Library Management System

Ghulam Ishaq Khan Institute, Topi

Faculty of Computer Science and Engineering (FCSE)

Computer Engineering (BCE)

CS221: Data Structures and Algorithms

Group Members:

Asad Ali 2022119

Mian Muhammad Arqam 2022294

Taqi Abass 2022444

1.Introduction:

1.1 Synopsis Project:

The Library Management System (LMS) is a comprehensive Data Structures and Algorithms (DSA) project designed to efficiently manage student information, books, and transactions in a library setting. The system incorporates various data structures such as linked lists, binary search trees (BST), arrays, stacks, and queues to provide a versatile and organized solution.

Key Features:

Student Management:

- BST Implementation: Students' information is efficiently managed using a Binary Search Tree (BST), allowing for quick retrieval and insertion based on student ID.
- Linked List: A doubly linked list is used to maintain a sequential list of students, enabling easy traversal and modification.

Book Management:

- Queue Implementation: The project incorporates a queue data structure to manage books in a first-in, first-out (FIFO) manner. This ensures a systematic approach to handling book inventory.

Array Operations:

- Merge Sort: An array is employed to store and manage student IDs. Merge sort is implemented for efficient sorting, facilitating quick search operations.

Stack Operations:

- Book Stack: A stack is utilized to manage a stack of books. This allows for easy addition and removal of books, mimicking a real-world scenario in a library.

User Interface:

- Command-Line Interface (CLI): The project features a simple and user-friendly CLI, allowing users to interact with the system through a series of text-based menu options.
- Add Student: Users can add new students to the system, and the information is simultaneously stored in the BST, linked list, and array.
- Delete Student: Students can be deleted based on their ID, with the operation reflected across all data structures.

- Display Students: The system provides the ability to display student information from both the array and linked list.
- Add Book: Users can add books to the library inventory, and the information is stored in a queue.

Error Handling:

- Array Full Warning: Users are notified when the array reaches its maximum capacity, preventing further additions until space is available.
- Queue Full Warning: A warning is displayed when the book queue reaches its maximum capacity, ensuring that users are aware of the limitation.

BST Display:

- In-order Traversal: The Binary Search Tree is displayed using in-order traversal, showcasing the hierarchical structure of student information.

Dynamic Memory Allocation:

- Node Creation: Dynamic memory allocation is utilized for creating nodes in the linked list, allowing for efficient memory usage.

Modular Design:

- Separate Classes: Different classes are created for each data structure and functionality, promoting a modular and organized code structure.
- Library Management System Class: The main class encapsulates the overall functionality of the library management system, ensuring a clear separation of concerns.

Continuous Execution:

- Infinite Loop: The program runs in an infinite loop, allowing users to perform multiple operations without restarting the application.

Educational significance:

The presented library management system project holds significant educational value by providing a hands-on application of fundamental concepts in data structures and algorithms. Through the integration of arrays, linked lists, stacks, queues, and binary search trees, students gain practical insights into designing and implementing complex systems. The project's educational significance lies in its ability to bridge theoretical knowledge with real-world scenarios, enabling students to understand the intricacies of data manipulation, algorithm optimization, and system design. By simulating a library environment, students learn the importance of selecting appropriate data structures for specific tasks, managing dynamic data, and implementing efficient algorithms for tasks such as sorting and searching. Additionally, the

project enhances students' problem-solving skills, analytical thinking, and software development capabilities, preparing them for future challenges in the field of computer science. Overall, this project serves as a valuable educational tool, offering a holistic learning experience that reinforces theoretical concepts while fostering practical skills essential for software development.

1.2 Objectives:

The objective of this code is to implement a comprehensive Library Management System that incorporates various data structures and algorithms. The system allows users to perform essential operations related to student management and book transactions. Key objectives include:

Student Management:

- Utilize a Binary Search Tree (BST) to efficiently organize and manage student records.
- Implement a linked list to maintain a dynamic list of students, facilitating easy addition and deletion.

Array Operations:

- Demonstrate the use of an array to store and manage student information.
- Implement a merge sort algorithm for array sorting.

Book Transaction System:

- Integrate a queue to handle book transactions, allowing users to add and remove books from the library.
- Utilize a stack to keep track of books and their details.

User Interaction:

- Provide a user-friendly interface through a console-based menu system.
- Enable users to add students, delete students by ID, display student information, add books, and exit the system.

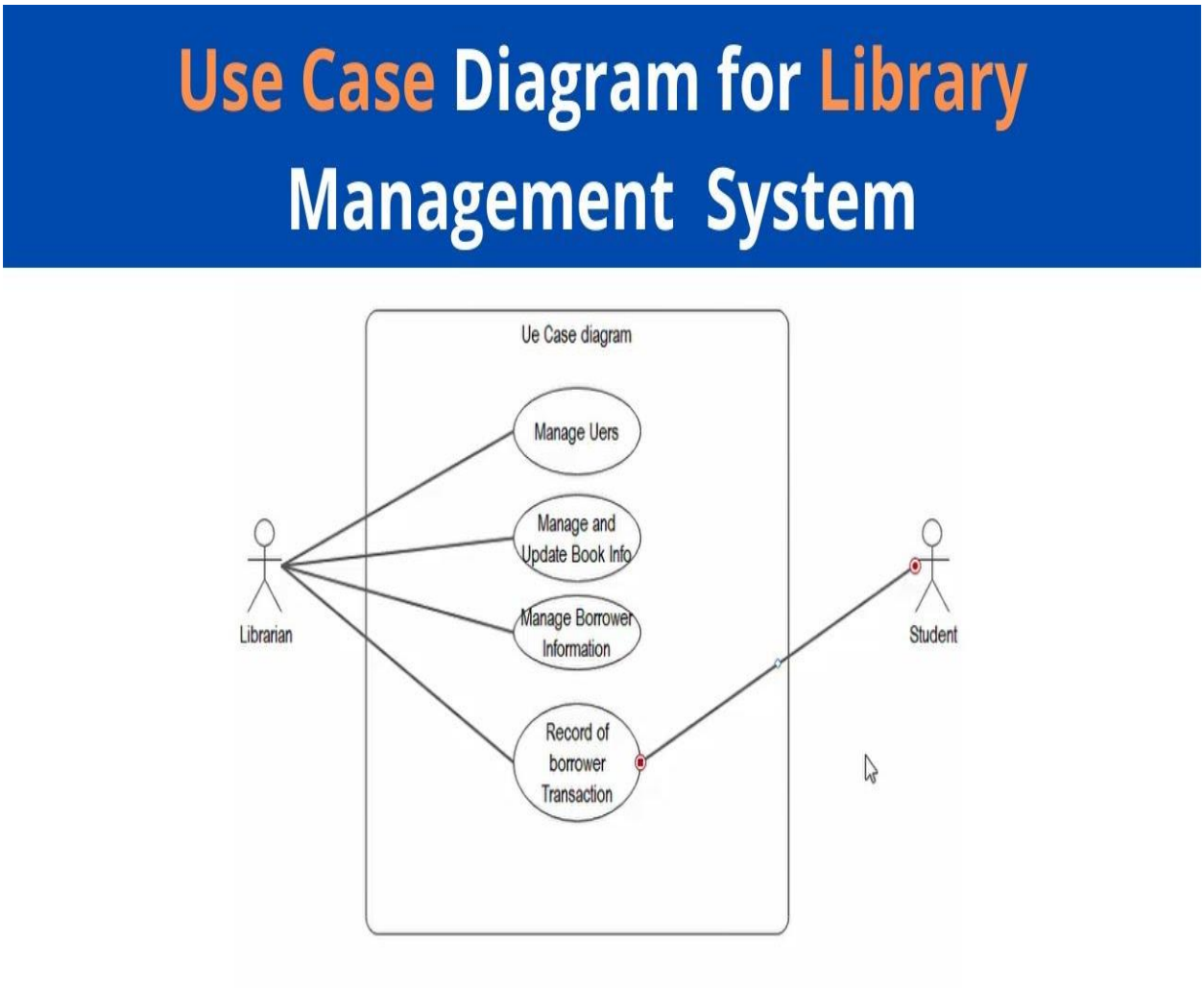
Educational Significance:

- Serve as an educational tool for students learning data structures and algorithms.
- Illustrate the practical application of arrays, linked lists, stacks, queues, and binary search trees in a real-world scenario.
- Enhance problem-solving skills, algorithmic thinking, and software development capabilities.
- By achieving these objectives, the code aims to create a functional and educational Library Management System, offering a hands-on experience in applying theoretical concepts to solve practical problems in computer science.

2. System Analysis and Design:

2.1 Use case diagram:

Leveraging advanced data structures is paramount in the dynamic realm of library management, aiming to enhance operational efficiency and elevate the overall user experience. The project's use case diagram serves as a pivotal visual representation, illustrating diverse interactions within the Library Management System. As a crucial element in system design, this diagram delineates key participants, including managers and students, elucidating their engagement with vital features such as book checkouts, returns, student data management, and the seamless integration of a binary search tree. Through this graphical representation, users gain insights into the myriad scenarios and functionalities encapsulated by the system, offering a blueprint for the development of a user-friendly and dependable library management solution.



2.1.2 Detail Description for each Use Case

Use Case	Description
Add Student	To enroll a new student in the library system, the Administrator must adhere to the steps outlined in the "Add Student" use case. This involves collecting vital information such as the student's age, name, and unique student ID.
Sort Students	Utilizing sorting algorithms within the "Sort Students" use case, the Administrator organizes the system's student list, ensuring a predetermined or alphabetical order for efficient data retrieval.
Search Students	For locating a student within the list, the Administrator employs the "Search Student" use case, enabling searches based on the student's ID, name, or age.
Check Out Book	The process by which a student borrows a book from the library is encapsulated in the "Check Out Book" use case. Upon submitting the requisite data, the system updates the inventory, verifying the book's availability.
Return Book	In the "Return Book" use case, a student returns a borrowed book to the library. The system updates the book's status, calculates any associated fines or penalties, and maintains a record of the transaction for future reference.
Manage Books	The "Manage Books" use case outlines the internal processes for handling book inventory, encompassing stock level tracking, status updates for books, and ensuring the accuracy of the library's collection.
Record Books	The "Record Books" use case illustrates the system's internal procedure for documenting each book transaction. This includes details about the involved student, the book's title, the

	type of transaction (return or checkout), and the timestamp.

3. Used Data Structures:

3.1 Arrays and Linked Lists:

Arrays: The ArrayList class uses arrays to store a fixed-size list of students. Using indices, arrays enable constant-time access to elements. Nevertheless, dynamic resizing is difficult because of their fixed size.

- **Advantages:** O (1) random access, simplicity.
- **Justification:** Ideal for scenarios where the number of elements is fixed, providing constant-time access to elements.

Linked Lists: Each element (node) of a linked list, which is represented by the LinkedList class, is a dynamic data structure that holds a reference to the node after it as well as a data element. They are appropriate for situations where the number of elements can fluctuate dynamically because they enable effective insertions and deletions at any point in the list.

- **Advantages:** Dynamic size, efficient insertions, and deletions.
- **Justification:** Well-suited for dynamic scenarios where elements can be added or removed without the need for contiguous memory.

3.2 Stack and Queue:

Stack: A stack data structure that adheres to the Last In, First Out (LIFO) principle is used by the Stack class. The elements at the top of the stack are changed or added. Tracking book checkouts with this structure—where the last book checked out is the first to be returned—is effective.

- **Advantages:** Last In, First Out (LIFO) structure, efficient for tracking operations.
- **Justification:** Mimics scenarios where the last operation is the first to be processed.

Queue: A queue data structure, which adheres to the First In, First Out (FIFO) principle, is used by the Queue class. Aspects are taken out of the front and added to the back. This organizational structure works well for processing book return requests in the order that they are received.

- **Advantages:** First In, First Out (FIFO) structure, suitable for managing operations in order.
- **Justification:** Models scenarios where operations are processed in the order they are received.

Binary Search Tree: A binary search tree is a hierarchical data structure with a maximum of two children for each node in the tree, implemented by the BinarySearchTree class. Because the nodes are arranged according to the values of their keys, insertion, deletion, and searching can be done quickly and effectively. Because of its logarithmic time complexity in common operations, this structure was selected.

- **Advantages:** Efficient searching, insertion, and deletion operations with logarithmic time complexity.
- **Justification:** Ideal for scenarios where quick and efficient retrieval of data is crucial, enhancing overall system functionality.

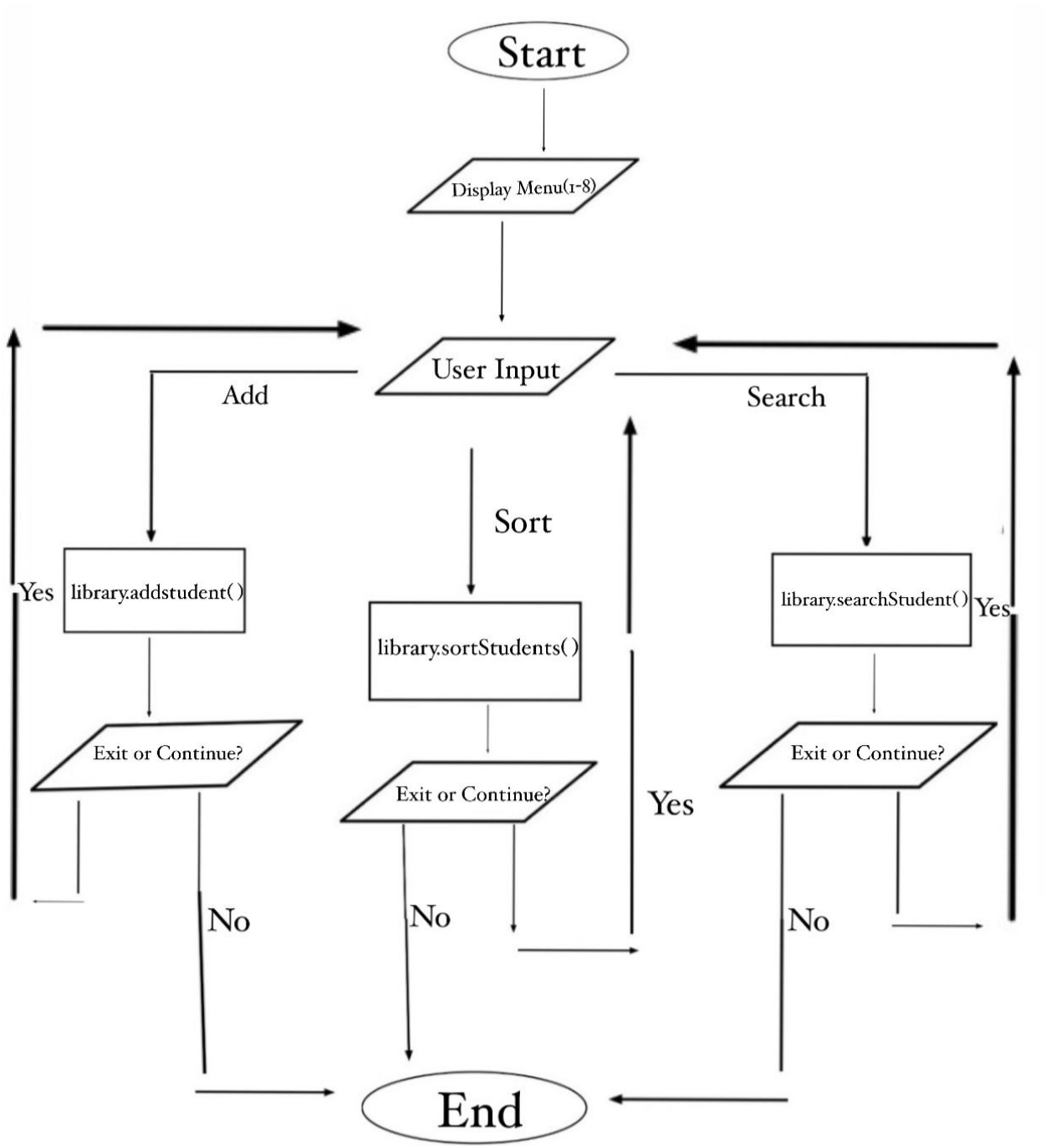
Summery:

- The provided C++ code implements a library management system with a focus on utilizing various data structures. It includes classes for students, linked lists, binary search trees (BST), arrays, books, stacks, and queues. The main functionalities of the system involve adding students to the system using a BST, a linked list, and an array simultaneously. The system also allows for the addition of books to a queue.
- The code operates within a menu-driven interface where the user can choose to add students, delete students by ID, display the list of students using both an array and a linked list, add books to a queue, and exit the program.
- The use of multiple data structures showcases a comprehensive approach to managing student and book data efficiently. The implementation integrates fundamental data structures like linked lists and arrays for student data, a binary search tree for efficient searching, and queues for book management. The stack is present but not extensively used in the provided code.
- Overall, the code provides a foundation for a library management system that leverages diverse data structures to enhance performance and organization.

Considerations:

- Time Complexity: For routine operations, the chosen data structures give priority to efficient time complexities.
- Space Efficiency: Considering the memory requirements of the system, data structures are selected to balance space efficiency.
- Data Nature: The design considers the dynamic nature of data, with varying numbers of elements and frequent operations.

4. System Design and Workflow:



Conclusion:

The provided C++ code implements a library management system with diverse data structures, including linked lists, binary search trees (BSTs), arrays, stacks, and queues. The system facilitates the addition and deletion of students using a BST, linked list, and array simultaneously, along with book management through a queue.

In summary, the code demonstrates the integration of different data structures to create a versatile library management system. The menu-driven interface enables operations such as adding and deleting students, displaying student information using arrays and linked lists, and adding books to a queue. Overall, the code exemplifies the synergy of various data structures for an efficient library management solution.