



**UNIVERSITI MALAYSIA TERENGGANU**

**FACULTY OF OCEAN ENGINEERING TECHNOLOGY AND  
INFORMATICS**

**CSF3013**

**DATA STRUCTURE AND ALGORITHM**

**Group K3**

**Library Management System**

**Group Members**

<b>Name</b>	<b>Matric No.</b>
Abbas Usman Adamu	S72803
Amir Musyrif bin Mohd Maliki	S71146

## Table of Content

<b>INTRODUCTION.....</b>	<b>3</b>
<b>Key Features and Functionalities:.....</b>	<b>3</b>
<b>ALGORITHM.....</b>	<b>4</b>
<b>Adding a book into the system pseudocode .....</b>	<b>4</b>
<b>Borrowing Book Pseudocode .....</b>	<b>4</b>
<b>Returning Book Pseudocode .....</b>	<b>5</b>
<b>Searching Book Pseudocode.....</b>	<b>5</b>
<b>Book recommendation Pseudocode.....</b>	<b>5</b>
<b>SOURCE CODE .....</b>	<b>6</b>
<b>Discussion of the Source Code .....</b>	<b>17</b>
<b>Output .....</b>	<b>18</b>
<b>Conclusion .....</b>	<b>21</b>
<b>References.....</b>	<b>21</b>

## INTRODUCTION

The Library Management System (LMS) is designed to facilitate efficient library operations by leveraging data structures and algorithms such as Binary Search Trees (BST), sorting algorithms (Quick Sort and Merge Sort), queues, and graph traversal techniques. This system aims to provide features for book management, user interactions, borrowing mechanisms, and personalized book recommendations.

### Key Features and Functionalities:

#### **Book Management:**

Books are stored in a Binary Search Tree (BST), allowing for efficient insertion, deletion, and search operations based on the book title.

Functions include adding books, removing books, and displaying the complete list of books.

#### **Borrowing System:**

A borrowing queue is implemented to handle requests when multiple users want the same book.

Tracks which user has borrowed a book and maintains a waitlist for subsequent users.

Allows users to borrow and return books seamlessly, with automatic notifications for the next user in the queue.

#### **Sorting Mechanisms:**

Quick Sort and Merge Sort algorithms are implemented to sort books based on user-selected criteria (title, author, or ISBN).

### **Search Functionality:**

Users can search for books using their title, author, or ISBN, ensuring quick and accurate results.

### **Recommendation System:**

Utilizes graph traversal techniques to recommend books based on user preferences and category connections.

Tracks user preferences and matches them to relevant categories, providing personalized recommendations.

## **ALGORITHM**

### **Adding a book into the system pseudocode**

Start

1. Prompt the user to enter the title of the book
2. Prompt the user to enter the author of the book
3. Prompt the user to enter the ISBN of the book
4. Create a new Book object using the title, author, and ISBN
5. Add the new Book object to the BookBST

Stop

### **Borrowing Book Pseudocode**

Start

1. Prompt user for name
2. Prompt user for title of book to borrow
3. Search for book in BookBST using title
  - a. If book is found:
    - i. Borrow the book using BorrowQueue

- ii. Display confirmation message

- b. If book is not found:

- i. Display message: "Book not available"

Stop

### **Returning Book Pseudocode**

Start

1. Prompt user for their name and the title of the book they want to return

2. Search for the book in the BorrowQueue (to check if it was borrowed)

- a. If the book is found:

- i. Remove the book from the queue

- ii. Mark the book as returned

- iii. Inform the user that the return was successful

- b. If the book is not found:

- i. Inform the user that the book was not borrowed

Stop

### **Searching Book Pseudocode**

Start

1. Prompt user for the search term (title, author, or ISBN)

2. Prompt user for the search type (title, author, or ISBN)

3. Search the BookBST using the specified search type

- a. If book is found:

- i. Display book details (title, author, ISBN)

- b. If book is not found:

- i. Display message: "Book not found"

Stop

### **Book recommendation Pseudocode**

Start

1. Prompt the user to enter the title of the previously borrowed book.
  2. Prompt the user to choose a recommendation method:
    - 1 for BFS (Breadth-First Search)
    - 2 for DFS (Depth-First Search)
  3. Initialize an empty list to store the recommended books.
  4. If BFS is chosen:
    - Perform BFS starting from the entered book title.
    - For each book visited, add it to the recommended books list.
    - Stop when all connected books have been explored.
  5. If DFS is chosen:
    - Perform DFS starting from the entered book title.
    - For each book visited, add it to the recommended books list.
    - Stop when all connected books have been explored.
  6. Display the list of recommended books.
- Stop

## SOURCE CODE

```
import java.util.*;

public class LibrarySystem {
    // Book class representing a book in the library
    public static class Book {
        String title;
        String author;
        int isbn;

        // Constructor for Book
        public Book(String title, String author, int isbn) {
            this.title = title;
            this.author = author;
            this.isbn = isbn;
        }
        // ToString method to display book details
    }
}
```

```

        @Override
        public String toString() {
            return title + " by " + author + " (ISBN: " + isbn + ")";
        }
    }

    // Node class for the binary search tree (BST)
    public static class BSTNode {
        Book book;
        BSTNode left, right;

        public BSTNode(Book book) {
            this.book = book;
            left = right = null;
        }
    }

    // Binary Search Tree (BST) for storing books
    public static class BookBST {
        private BSTNode root;

        // Method to add a book to the BST
        public void addBook(Book book) {
            root = insert(root, book);
        }

        // Recursive insert method for BST
        private BSTNode insert(BSTNode root, Book book) {
            if (root == null) return new BSTNode(book);
            if (book.title.compareToIgnoreCase(root.book.title) < 0)
root.left = insert(root.left, book);
            else root.right = insert(root.right, book);
            return root;
        }

        // Method to search for a book in the BST by title, author, or ISBN
        public Book searchBook(String query, String type) {
            return search(root, query.toLowerCase(), type);
        }

        // Recursive search method for BST
        private Book search(BSTNode root, String query, String type) {
            if (root == null) return null;
            boolean found = (type.equals("title") &&
root.book.title.equalsIgnoreCase(query)) ||
                            (type.equals("author") &&
root.book.author.equalsIgnoreCase(query)) ||
                            (type.equals("isbn") &&
String.valueOf(root.book.isbn).equals(query));
            if (found) return root.book;
            if (query.compareTo(root.book.title.toLowerCase()) < 0) return
search(root.left, query, type);

```

```

        return search(root.right, query, type);
    }
    // Method to remove a book from the BST by title
    public void removeBook(String title) {
        root = remove(root, title.toLowerCase());
    }

    private BSTNode remove(BSTNode root, String title) {
        if (root == null) return null;
        if (title.compareTo(root.book.title.toLowerCase()) < 0)
            root.left = remove(root.left, title);
        else if (title.compareTo(root.book.title.toLowerCase()) > 0)
            root.right = remove(root.right, title);
        else {
            if (root.left == null) return root.right;
            if (root.right == null) return root.left;
            root.book = findMin(root.right).book;
            root.right = remove(root.right, root.book.title);
        }
        return root;
    }

    private BSTNode findMin(BSTNode root) {
        while (root.left != null) root = root.left;
        return root;
    }

    public List<Book> getAllBooks() {
        List<Book> books = new ArrayList<>();
        inorder(root, books);
        return books;
    }

    private void inorder(BSTNode root, List<Book> list) {
        if (root != null) {
            inorder(root.left, list);
            list.add(root.book);
            inorder(root.right, list);
        }
    }
    // Method to display all books in the BST
    public void displayAllBooks() {
        List<Book> books = getAllBooks();
        if (books.isEmpty()) {
            System.out.println("No books available in the library.");
        } else {
            System.out.println("\nAvailable Books in the Library:");
            for (Book book : books) {

```



```

        System.out.println(book);
    }
}

// Queue class for managing borrowed books and users
public static class BorrowQueue {
    // Map to store who is borrowing the book and the waiting queue for
    each book
    Map<Book, Queue<String>> bookWaitLists = new HashMap<>();
    Map<String, Book> borrowedBooks = new HashMap<>();

    public boolean borrowBook(Book book, String userName) {
        if (book != null) {
            // If the book is already borrowed, add user to the waiting
            list
            if (borrowedBooks.containsKey(book)) {
                bookWaitLists.computeIfAbsent(book, k -> new
                LinkedList<>()).offer(userName);
                // Display the updated waiting list including the
                current user at the bottom
                System.out.println("\nThe following people are waiting
                for the book: " + book.title);
                List<String> waitingList = new
                ArrayList<>(bookWaitLists.get(book));
                for (int i = 0; i < waitingList.size(); i++) {
                    System.out.println((i + 1) + ". " +
                    waitingList.get(i));
                }
                return false;
            } else {
                // If the book is available, borrow it
                borrowedBooks.put(userName, book);
                System.out.println(userName + " borrowed the book: " +
                book.title);
                return true;
            }
        }
        System.out.println("Book not available.");
        return false;
    }

    public void returnBook(Book book, String userName) {
        if (borrowedBooks.containsKey(book) &&
        borrowedBooks.containsKey(userName)) {
            borrowedBooks.remove(userName);

```

```

        System.out.println(userName + " returned the book: " +
book.title);

        // If there are users waiting for this book, let the next
user borrow it
        if (bookWaitLists.containsKey(book) &&
!bookWaitLists.get(book).isEmpty()) {
            String nextUser = bookWaitLists.get(book).poll();
            borrowedBooks.put(nextUser, book);
            System.out.println(nextUser + " is now borrowing the
book: " + book.title);
            // Display the updated waiting list
            System.out.println("\nThe following people are still
waiting for the book: " + book.title);
            List<String> waitingList = new
ArrayList<>(bookWaitLists.get(book));
            for (int i = 0; i < waitingList.size(); i++) {
                System.out.println((i + 1) + ". " +
waitingList.get(i));
            }
        } else {
            System.out.println("This book wasn't borrowed by " +
userName + ".");
        }
    }

    public String nextUserInLine(Book book) {
        if (bookWaitLists.containsKey(book)) {
            return bookWaitLists.get(book).peek(); // Peek to show who
is next in line
        }
        return null;
    }
}

// SortBooks class for sorting books
public static class SortBooks {

    // Quick Sort for sorting books
    public static void quickSort(List<Book> books, int low, int high,
String sortBy) {
        if (low < high) {
            int pi = partition(books, low, high, sortBy);
            quickSort(books, low, pi - 1, sortBy);
            quickSort(books, pi + 1, high, sortBy);
        }
    }
}

```

```
// Partition method for Quick Sort
private static int partition(List<Book> books, int low, int high,
String sortBy) {
    Book pivot = books.get(high);
    int i = low - 1;
    for (int j = low; j < high; j++) {
        boolean shouldSwap = false;
        if (sortBy.equals("title")) {
            shouldSwap =
books.get(j).title.compareToIgnoreCase(pivot.title) < 0;
        } else if (sortBy.equals("author")) {
            shouldSwap =
books.get(j).author.compareToIgnoreCase(pivot.author) < 0;
        } else if (sortBy.equals("isbn")) {
            shouldSwap = books.get(j).isbn < pivot.isbn;
        }
        if (shouldSwap) {
            i++;
            Collections.swap(books, i, j);
        }
    }
    Collections.swap(books, i + 1, high);
    return i + 1;
}

// Merge Sort for sorting books
public static void mergeSort(List<Book> books, int left, int right,
String sortBy) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(books, left, mid, sortBy);
        mergeSort(books, mid + 1, right, sortBy);
        merge(books, left, mid, right, sortBy);
    }
}

private static void merge(List<Book> books, int left, int mid, int
right, String sortBy) {
    List<Book> sorted = new ArrayList<>(books.subList(left, right +
1));

    Collections.sort(sorted, new Comparator<Book>() {
        @Override
        public int compare(Book b1, Book b2) {
            if (sortBy.equals("title")) {
                return b1.title.compareToIgnoreCase(b2.title);
            } else if (sortBy.equals("author")) {
                return b1.author.compareToIgnoreCase(b2.author);
            } else if (sortBy.equals("isbn")) {

```

```

        return Integer.compare(b1.isbn, b2.isbn);
    }
    return 0;
}
});

for (int i = left; i <= right; i++) {
    books.set(i, sorted.get(i - left));
}
}
}

// BookGraph class for book recommendations
public static class BookGraph {
    private Map<String, Set<String>> bookCategoryGraph; // Maps book
titles to connected categories

    public BookGraph() {
        this.bookCategoryGraph = new HashMap<>();
    }

    // Add a relationship between a book and a category
    public void addBookCategory(String bookTitle, String category) {
        bookCategoryGraph.computeIfAbsent(bookTitle, k -> new
HashSet<>()).add(category);
    }

    // Add a relationship between two books (for recommendation
purposes)
    public void addBookConnection(String bookTitle1, String bookTitle2)
{
        bookCategoryGraph.computeIfAbsent(bookTitle1, k -> new
HashSet<>()).add(bookTitle2);
        bookCategoryGraph.computeIfAbsent(bookTitle2, k -> new
HashSet<>()).add(bookTitle1);
    }

    // BFS to recommend books based on a previous book title
    public List<String> recommendBooksBFS(String bookTitle) {
        List<String> recommendations = new ArrayList<>();
        Set<String> visited = new HashSet<>();
        Queue<String> queue = new LinkedList<>();
        queue.offer(bookTitle);
        visited.add(bookTitle);

        while (!queue.isEmpty()) {
            String currentBook = queue.poll();
            for (String neighbor :
bookCategoryGraph.getOrDefault(currentBook, new HashSet<>())) {

```

```

        if (!visited.contains(neighbor)) {
            recommendations.add(neighbor);
            visited.add(neighbor);
            queue.offer(neighbor);
        }
    }
}
return recommendations;
}

// DFS to recommend books based on a previous book title
public List<String> recommendBooksDFS(String bookTitle) {
    List<String> recommendations = new ArrayList<>();
    Set<String> visited = new HashSet<>();
    dfs(bookTitle, visited, recommendations);
    return recommendations;
}

private void dfs(String bookTitle, Set<String> visited, List<String>
recommendations) {
    visited.add(bookTitle);
    for (String neighbor : bookCategoryGraph.getOrDefault(bookTitle,
new HashSet<>())) {
        if (!visited.contains(neighbor)) {
            recommendations.add(neighbor);
            dfs(neighbor, visited, recommendations);
        }
    }
}
}
}
}

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
// Main class for managing the library system
public class LibraryManagementSystem {
    public static void main(String[] args) {
        LibrarySystem.BookBST bst = new LibrarySystem.BookBST();
        LibrarySystem.BorrowQueue queue = new LibrarySystem.BorrowQueue();
        LibrarySystem.SortBooks sorter = new LibrarySystem.SortBooks();
        LibrarySystem.BookGraph bookGraph = new LibrarySystem.BookGraph();
        Scanner scanner = new Scanner(System.in);

        // Adding sample books
        LibrarySystem.Book java = new LibrarySystem.Book("Java Programming",
"John Doe", 101);

```

```

        LibrarySystem.Book python = new LibrarySystem.Book("Python Basics",
"Jane Smith", 102);
        LibrarySystem.Book dataStruct = new LibrarySystem.Book("Data
Structures", "Alice Brown", 103);

        bst.addBook(java);
        bst.addBook(python);
        bst.addBook(dataStruct);

        // Add books and their categories to the graph
        bookGraph.addBookCategory("Java Programming", "Programming");
        bookGraph.addBookCategory("Python Basics", "Programming");
        bookGraph.addBookCategory("Data Structures", "Computer Science");

        // Create connections between books for recommendation purposes
        bookGraph.addBookConnection("Java Programming", "Python Basics");
        bookGraph.addBookConnection("Data Structures", "Java Programming");
        // Display all books
        bst.displayAllBooks();

        while (true) {
            System.out.println("\nChoose an option:");
            System.out.println("1. Add a book");
            System.out.println("2. Remove a book");
            System.out.println("3. Search for a book");
            System.out.println("4. Borrow a book");
            System.out.println("5. Return a book");
            System.out.println("6. Sort books (Merge Sort or Quick Sort)");
            System.out.println("7. Get Book Recommendation");
            System.out.print("Enter choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter title: ");
                    String title = scanner.nextLine();
                    System.out.print("Enter author: ");
                    String author = scanner.nextLine();
                    System.out.print("Enter ISBN: ");
                    int isbn = scanner.nextInt();
                    bst.addBook(new LibrarySystem.Book(title, author,
isbn));

                    break;
                case 2:
                    System.out.print("Enter title to remove: ");
                    String removeTitle = scanner.nextLine();
                    bst.removeBook(removeTitle);

```

```

        break;
    case 3:
        System.out.print("Enter title/author/ISBN to search: ");
        String query = scanner.nextLine();
        System.out.print("Search by (title/author/isbn): ");
        String searchType = scanner.nextLine();
        LibrarySystem.Book foundBook = bst.searchBook(query,
searchType);

        if (foundBook != null) {
            System.out.println("Book found: " + foundBook);
        } else {
            System.out.println("Book not found.");
        }
        break;
    case 4:
        System.out.print("Enter your name: ");
        String userName = scanner.nextLine();
        System.out.print("Enter title to borrow: ");
        String borrowTitle = scanner.nextLine();
        LibrarySystem.Book borrowBook =
bst.searchBook(borrowTitle, "title");
        if (borrowBook != null) {
            queue.borrowBook(borrowBook, userName);
        } else {
            System.out.println("Book not available.");
        }
        break;
    case 5:
        System.out.print("Enter your name: ");
        String returnName = scanner.nextLine();
        System.out.print("Enter title to return: ");
        String returnTitle = scanner.nextLine();
        LibrarySystem.Book returnBook =
bst.searchBook(returnTitle, "title");
        queue.returnBook(returnBook, returnName);
        break;
    case 6:
        System.out.println("Choose sorting method:");
        System.out.println("1. Merge Sort");
        System.out.println("2. Quick Sort");
        int sortChoice = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Sort by (title/author/isbn): ");
        String sortBy = scanner.nextLine();
        List<LibrarySystem.Book> books = bst.getAllBooks();
        if (sortChoice == 1) {
            sorter.mergeSort(books, 0, books.size() - 1,
sortBy);

```

```

    } else if (sortChoice == 2) {
        sorter.quickSort(books, 0, books.size() - 1,
sortBy);
    }
    System.out.println("\nSorted Books:");
    for (LibrarySystem.Book book : books)
System.out.println(book);
    break;
case 7:
    System.out.print("Enter title of the book you previously
borrowed: ");

    String previousBook = scanner.nextLine();
    System.out.println("Choose recommendation method:");
    System.out.println("1. BFS (Breadth-First Search)");
    System.out.println("2. DFS (Depth-First Search)");
    int methodChoice = scanner.nextInt();
    scanner.nextLine();

    List<String> recommendedBooks = new ArrayList<>();
    if (methodChoice == 1) {
        recommendedBooks =
bookGraph.recommendBooksBFS(previousBook);
    } else if (methodChoice == 2) {
        recommendedBooks =
bookGraph.recommendBooksDFS(previousBook);
    }

    if (!recommendedBooks.isEmpty()) {
        System.out.println("Recommended books based on your
previous borrow:");
        for (String bookTitle : recommendedBooks) {
            System.out.println(bookTitle);
        }
    } else {
        System.out.println("No recommendations available.");
    }
    break;
case 8:
    System.out.println("Exiting system.");
    return;
}
}
}}

```



## Discussion of the Source Code

The source code for the Library Management System is structured into several modules, each corresponding to key functionalities:

1. **Book Management Module:** Handles storage, retrieval, and manipulation of book records using a BST. **Insert Book:** Traverse the BST to find the correct position and insert the book node. **Search Book:** Perform a binary search on the tree based on the book's title or ISBN. **Delete Book:** Locate the book node and restructure the tree to maintain BST properties.
2. **Borrowing System Module:** Manages the queue system for borrowing books. **Enqueue:** Add a user to the borrowing queue. **Dequeue:** Remove a user after borrowing a book. **Check Next User:** Retrieve the next user in line.
3. **Sorting Module:** Implements Quick Sort and Merge Sort algorithms for organizing book lists. **Quick Sort:** Partition the book list around a pivot and recursively sort the sub lists. **Merge Sort:** Divide the book list into halves, sort each half, and merge them.
4. **Search Module:** Facilitates searching for books based on title, author, or ISBN.
5. **Recommendation System Module:** Uses graph traversal for personalized book suggestions.

## Efficiency of the Code

**BST Operations:** Logarithmic time complexity for insertion, deletion, and search.

**Sorting:** Quick Sort is optimal for small datasets, while Merge Sort is stable for larger datasets.

**Graph Traversal:** BFS ensures linear time complexity relative to the number of nodes and edges.

## Modularity:

Each functionality is encapsulated in separate modules, enabling easy updates and debugging.

## Scalability:

The system can handle increasing data volumes with minimal performance loss, thanks to efficient algorithms.

### Challenges Addressed:

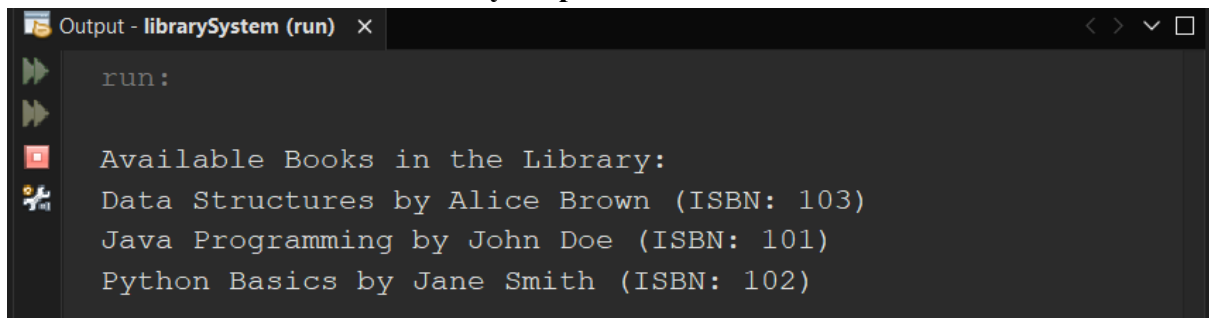
Robust error handling ensures data integrity during critical operations. Efficient recommendation algorithms enhance user experience without overloading system resources.

## Output

Output for Adding book

```
Enter choice: 1
Enter title: usns unfasd fs
Enter author: usman
Enter ISBN: 104
```

### List of Available books in the library output



```
run:
Available Books in the Library:
Data Structures by Alice Brown (ISBN: 103)
Java Programming by John Doe (ISBN: 101)
Python Basics by Jane Smith (ISBN: 102)
```

### Borrowing Book

The system will automatically save the users name and the book he borrowed.

```
Enter choice: 4
Enter your name: usman
Enter title to borrow: Java Programming
usman borrowed the book: Java Programming
```

## Returning Book

The system will save the users name and the previous book he borrowed after that then the user can return the book.

```
Choose an option:
1. Add a book
2. Remove a book
3. Search for a book
4. Borrow a book
5. Return a book
6. Sort books (Merge Sort or Quick Sort)
7. Get Book Recommendation
Enter choice: 5
Enter your name: usman
Enter title to return: Data Structures
usman returned the book: Data Structures
```

if the user did not borrow book and he tries to return the output will be:

```
Choose an option:
1. Add a book
2. Remove a book
3. Search for a book
4. Borrow a book
5. Return a book
6. Sort books (Merge Sort or Quick Sort)
7. Get Book Recommendation
Enter choice: 5
Enter your name: usman
Enter title to return: Data Structures
This book wasn't borrowed by usman.
```

## Recommendation after borrowing a book

```
7. Get Book Recommendation
Enter choice: 7
Enter title of the book you previously borrowed: Data Structures
Choose recommendation method:
1. BFS (Breadth-First Search)
2. DFS (Depth-First Search)
1
Recommended books based on your previous borrow:
Computer Science
Java Programming
Programming
Python Basics
```

## Searching Book

```
Choose an option:
1. Add a book
2. Remove a book
3. Search for a book
4. Borrow a book
5. Return a book
6. Sort books (Merge Sort or Quick Sort)
7. Display all books
8. Get Book Recommendations
9. Exit
Enter choice: 3
Enter title/author/ISBN to search: Alice Brown
Search by (title/author/isbn): author
Book found: Data Structures by Alice Brown (ISBN: 103)
```

## Output for sorting books using merge sort

```
Enter choice: 6
Choose sorting method:
1. Merge Sort
2. Quick Sort
1
Sort by (title/author/isbn): title

Sorted Books:
Data Structures by Alice Brown (ISBN: 103)
Java Programming by John Doe (ISBN: 101)
Python Basics by Jane Smith (ISBN: 102)
```

## Conclusion

The Library Management System (LMS) is designed to streamline and optimize the operations of managing books, borrowing, and recommendations in a library environment. By implementing essential data structures like Binary Search Trees (BST), Linked Lists, and Queues, the system efficiently handles tasks such as book storage, borrowing requests, and sorting. The use of algorithms like Merge Sort and Quick Sort enhances the sorting of books based on various criteria, ensuring quick and accurate retrieval of book information. Additionally, the integration of a book recommendation system through BFS and DFS provides a personalized experience for users, enhancing their engagement with the library's resources.

Overall, the LMS aims to improve the user experience by providing a seamless and efficient platform for managing books, borrowing, and recommendations, while simplifying administrative tasks through automation. The system's flexible design and scalability make it adaptable to libraries of varying sizes and complexity, offering significant improvements over traditional manual systems.

## References

Sharma, S., & Jain, M. (2018). *Design and implementation of a library management system*. International Journal of Advanced Research in Computer Science and Software Engineering, 8(3), 89-93. Retrieved from <http://www.ijarcsse.com>

Kaur, H., & Singh, M. (2017). *Design and development of an automated library management system*. International Journal of Advanced Research in Computer Science, 8(7), 40-45. Retrieved from <https://www.ijarcs.info>

Gupta, N., & Verma, S. (2020). *A survey on library management systems and their implementation*. Journal of Computer Science and Engineering, 11(4), 77-82. <https://doi.org/10.1016/j.jcse.2020.01.003>

Sourabh, S. (2021). *Library management system in Java (source code)*. GitHub. Retrieved from <https://github.com/sourabh07/Library-Management-System>

Agarwal, M., & Kaur, G. (2021). *Library management system project in Java*. GitHub. Retrieved from <https://github.com/ManeetAgarwal/Library-Management-System>