1. **Understand Search Algorithms:**

* ***Linear Search:***

*Definition:*  Linear search is a simple search algorithm that checks each element in a list sequentially until the desired element is found or the list ends.

*Procedure*:

1. Start from the first element of the list.
2. Compare the current element with the target element.
3. If they match, return the current element or its index.
4. If they don't match, move to the next element and repeat the process.
5. If the end of the list is reached and the element is not found, return null or a value indicating the element is not present.

**Time Complexity:** O(n), where n is the number of elements in the list. This is because, in the worst case, each element must be checked once.

* ***Binary Search:***

*Definition*: Binary search is a more efficient algorithm than linear search but requires that the list be sorted. It works by repeatedly dividing the search interval in half**.**

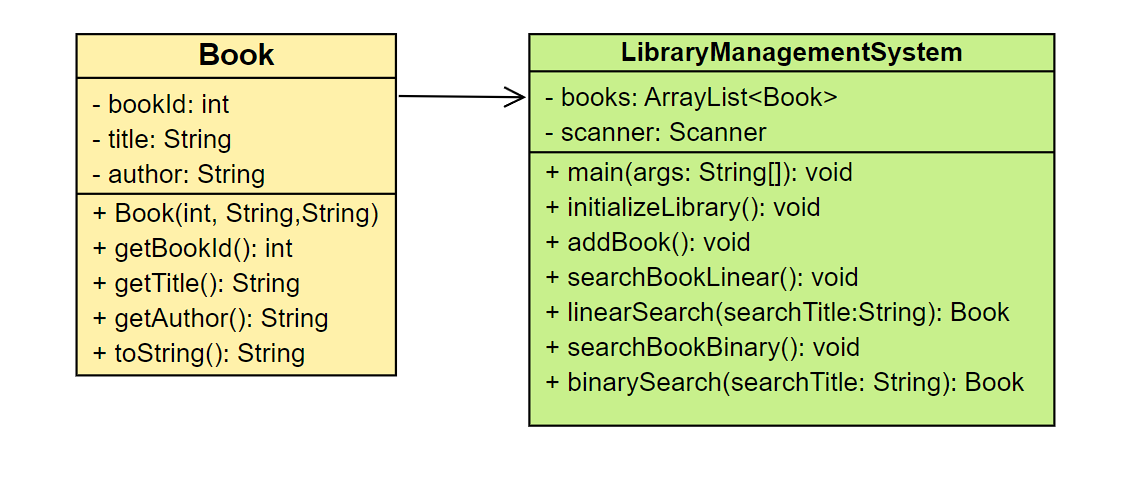
*Procedure***:**

1. Start with the entire list as the search interval.
2. Find the middle element of the list.
3. If the middle element matches the target, return the element or its index.
4. If the target is less than the middle element, repeat the search on the left half of the list.
5. If the target is greater than the middle element, repeat the search on the right half of the list.
6. Continue this process until the target is found or the interval is empty.

**Time Complexity:** O(log n), where n is the number of elements in the list. This is due to the halving of the search space in each step.

1. **Setup:**

* **Class Book:** To manage a collection of books, we create a class named *Book* with attributes such as *bookId*, *title*, and *author*. The *bookId* uniquely identifies each book, while the *title* and *author* provide additional descriptive information. This class encapsulates the properties of a book and serves as the basis for operations like searching and sorting.

1.  **Implementation:**

* **Linear Search Implementation:** To find a book by title using linear search, iterate through the list of *Book* objects and compare the *title* attribute of each book with the search query. This method does not require the list to be sorted, making it simple but potentially slow for large datasets, as it may need to scan each element.
* **Binary Search Implementation:** For binary search, ensure the list of *Book* objects is sorted by the *title* attribute. Once sorted, the binary search algorithm can be applied. This involves comparing the search query with the middle element's title, then narrowing down the search to either the left or right half based on the comparison result. This method significantly reduces the number of comparisons needed, especially for large lists.

Here is the github code-  [Link](https://github.com/Akashmondal55/Akash_5016855/tree/main/Week-1/DSA/Exercise-6)

1. **Analysis:**

* **Time Complexity Comparison:**
* *Linear Search:* The time complexity of linear search is O(n), where n is the number of elements in the list. In the worst case, the search will check all elements.
* *Binary Search:* The time complexity of binary search is O(log n), assuming the list is sorted. This logarithmic time complexity results from halving the search space with each comparison.
* **When to Use Each Algorithm:**
* *Linear Search:* This method is appropriate when the list is unsorted or small. It is straightforward to implement and does not require any prior sorting of the data.
* *Binary Search:* This method is efficient for large datasets but requires the list to be sorted beforehand. If the data is already sorted or can be sorted with minimal effort, binary search is preferable due to its faster performance on large lists.

Choosing between these algorithms depends on the dataset's size and order, with linear search being more flexible but less efficient for large datasets, and binary search offering faster performance at the cost of requiring sorted data.