# LIBRARY MANAGEMENT SYSTEM

Muhammad Hamza (22k-4523)
Abdul Wasey (22k-4172)
Saad Arshad (22k-4141)

08-Dec-23

—

Data Structures And Algorithms

—

Submitted To:
**Sir Basit Jasani**

# INTRODUCTION

The Library Management System aims to provide an efficient and user-friendly platform for managing library resources. This program utilizes C++ programming language and includes key features such as book addition, removal, user management, book checkout, and notification center. The programs is designed to entertain both librarians and students, with different functionalities each.

# Used Data Structure Concepts

## Binary Search Tree

BST is Used for efficient searching and retrieval of books based on their unique identifiers (IDs).
The BST is employed to organize books in a hierarchical structure where each node holds a book's information. The left subtree contains books with lower IDs, while the right subtree contains those with higher IDs.

## Stack

Stack maintains a history log of checked-out books and administrative actions.
It allowed the easy retrieval of the most recent actions.

## QUEUE

Queue manages the queue of patrons waiting for a specific book.
It is utilized to handle book requests in a first come, first served manner.

## Vectors

Stores collection of books, patrons, and checked-out books.

---

## Unordered Map

Manages the availability status of the books, keeps track of the patrons waiting for a particular book, and tracks checked-out books for each patron.

---

# Efficiency of Project

## Time Complexity Analysis:

The ‘’Library” class employs various data structures and operations with distinct time complexities.

Adding a book with Library:addBook() is constant time (O(1)) as it involves updating the catalog and books.txt file independently of existing data.

In contrast, removing a book with Library::removeBook(int bookId) is linear (O(n)) due to the search operation in the catalog.

Displaying the catalog (Library::displayCatalogue()) is also linear (O(n)) as it reads data from the books.txt file.

Requesting a book (Library::requestBook(int bookId, const Patron& patron)) and checking it out (Library::checkoutBook(int bookId, const Patron& patron)) have constant (O(1)) and linear (O(n)) complexities, respectively.

Notifying users (Library::notifyUsers()) depends on the sizes of bookRequests and checkoutHistory, resulting in O(n + m).

The overall checkout process (Library::checkoutProcess(const Patron& patron)) is O(n), dominated by the displayCatalogue() function. Finally, displaying admin history (Library::displayHistory()) has a linear time complexity (O(m)) proportional to the adminHistory stack's size.

## Space Complexity Considerations:

The Library class exhibits diverse space complexities across its operations.

Adding a book (Library::addBook()) incurs constant space (O(1)) as it involves minimal data creation.

Removing a book (Library::removeBook(int bookId)) has a linear space complexity (O(n)) primarily due to the storage requirements of the catalog vector.

Displaying the catalog (Library::displayCatalogue()) and requesting a book (Library::requestBook(int bookId, const Patron& patron)) both have constant space complexities (O(1)) since they involve minimal additional storage.

Checking out a book (Library::checkoutBook(int bookId, const Patron& patron)) and the overall checkout process (Library::checkoutProcess(const Patron& patron)) have linear space complexities (O(n)), with the space increasing proportionally with the number of books in the catalog.

Notifying users (Library::notifyUsers()) relies on the bookRequests and checkoutHistory data structures, resulting in space complexity O(n + m), where n is the number of books and m is the number of checked-out books.

Displaying admin history (Library::displayHistory()) has linear space complexity (O(m)), directly related to the size of the adminHistory stack.

## CONCLUSION

In conclusion, the designed library management system demonstrates an effective and organized approach to cataloging and managing books, patrons, and administrative actions. The implementation employs a variety of data structures, including vectors, maps, and stacks, to facilitate essential functionalities such as book addition, removal, and checkout processes. The time complexities of the system's operations have been analyzed, showcasing the efficiency of various functions in handling different tasks. Additionally, the space complexities highlight the system's ability to manage memory usage across diverse operations. The system's overall performance aligns with the design goals of providing a user-friendly interface for both patrons and

administrators, ensuring seamless book management and patron interactions.

## CHANGES MADE AFTER FIRST EVALUATION:

1) When the student successfully performed any action, he was being logged out. So, we changed it and now the program asks the student whether to logout or not.

2) We also added the limited wrong password attempts for the admin.

3) We have added the book searching functionality. Now the student is also able to look for the book through its id. This efficient searching is done by Binary Search Tree which is implemented by the data from the file and linked list.

4) Earlier the student didn't have the option of returning a book. Now the student can do so as we have implemented this functionality too.