**Exercise 6: Library Management System**

**Scenario:**

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

1. Understand Search Algorithms:

* Explain linear search and binary search algorithms.
* **Linear Search:** **Linear search** is a method for searching for an element in a collection of elements. In linear search, each element of the collection is visited one by one in a sequential fashion to find the desired element. Linear search is also known as sequential search.
* **Algorithm:**
* ***Start:****Begin at the first element of the collection of elements.*
* ***Compare:****Compare the current element with the desired element.*
* ***Found:****If the current element is equal to the desired element, return true or index to the current element.*
* ***Move:****Otherwise, move to the next element in the collection.*
* ***Repeat:****Repeat steps 2-4 until we have reached the end of collection.*
* ***Not found:****If the end of the collection is reached without finding the desired element, return that the desired element is not in the array.*
* **Binary Search:** Binary search is a search algorithm used to find the position of a target value within a sorted array. It works by repeatedly dividing the search interval in half until the target value is found or the interval is empty. The search interval is halved by comparing the target element with the middle value of the search space.

To apply Binary Search algorithm:

* The data structure must be sorted.
* Access to any element of the data structure should take constant time.
* **Algorithm:**
* *Divide the search space into two halves by finding the middle index “mid”.*
* *Compare the middle element of the search space with the****key****.*
* *If the****key****is found at middle element, the process is terminated.*
* *If the****key****is not found at middle element, choose which half will be used as the next search space.*
* *If the****key****is smaller than the middle element, then the****left****side is used for next search.*
* *If the****key****is larger than the middle element, then the****right****side is used for next search.*
* *This process is continued until the****key****is found or the total search space is exhausted.*

2. Setup:

* **Class Creation**: Create a class **Book** with attributes like **bookId**, **title**, and **author**

class Book implements Comparable<Book> {

    int bookId;

    String title;

    String author;

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

    this.bookId = bookId;

    this.title = title;

    this.author = author;

    }

**class Book implements Comparable<Book>**

* **class Book**: This defines a new class named Book.
* **implements Comparable<Book>**: This indicates that the Book class implements the Comparable interface for Book objects. The Comparable interface requires the class to define a compareTo method, which will be used to compare Book objects, typically for sorting purposes.

**Instance variables (or attributes) of the Book class:**

* **int bookId**: An integer representing the unique identifier for each book.
* **String title**: A string representing the title of the book.
* **String author**: A string representing the author of the book.

**Constructor:**

* **public Book(int bookId, String title, String author)**: The constructor has three parameters: bookId (an integer), title (a string), and author (a string). These parameters are used to initialize the instance variables.
* **this.bookId = bookId;**: The this keyword is used to refer to the current instance of the class. This line assigns the value of the parameter bookId to the instance variable bookId.
* **this.title = title;**: Similarly, this line assigns the value of the parameter title to the instance variable title.
* **this.author = author;**: This line assigns the value of the parameter author to the instance variable author.

3. Implementation:

* Implement linear search to find books by title:

  private static Book linearSearch(List<Book> books, String title) {

        for (Book book : books) {

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

                return book;

            }

        }

        return null;

    }

* Implement binary search to find books by title:

    private static Book binarySearch(List<Book> books, String title) {

        int left = 0;

        int right = books.size() - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            int comparison = books.get(mid).title.compareToIgnoreCase(title);

            if (comparison == 0) {

                return books.get(mid);

            } else if (comparison < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

Here is the github repo link –

4. Analysis:

* Compare the time complexity of linear and binary search

**Time Complexity** of Linear Search**:**

* **Iteration over the List:**
* The loop iterates over each element in the list books.
* If there are n books in the list, the loop will run n times in the worst case (when the book is not found or is the last book in the list).
* **Total Comparisons:**
* The equalsIgnoreCase method compares two strings character by character. In the worst case, this comparison takes O(m) time, where m is the length of the strings being compared.
* However, since m is usually small compared to n (and often considered constant), we focus on the iteration over the list.
* **Time Complexity:**
* **Worst-case Complexity:** O(n \* m) - if the book is found at the end of the list or not found at all, where n is the number of books and m is the length of the title strings.
* **Best-case Complexity:** O(1) - if the book is found at the beginning of the list.

For typical usage where string length is considered constant, the time complexity is usually described as O(n).

**Time Complexity** of Binary Search**:**

 **Binary Search Loop:**

* The binary search algorithm repeatedly divides the search range in half.
* Each iteration cuts the search space in half, leading to a logarithmic number of iterations.
* The number of iterations is proportional to log2(n), where n is the number of books in the list.

 **Comparison:**

* The compareToIgnoreCase method compares two strings character by character. In the worst case, this comparison takes O(m) time, where m is the length of the strings being compared.
* However, since m is usually small compared to n (and often considered constant), we focus on the binary search iterations.
* **Time Complexity:**
* **Worst-case Complexity:** O(log n \* m) - where n is the number of books and m is the length of the title strings.
* **Best-case Complexity:** O(1) - if the book is found at the first midpoint.

In practical terms, if the length of the title strings is relatively small and constant, the time complexity can be simplified to O(log n), where n is the number of books.

* Discuss when to use each algorithm based on the data set size and order.
* **Data Set Size:**
* **Small Data Sets:** Linear search is often sufficient due to its simplicity.
* **Large Data Sets:** Binary search is preferred for its efficiency.
* **Order of Data:**
* **Unsorted Data:** Use linear search unless sorting the data is feasible and the efficiency gain from binary search is needed.
* **Sorted Data:** Use binary search to take advantage of its logarithmic time complexity.
* **Frequency of Searches:**
* **Single/Few Searches:** Linear search might be more practical, especially if the data is unsorted.
* **Multiple Searches:** Sorting the data initially and using binary search will be more efficient overall.

Output:



