**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.

**Solutions:**

1. **Understanding:**

#### Linear Search

* **Algorithm**: Traverses each element in the list sequentially until the target element is found or the end of the list is reached.
* **Time Complexity**: O(n)
* **Best Case**: O(1) (if the target element is at the beginning of the list)
* **Worst Case**: O(n) (if the target element is at the end of the list or not present)

#### Binary Search

* **Algorithm**: Works on a sorted list by repeatedly dividing the search interval in half. If the target value is less than the middle element, search the left half; otherwise, search the right half.
* **Time Complexity**: O(log n)
* **Best Case**: O(1) (if the target element is the middle element)
* **Worst Case**: O(log n) (if the target element is at the ends of the list or not present)

1. **Setup and Implementation :**

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;

}

public String getBookId() {

return bookId;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

@Override

public String toString() {

return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author;}

}

public class LibraryManagementSystem {

private Book[] books;

private int size;

public LibraryManagementSystem(int capacity) {

books = new Book[capacity];

size = 0;

}

public void addBook(Book book) {

if (size < books.length) {

books[size++] = book;

} else {

System.out.println("Library is full. Cannot add more books.");

}

}

// Linear search to find books by title

public Book linearSearchByTitle(String title) {

for (int i = 0; i < size; i++) {

if (books[i].getTitle().equalsIgnoreCase(title)) {

return books[i];

}

}

return null;

}

// Binary search to find books by title (assuming the list is sorted)

public Book binarySearchByTitle(String title) {

int left = 0;

int right = size - 1;

while (left <= right) {

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

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

if (comparison == 0) {

return books[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

LibraryManagementSystem lms = new LibraryManagementSystem(5);

Book book1 = new Book("1", "Java Programming", "John Doe");

Book book2 = new Book("2", "Data Structures", "Jane Smith");

Book book3 = new Book("3", "Algorithms", "Alice Johnson");

lms.addBook(book1);

lms.addBook(book2);

lms.addBook(book3);

System.out.println("Searching for 'Data Structures' using Linear Search:");

Book foundBook = lms.linearSearchByTitle("Data Structures");

System.out.println(foundBook != null ? foundBook : "Book not found");

System.out.println("Searching for 'Algorithms' using Binary Search:");

foundBook = lms.binarySearchByTitle("Algorithms");

System.out.println(foundBook != null ? foundBook : "Book not found");

}

}

1. **Analysis:**

#### Time Complexity of Linear and Binary Search

**Linear Search**:

* 1. **Best Case**: O(1)
  2. **Worst Case**: O(n)
  3. **Average Case**: O(n/2) which simplifies to O(n)

**Binary Search**:

* 1. **Best Case**: O(1)
  2. **Worst Case**: O(log n)
  3. **Average Case**: O(log n)

#### When to Use Each Algorithm:

**Linear Search**:

* 1. **Unsorted Data**: Suitable for unsorted lists as it does not require any preprocessing.
  2. **Small Data Sets**: Efficient for small datasets where the overhead of sorting is not justified.

**Binary Search**:

* 1. **Sorted Data**: Requires the list to be sorted beforehand.
  2. **Large Data Sets**: More efficient for large datasets due to its logarithmic time complexity.
  3. **Static Data**: Ideal for datasets that do not change frequently, as maintaining the sorted order can be costly.