Exercise 6: Library Management System

Scenario:

You are developing a library management system where users can search for books by title or author.

Steps:

1. **Understand Search Algorithms:**
    - Explain linear search and binary search algorithms.
2. **Setup:**
    - Create a class **Book** with attributes like **bookId**, **title**, and **author**.
3. **Implementation:**
    - Implement linear search to find books by title.
    - Implement binary search to find books by title (assuming the list is sorted).
4. **Analysis:**
    - Compare the time complexity of linear and binary search.
    - Discuss when to use each algorithm based on the data set size and order.

→ 1. Understand Search Algorithms

Linear Search

- Scans each element one by one.
- No need for sorted data.
- ✅ Best for **unsorted** or **small** data sets.
- **Time Complexity**:
    - Best Case: O(1)
    - Average/Worst Case: O(n)

Binary Search

- Works only on **sorted** data.
- Divides search space in half at every step.
- ✅ Best for **large**, **sorted** data sets.
- **Time Complexity**:
    - Best Case: O(1)
    - Average/Worst Case: O(log n)

2. Setup: Book Class

```
public class Book {
    int bookId;
    String title;
    String author;
    public Book(int bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }
    @Override
    public String toString() {
        return "Book[ID=" + bookId + ", Title=" + title + ", Author=" + author + "]";
    }
}
```

3. Implementation

- **Linear Search**

```
public class BookSearch {
    // Linear Search by title
    public static Book linearSearch(Book[] books, String title) {
        for (Book book : books) {
            if (book.title.equalsIgnoreCase(title)) {
                return book;
            }
        }
        return null;
    }
```

- **Binary Search (List must be sorted by title)**

```
    // Binary Search by title
    public static Book binarySearch(Book[] books, String title) {
        int low = 0;
        int high = books.length - 1;
```

```
        while (low <= high) {
            int mid = (low + high) / 2;
            int compare = books[mid].title.compareToIgnoreCase(title);
            if (compare == 0) {
                return books[mid];
            } else if (compare < 0) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return null;
    }
```

Example Main Method

```
import java.util.Arrays;
import java.util.Comparator;
public class Main {
    public static void main(String[] args) {
        Book[] books = {
            new Book(101, "Java Programming", "James Gosling"),
            new Book(102, "Data Structures", "Robert Lafore"),
            new Book(103, "Algorithms", "CLRS"),
            new Book(104, "Design Patterns", "GoF")
        };
        // Sort books for binary search
        Arrays.sort(books, Comparator.comparing(b -> b.title.toLowerCase()));
        // Linear Search
        System.out.println("Linear Search for 'Data Structures':");
        Book result1 = BookSearch.linearSearch(books, "Data Structures");
        System.out.println(result1 != null ? result1 : "Not Found");
```

```
        // Binary Search
        System.out.println("\nBinary Search for 'Algorithms':");
        Book result2 = BookSearch.binarySearch(books, "Algorithms");
        System.out.println(result2 != null ? result2 : "Not Found");
    }
}
```

4. Analysis

- **Time Complexity**

Algorithm	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)

- **When to Use Which?**

Use Case	Recommended Search
Unsorted or small dataset	Linear Search
Sorted and large dataset	Binary Search
Data updated frequently (unsorted)	Linear Search
Static sorted catalog	Binary Search

OUTPUT: