Library Management System with Search Algorithms

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class LibraryManagementSystem {

// Book class representing library books

public static class Book {

private final String bookId;

private final String title;

private final String author;

private boolean available;

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

this.bookId = bookId;

this.title = title;

this.author = author;

this.available = true;

}

// Getters

public String getBookId() { return bookId; }

public String getTitle() { return title; }

public String getAuthor() { return author; }

public boolean isAvailable() { return available; }

// Setter

public void setAvailable(boolean available) {

this.available = available;

}

@Override

public String toString() {

return String.format("%s: '%s' by %s [%s]",

bookId, title, author, available ? "Available" : "Checked Out");

}

}

private List<Book> books;

private boolean isSorted; // Tracks if books are sorted by title

public LibraryManagementSystem() {

books = new ArrayList<>();

isSorted = false;

}

// Add book to library - O(1) if not sorting, O(n log n) if sorting

public void addBook(Book book) {

books.add(book);

isSorted = false; // Adding a book may invalidate the sorted order

System.out.println("Added book: " + book);

}

// Sort books by title - O(n log n)

public void sortBooksByTitle() {

Collections.sort(books, Comparator.comparing(Book::getTitle));

isSorted = true;

System.out.println("Books sorted by title");

}

// Linear search by title - O(n)

public List<Book> linearSearchByTitle(String title) {

List<Book> results = new ArrayList<>();

for (Book book : books) {

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

results.add(book);

}

}

return results;

}

// Binary search by title (requires sorted list) - O(log n)

public List<Book> binarySearchByTitle(String title) {

if (!isSorted) {

System.out.println("Warning: Books not sorted, using linear search instead");

return linearSearchByTitle(title);

}

List<Book> results = new ArrayList<>();

int left = 0;

int right = books.size() - 1;

while (left <= right) {

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

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

if (comparison == 0) {

// Found a match, collect all matching books (there may be duplicates)

results.add(books.get(mid));

// Check left side for duplicates

int temp = mid - 1;

while (temp >= 0 && books.get(temp).getTitle().equalsIgnoreCase(title)) {

results.add(books.get(temp));

temp--;

}

// Check right side for duplicates

temp = mid + 1;

while (temp < books.size() && books.get(temp).getTitle().equalsIgnoreCase(title)) {

results.add(books.get(temp));

temp++;

}

return results;

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return results;

}

// Search by author (always uses linear search) - O(n)

public List<Book> searchByAuthor(String author) {

List<Book> results = new ArrayList<>();

for (Book book : books) {

if (book.getAuthor().equalsIgnoreCase(author)) {

results.add(book);

}

}

return results;

}

// Display all books - O(n)

public void displayAllBooks() {

System.out.println("\n=== Library Catalog (" + books.size() + " books) ===");

for (Book book : books) {

System.out.println(book);

}

System.out.println("=== End of Catalog ===");

}

public static void main(String[] args) {

LibraryManagementSystem library = new LibraryManagementSystem();

// Add books to the library

library.addBook(new Book("B001", "The Great Gatsby", "F. Scott Fitzgerald"));

library.addBook(new Book("B002", "To Kill a Mockingbird", "Harper Lee"));

library.addBook(new Book("B003", "1984", "George Orwell"));

library.addBook(new Book("B004", "The Great Gatsby", "F. Scott Fitzgerald")); // Duplicate title

library.addBook(new Book("B005", "Animal Farm", "George Orwell"));

library.addBook(new Book("B006", "Pride and Prejudice", "Jane Austen"));

// Display all books

library.displayAllBooks();

// Linear search example

System.out.println("\nLinear Search for 'The Great Gatsby':");

List<Book> linearResults = library.linearSearchByTitle("The Great Gatsby");

linearResults.forEach(System.out::println);

// Binary search (without sorting first)

System.out.println("\nBinary Search (without sorting first) for '1984':");

List<Book> binaryResults1 = library.binarySearchByTitle("1984");

binaryResults1.forEach(System.out::println);

// Sort books and perform binary search

library.sortBooksByTitle();

System.out.println("\nBinary Search (after sorting) for '1984':");

List<Book> binaryResults2 = library.binarySearchByTitle("1984");

binaryResults2.forEach(System.out::println);

// Search by author

System.out.println("\nSearch for books by George Orwell:");

List<Book> authorResults = library.searchByAuthor("George Orwell");

authorResults.forEach(System.out::println);

}

}

Key Components Explained

1. Search Algorithms

**Linear Search**:

* Sequentially checks each element until a match is found
* Works on both sorted and unsorted data
* Simple to implement
* Time complexity: O(n)

**Binary Search**:

* Requires sorted data
* Repeatedly divides the search interval in half
* Much faster than linear search for large datasets
* Time complexity: O(log n)

2. Book Class

* Contains bookId, title, author, and available status
* Includes getters and setter for availability
* Clean toString() method for display

3. Core Operations

**Adding Books**:

* Maintains a list of books
* Tracks whether the list is sorted

**Sorting**:

* Uses Collections.sort() with a title comparator
* Sets isSorted flag to true after sorting

**Searching**:

* Linear search works on unsorted data
* Binary search checks sort status and falls back to linear if needed
* Author search always uses linear search (since we're not sorting by author)

Time Complexity Analysis:

| **Operation** | **Time Complexity** | **Notes** |
| --- | --- | --- |
| Add book | O(1) or O(n log n) | O(1) if not sorting after add |
| Sort books by title | O(n log n) | Using Java's TimSort |
| Linear search by title | O(n) | Checks every book |
| Binary search by title | O(log n) | Only on sorted data |
| Search by author | O(n) | Always linear search |
| Display all books | O(n) | Must visit each book |

When to Use Each Search Algorithm

**Use Linear Search When**:

* The collection is small (n < 100)
* The data is unsorted and sorting isn't practical
* You need to search by multiple different fields (like author)
* You're adding/removing items frequently and maintaining sorted order is costly

**Use Binary Search When**:

* The collection is large (n > 1000)
* You can afford to keep the data sorted
* You're primarily searching by one field (title in this case)
* Searches are frequent but data changes are infrequent

Output:

