**Solution 6: Library Management System**

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

**>> Explain linear search and binary search algorithms.**

Understanding Search Algorithms

Linear Search

* Description: Linear search is a simple search algorithm that checks each element in a list sequentially until the desired element is found or the end of the list is reached.
* Time Complexity: O(n), where n is the number of elements in the list. In the worst case, it requires checking every element.
* Space Complexity: O(1) as it uses a constant amount of extra space.

How it Works:

1. Start from the beginning of the list.
2. Compare each element with the target value.
3. If the element matches, return the index or the element.
4. If the end of the list is reached without finding the target, return a "not found" indicator.

Binary Search

* Description: Binary search is a more efficient search algorithm that requires the list to be sorted. It works by repeatedly dividing the search interval in half.
* Time Complexity: O(logn), where n is the number of elements in the list. This is because each comparison reduces the search space by half.
* Space Complexity: O(1) for iterative implementation or O(logn) for recursive implementation due to call stack usage.

How it Works:

1. Start with the entire list.
2. Compare the target value with the middle element of the list.
3. If the target matches the middle element, return the index or the element.
4. If the target is smaller, search the left half of the list.
5. If the target is larger, search the right half of the list.
6. Repeat until the target is found or the search interval is empty.

**Analysis**

**>> Compare the time complexity of linear and binary search.**

**Time Complexity Comparison**

1. **Linear Search**:
   * **Time Complexity**: O(n)
     + In the worst case, linear search must check each element of the list to find the target value or determine that it is not present. This means it scales linearly with the size of the dataset.
   * **Space Complexity**: O(1)
     + Linear search uses a constant amount of additional space regardless of the input size.
2. **Binary Search**:
   * **Time Complexity**: O(logn)
     + Binary search divides the search interval in half at each step, which makes it significantly faster than linear search for large datasets. The time complexity grows logarithmically with the size of the dataset.
   * **Space Complexity**:
     + **Iterative Implementation**: O(1)– Uses a constant amount of additional space.
     + **Recursive Implementation**: O(logn) – Due to the recursive call stack.

**>> Discuss when to use each algorithm based on the data set size and order.**

**1. Linear Search:**

* **Data Set Size**:
  + Effective for small to moderately sized datasets where performance isn't a critical issue.
* **Order of Data**:
  + Does not require the data to be sorted. It works equally well with sorted or unsorted lists.
* **Use Case**:
  + Ideal for cases where the data is not sorted, or where the dataset is small enough that the difference in efficiency is negligible.
  + Suitable for simple or one-time searches where sorting the data would not be practical or beneficial.

**2. Binary Search:**

* **Data Set Size**:
  + Best suited for large datasets where the efficiency of searching is crucial. The logarithmic time complexity provides a significant performance boost over linear search as the dataset grows.
* **Order of Data**:
  + Requires the data to be sorted. If the data is not sorted, binary search cannot be performed correctly without first sorting the data.
* **Use Case**:
  + Ideal for scenarios where searches are frequent and the dataset is large. For instance, it is useful in applications involving sorted databases or large-scale search systems.
  + If the data set is modified frequently (i.e., elements are added or removed often), maintaining a sorted list might involve additional overhead.

**Linear Search** is straightforward and works on any unsorted list. It is easy to implement and doesn't require preprocessing. It is preferred for small datasets or when the data is frequently changing, as it doesn't require sorting.

**Binary Search** is more efficient for large datasets due to its O(logn) time complexity, but it requires the list to be sorted. For static datasets where search operations are frequent, binary search is ideal due to its performance advantages over linear search. However, maintaining a sorted dataset involves additional complexity, especially if the dataset changes frequently.

**How to Run the code :**

* Run Main.java file