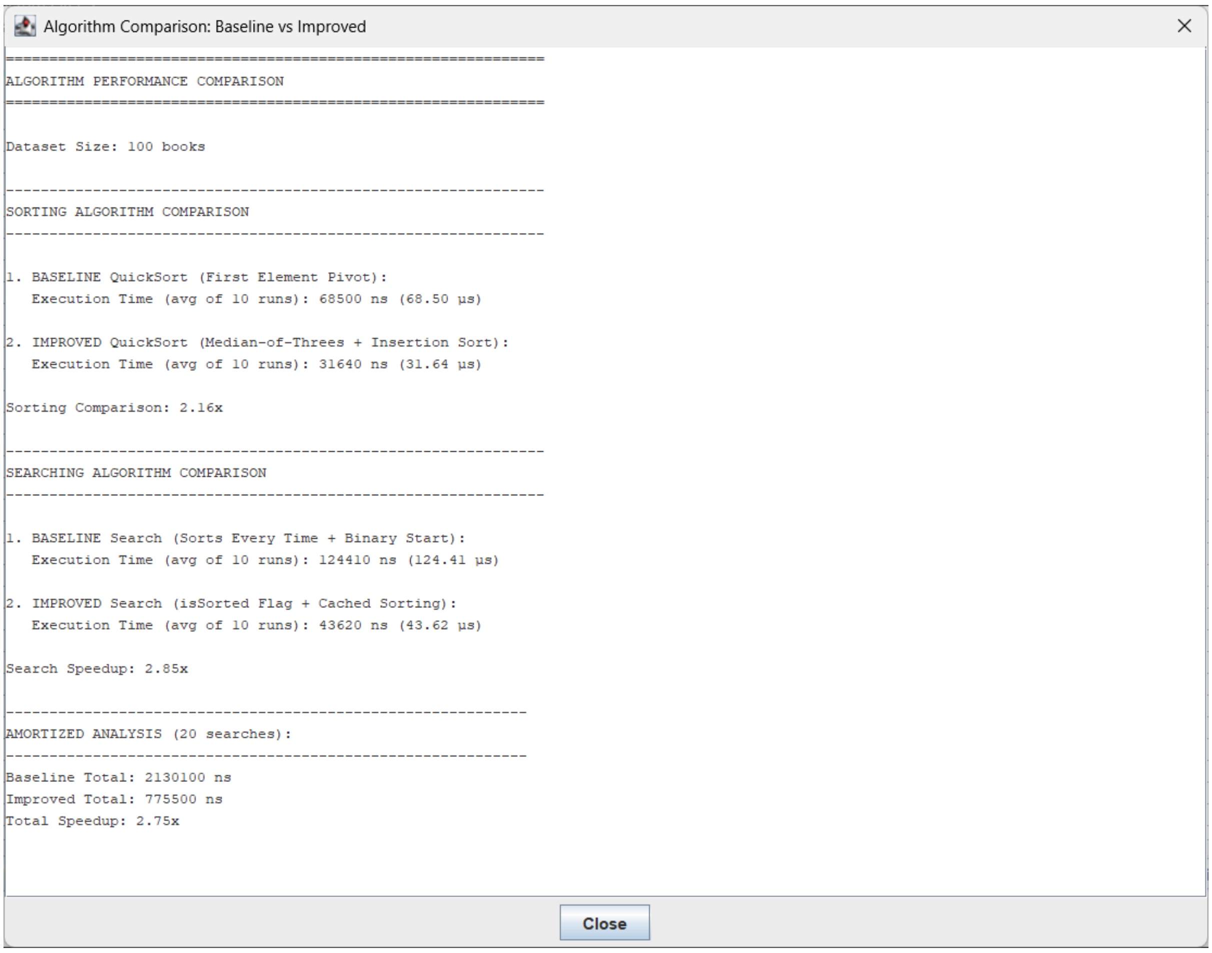
Book ID: B101

Title: Ender's Game

Author: Orson Scott Card

ISBN: 978-0-312-93208-1

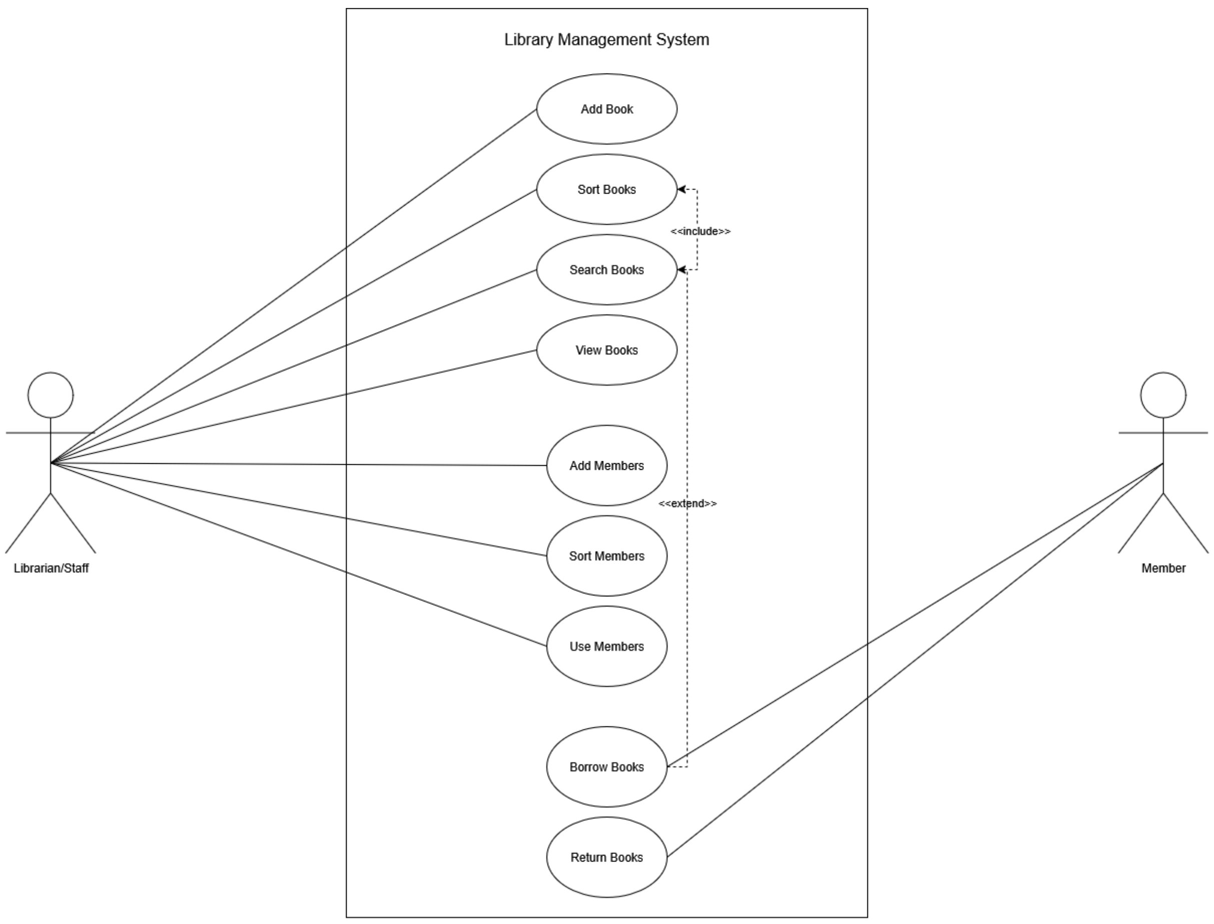
Add Book - Interface for adding new books



```
==============================================================
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
```

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 displayed 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: Sorts the dataset before every query and performs a linear search through the sorted list.
  - Improved: Utilizes an isSorted flag and a cached sorted list using the sorting algorithm to skip redundant sorting, combined with a hybrid binary search that switches to linear search when the range becomes small, improving amortized runtime across repeated queries.
- 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× – 2.2×
Searching (avg of 10 runs)	12,000 – 31,000	5,500 – 11,000	2.0× – 2.9×
Amortized (20 searches)	180,000 – 450,000	140,000 – 260,000	1.8× – 2.3×

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× and 3× 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

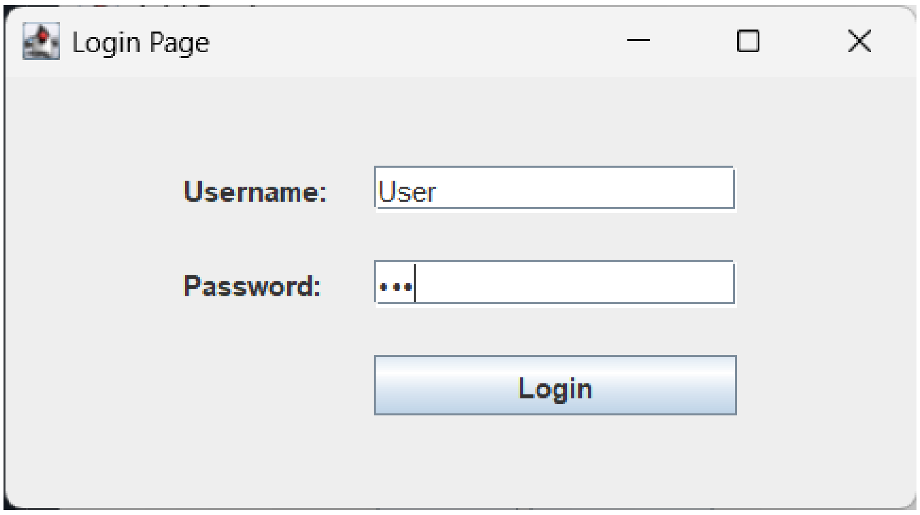
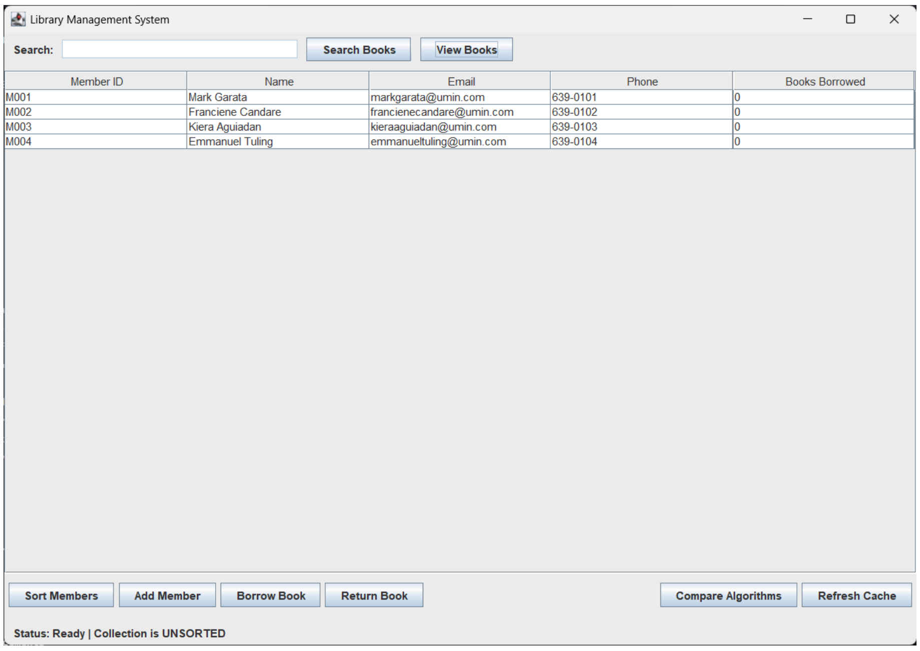
Goetz, B. (2006). *Java Concurrency in Practice*. Addison-Wesley Professional.

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	
View Members	

Login Page

Username: User

Password: •••

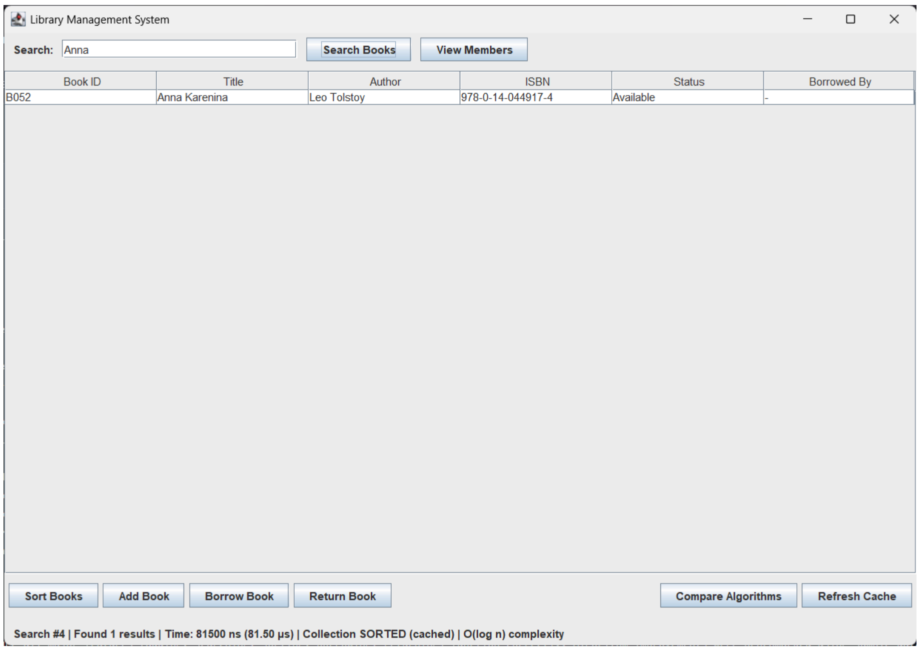
Library Management System

Search:

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

Status: Ready | Collection is UNSORTED

Search Book



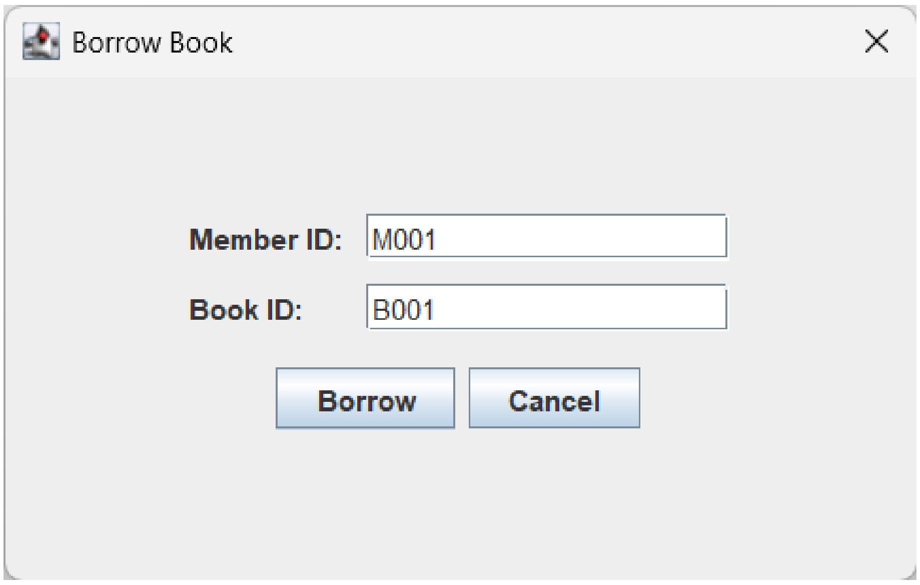
Library Management System

Search: Anna

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 μs) | Collection SORTED (cached) | O(log n) complexity

Borrow Book

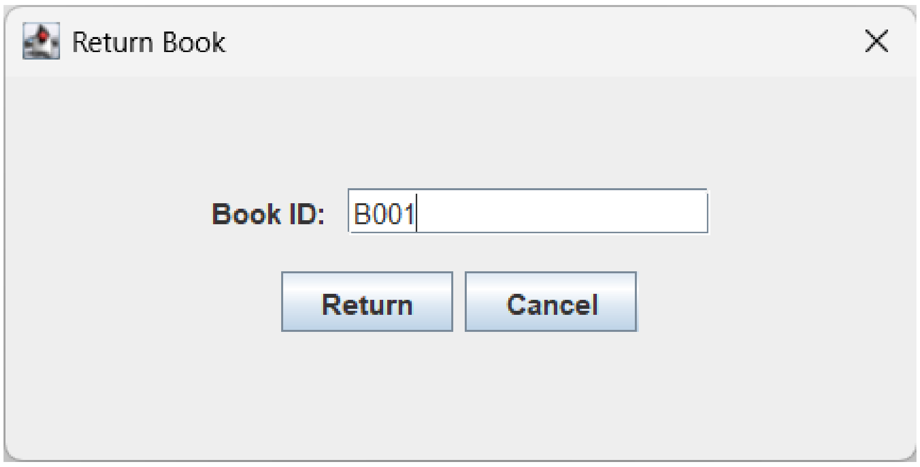


Borrow Book

Member ID: M001

Book ID: B001

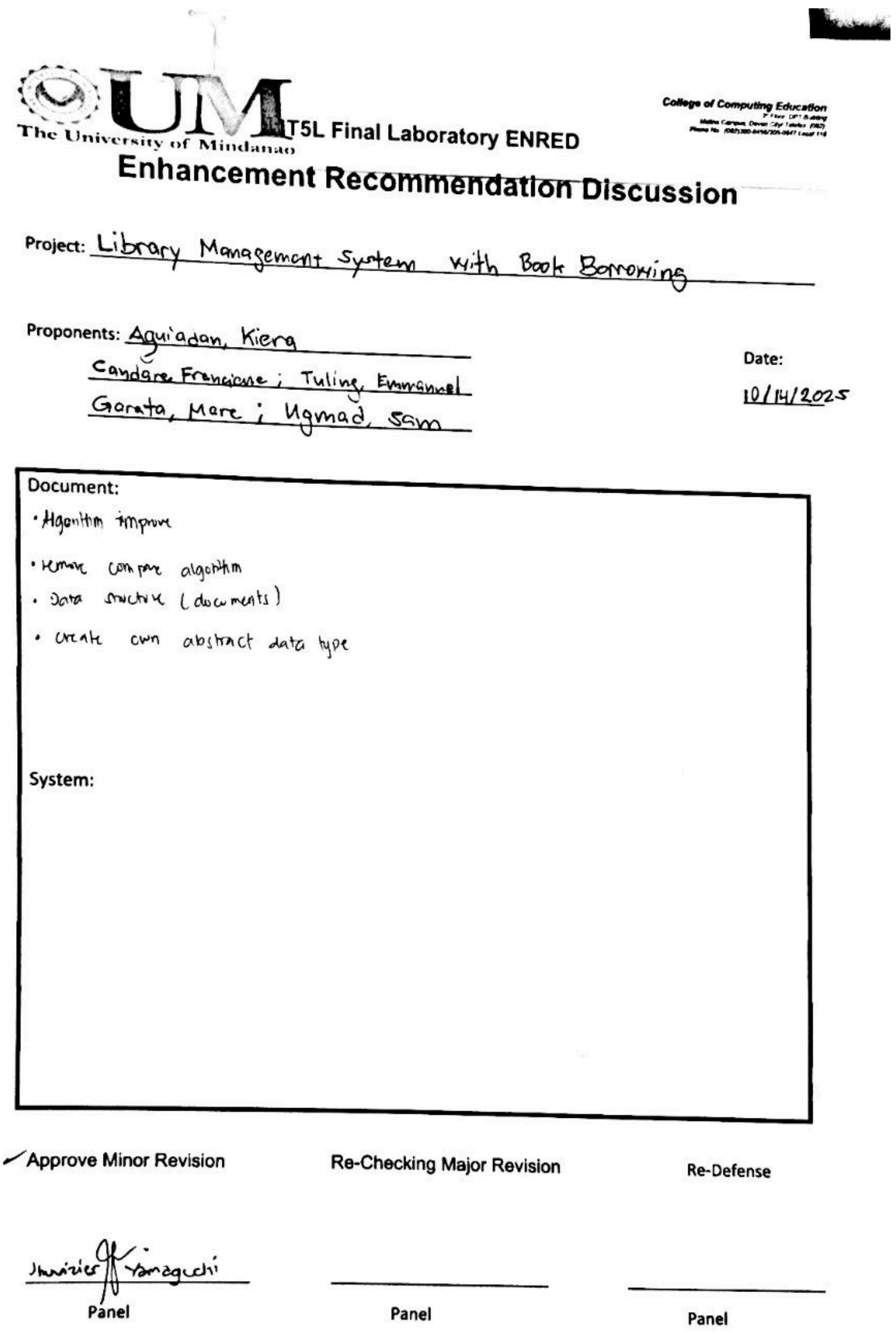
Return Book



Return Book

Book ID: B001

ENRED



The University of Mindanao

College of Computing Education

T5L Final Laboratory ENRED

Enhancement Recommendation Discussion

Project: Library Management System with Book Borrowing

Proponents: Aguiadan, Kiera
Candare Franciane ; Tuling, Emmanuel
Garata, Marc ; Ugmad, Sam

Date: 10/14/2025

Document:

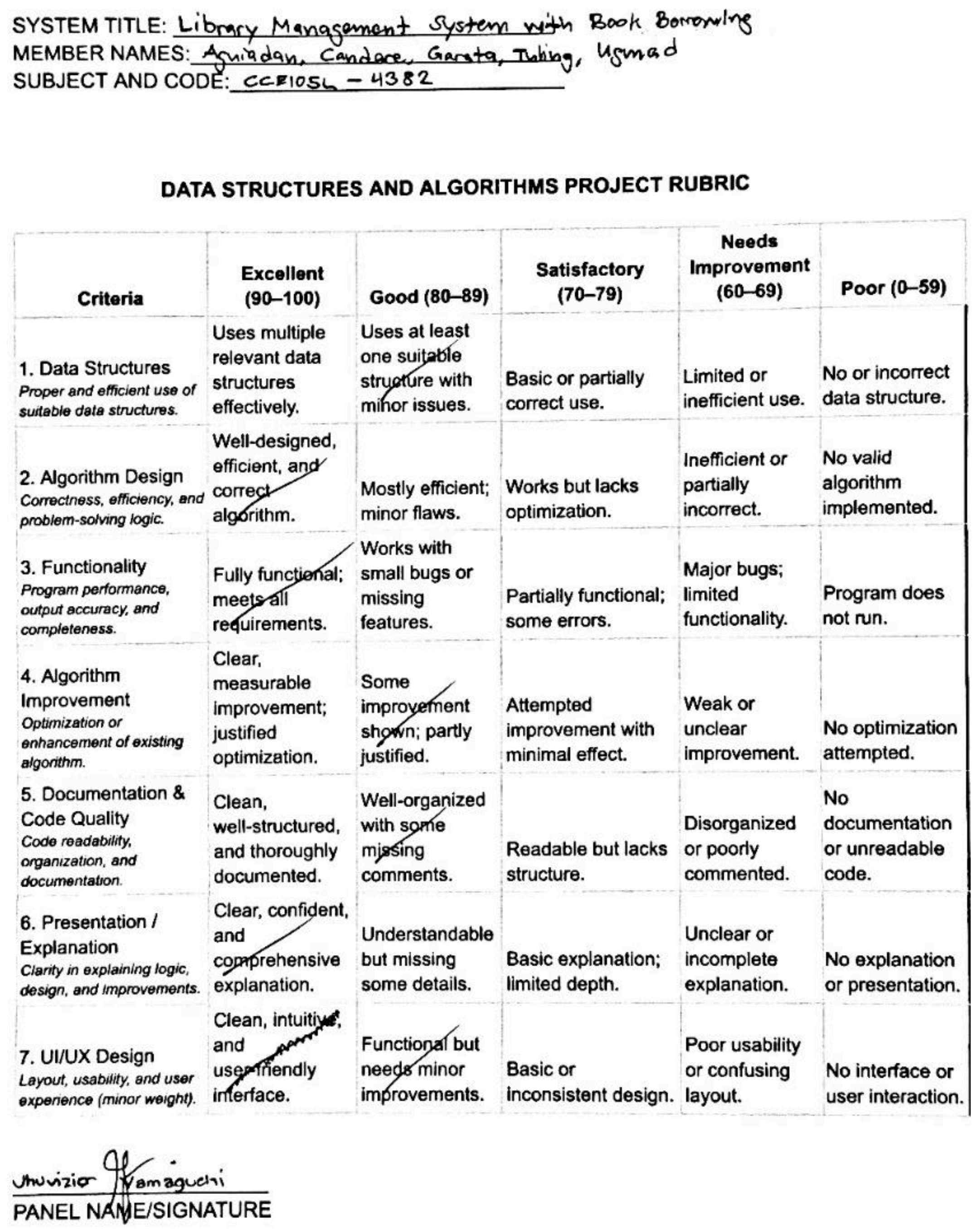
- Algorithm improve
- Remove compare algorithm
- Data structure (documents)
- create own abstract data type

System:

[✓ Approve Minor Revision] Re-Checking Major Revision Re-Defense

[illegible]		
Panel	Panel	Panel

RUBRICS



SYSTEM TITLE: Library Management System with Book Borrowing
MEMBER NAMES: Aguiadan, Candare, Garata, Tubing, Usmad
SUBJECT AND CODE: CCE105L – 4382

DATA STRUCTURES AND ALGORITHMS PROJECT RUBRIC

Criteria	**Excellent (90–100)**	**Good (80–89)**	**Satisfactory (70–79)**	**Needs Improvement (60–69)**	**Poor (0–59)**
1. Data Structures *Proper and efficient use of suitable data structures.*	Uses multiple relevant data structures effectively.	[✓ Uses at least one suitable structure with minor issues.]	Basic or partially correct use.	Limited or inefficient use.	No or incorrect data structure.
2. Algorithm Design *Correctness, efficiency, and problem-solving logic.*	[✓ Well-designed, efficient, and correct algorithm.]	Mostly efficient; minor flaws.	Works but lacks optimization.	Inefficient or partially incorrect.	No valid algorithm implemented.
3. Functionality *Program performance, output accuracy, and completeness.*	[✓ Fully functional; meets all requirements.]	Works with small bugs or missing features.	Partially functional; some errors.	Major bugs; limited functionality.	Program does not run.
4. Algorithm Improvement *Optimization or enhancement of existing algorithm.*	Clear, measurable improvement; justified optimization.	[✓ Some improvement shown; partly justified.]	Attempted improvement with minimal effect.	Weak or unclear improvement.	No optimization attempted.
5. Documentation & Code Quality *Code readability, organization, and documentation.*	Clean, well-structured, and thoroughly documented.	[✓ Well-organized with some missing comments.]	Readable but lacks structure.	Disorganized or poorly commented.	No documentation or unreadable code.
6. Presentation / Explanation *Clarity in explaining logic, design, and improvements.*	[✓ Clear, confident, and comprehensive explanation.]	Understandable but missing some details.	Basic explanation; limited depth.	Unclear or incomplete explanation.	No explanation or presentation.
7. UI/UX Design *Layout, usability, and user experience (minor weight).*	[✓ Clean, intuitive, and user-friendly interface.]	[✓ Functional but needs minor improvements.]	Basic or inconsistent design.	Poor usability or confusing layout.	No interface or user interaction.

Jhuvizio Yamaguchi
PANEL NAME/SIGNATURE