### Linear Search and Binary Search Algorithms

Linear Search:

- Description: Sequentially checks each element until the target is found or the end is reached.

- Time Complexity:

- Best Case: O(1) - Target is the first element.

- Average/Worst Case: O(n) - Target is in the middle/end or not present.

- Advantages: Simple, works on unsorted arrays.

- Disadvantages: Inefficient for large datasets.

Binary Search:

- Description: Searches a sorted array by repeatedly dividing the search interval in half.

- Time Complexity:

- Best Case: O(1) - Target is the middle element.

- Average/Worst Case: O(log n) - Interval reduces by half each time.

- Advantages: Efficient for large, sorted datasets.

- Disadvantages: Requires the array to be sorted.

### Analysis of Library Management System

1. Time Complexity Analysis:

- Linear Search:

- Best Case: O(1) - The book to be found is the first book.

- Worst Case: O(n) - The book to be found is the last book, or it does not exist in the array.

- Binary Search:

- Best Case: O(1) - The book to be found is at the midpoint on the first comparison.

- Worst Case: O(log n) - The search space is halved with each comparison, resulting in logarithmic time complexity.

2. Comparison of Time Complexity:

- Linear Search:

- Linear search checks each element one by one, which makes it O(n) in the worst case. It is simple and doesn't require the array to be sorted. It is suitable for small or unsorted datasets.

- Binary Search:

- Binary search, on the other hand, is much more efficient with a time complexity of O(log n), but it requires the array to be sorted. It is ideal for large datasets where the overhead of sorting is outweighed by the faster search time.

3. Discussion on When to Use Each Algorithm:

- Linear Search:

- Use linear search when the dataset is small or unsorted.

- It is straightforward and doesn't require preprocessing (i.e., sorting).

- Ideal for cases where the number of searches is minimal, making the overhead of sorting unnecessary.

- Binary Search:

- Use binary search for large datasets where the array can be kept sorted.

- It is more efficient for repeated searches after the initial sorting.

- The overhead of sorting is justified if the search operation is frequent and the dataset is relatively static or changes infrequently.

Example Analysis Using Code:

- Adding Books:

- The `addBook` method adds a new book to the array. This operation is O(1) if there is space in the array.

- Searching for a Book by Title:

- The `linearSearchByTitle` method traverses the array linearly to find the book with the given title. This is O(n) in the worst case.

- The `binarySearchByTitle` method performs a binary search on a sorted array, which is O(log n) in the worst case.

- Sorting Books:

- The `sortBooksByTitle` method sorts the array using the built-in `Arrays.sort` method. The time complexity for this operation is O(n log n).

Conclusion:

- Linear Search:

- Suitable for small or unsorted datasets with minimal search operations.

- Simple and requires no preprocessing.

- Binary Search:

- More efficient for large, sorted datasets with frequent search operations.

- The initial overhead of sorting is justified by the faster search times.

- The choice of search algorithm depends on the size and nature of the dataset, as well as the frequency and type of search operations required.