**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.

import java.util.Arrays;

class Book {

int bookId;

String title;

String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

}

public class LibraryManagementSystem {

public static int linearSearchByTitle(Book[] books, String title) {

for (int i = 0; i < books.length; i++) {

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

return i;

}

}

return -1;

}

public static int binarySearchByTitle(Book[] books, String title) {

int left = 0;

int right = books.length - 1;

while (left <= right) {

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

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

if (comparison == 0) {

return mid;

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Book[] books = {

new Book(1, "The Lord of the Rings", "J.R.R. Tolkien"),

new Book(2, "The fault in our stars", "John Green"),

new Book(3, "1984", "George Orwell")

};

int index = *linearSearchByTitle*(books, "The fault in our stars");

if (index != -1) {

System.***out***.println("Book found at index: " + index);

} else {

System.***out***.println("Book not found");

}

Arrays.*sort*(books, (b1, b2) -> b1.title.compareToIgnoreCase(b2.title));

index = *binarySearchByTitle*(books, "1984");

if (index != -1) {

System.***out***.println("Book found at index: " + index);

} else {

System.***out***.println("Book not found");

}

}

}

OUTPUT:

Book found at index: 1

Book found at index: 0

**1. Linear Search:** Linear search is a simple algorithm that iterates through a list sequentially, comparing each element with the target value we're looking for. It continues until it finds a match or reaches the end of the list.

**2.Time Complexity:**

**Best Case:** O(1) - This occurs when the target element is the first element in the list. We only need one comparison.

**Average Case:** O(n) - On average, we need to compare the target element with half the elements in the list. This grows linearly with the size of the list (n).

**Worst Case:** O(n) - This happens when the target element is not present in the list or is the last element. We need to compare it with all elements.

**3. Binary Search:** Binary search is a much faster algorithm but requires the list to be sorted in ascending order. It works by repeatedly dividing the search space in half. It compares the target value with the middle element of the remaining list.

If the target value is equal to the middle element, the search is successful.

If the target value is less than the middle element, the search continues in the lower half of the list.

If the target value is greater than the middle element, the search continues in the upper half of the list.

**4.Time Complexity:**

**Best Case:** O(1) - This occurs when the target element is the middle element in the first iteration. We only need one comparison.

**Average Case:** O(log n) - On average, the search space is halved with each iteration, leading to logarithmic growth in comparisons with the list size (n).

**Worst Case:** O(log n) - Similar to the average case, even in the worst scenario (target element not present), we still only need to compare it with a logarithmic number of elements.

**5.Use Linear Search when:**

The data set is small (a few elements). The constant time complexity outweighs the benefit of a faster search in binary search.

The data is unsorted. Binary search requires a sorted list.

**Use Binary Search when:**

The data set is large (hundreds or thousands of elements). Binary search's logarithmic time complexity makes it much faster.

The data is sorted and you plan to perform multiple searches on it. The initial sorting cost is amortized (spread out) over the faster searches.