**Exercise 6: Library Management System**

**1. Understand Search Algorithms**

**Linear Search:**

**Definition: Linear search is a simple search algorithm that checks each element in the list sequentially until the desired element is found or the list ends.**

**Complexity: The time complexity of linear search is O(n) in the worst case, where n is the number of elements in the list.**

**Binary Search:**

**Definition: Binary search is an efficient search algorithm that finds the position of a target value within a sorted array. It repeatedly divides the search interval in half.**

**Complexity: The time complexity of binary search is O(log n) in the worst case, where n is the number of elements in the list.**

**2. Setup**

**// Book.java**

**public class Book {**

**private String bookId;**

**private String title;**

**private String author;**

**public Book(String bookId, String title, String author) {**

**this.bookId = bookId;**

**this.title = title;**

**this.author = author;**

**}**

**// Getters and Setters**

**public String getBookId() {**

**return bookId;**

**}**

**public void setBookId(String bookId) {**

**this.bookId = bookId;**

**}**

**public String getTitle() {**

**return title;**

**}**

**public void setTitle(String title) {**

**this.title = title;**

**}**

**public String getAuthor() {**

**return author;**

**}**

**public void setAuthor(String author) {**

**this.author = author;**

**}**

**@Override**

**public String toString() {**

**return "Book{" +**

**"bookId='" + bookId + '\'' +**

**", title='" + title + '\'' +**

**", author='" + author + '\'' +**

**'}';**

**}**

**}**

**3. Implementation**

**import java.util.Collections;**

**import java.util.Comparator;**

**public class LibraryManagementSystem {**

**private List<Book> books;**

**public LibraryManagementSystem() {**

**this.books = new ArrayList<>();**

**}**

**// Add a book**

**public void addBook(Book book) {**

**books.add(book);**

**}**

**// Linear search to find a book by title**

**public Book linearSearchByTitle(String title) {**

**for (Book book : books) {**

**if (book.getTitle().equalsIgnoreCase(title)) {**

**return book;**

**}**

**}**

**return null; // Book not found**

**}**

**// Binary search to find a book by title (assuming the list is sorted)**

**public Book binarySearchByTitle(String title) {**

**// Ensure the list is sorted by title**

**Collections.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));**

**int left = 0;**

**int right = books.size() - 1;**

**while (left <= right) {**

**int mid = left + (right - left) / 2;**

**Book midBook = books.get(mid);**

**int comparison = midBook.getTitle().compareToIgnoreCase(title);**

**if (comparison == 0) {**

**return midBook;**

**} else if (comparison < 0) {**

**left = mid + 1;**

**} else {**

**right = mid - 1;**

**}**

**}**

**return null; // Book not found**

**}**

**}**

**4. Analysis**

**Time Complexity:**

* **Linear Search:**

**Worst Case: O(n) - The search goes through all elements.**

**Best Case: O(1) - The search finds the element at the first position.**

**Average Case: O(n) - Generally, it involves scanning half of the list.**

* **Binary Search:**

**Worst Case: O(log n) - The search interval is halved with each step.**

**Best Case: O(1) - The search finds the element at the middle position initially.**

**Average Case: O(log n) - Typically involves log(n) steps.**

**When to Use Each Algorithm:**

* **Linear Search:**

**Small Data Sets: When the dataset is small, the overhead of sorting for binary search may not be justified.**

**Unsorted Data: When the data is not sorted and sorting is not feasible due to time constraints or overhead.**

* **Binary Search:**

**Large Data Sets: More efficient for larger datasets due to logarithmic time complexity.**

**Sorted Data: Requires the data to be sorted, which adds a preprocessing step but results in much faster searches.**