**LIBRARY MANAGEMENT SYSTEM**

**Linear Search**

Linear search is a basic technique used to find a particular element in a list or array. It checks each item one by one from the beginning until the required element is located or the list ends.

**How It Works:**

1. Start checking from the first element of the array.
2. Compare each element with the target value.
3. If a match is found, return the position or value.
4. If not, move to the next element.
5. Continue until the end is reached.
6. If the element is not found, indicate that the item doesn’t exist in the list.

**Time Complexity:**

* **Worst-case:** O(n), where *n* is the total number of elements.  
  This means it might have to check every item to find a match (or determine it’s not present).

**Example:**  
In the array [3, 5, 7, 9, 11], to find 7, the search checks 3, then 5, and then 7 is found at the third position.

**Binary Search**

Binary search is a much quicker search method, but it only works on sorted data. It works by repeatedly dividing the search range in half and checking the middle element until the target is located or the range becomes empty.

**How It Works:**

1. Begin with the entire sorted array.
2. Check the middle element.
3. If it matches the target, return it.
4. If the target is smaller, continue the search in the left half.
5. If larger, search the right half.
6. Repeat the process until the item is found or the search space is empty.

**Time Complexity:**

* **Worst-case:** O(log n), where n is the number of elements.  
  This efficiency comes from cutting the search space in half each time.

**Example:**  
Given a sorted array [3, 5, 7, 9, 11], searching for 7 starts by checking the middle, which is already 7, so it's found immediately.

**Comparison of the time complexity of linear and binary search**

**Linear Search:**

Time Complexity: O(n). In the worst case, you may need to check each element in the list. If there are `n` elements, you may need to perform `n` comparisons.

**Binary Search:**

Time Complexity: O(log n). Binary search works by repeatedly dividing the search space in half. With each step, it reduces the search interval, so the number of comparisons needed grows logarithmically with the number of elements.

**When to Use Each Algorithm**

**When to Use Linear Search:**

* **Unsorted Collections:** Ideal when the data is not sorted.
* **Small Datasets:** For a small number of elements, performance differences are minimal.
* **Quick & Simple:** Easy to code, suitable for rare search operations.

**Advantages:**

* Very simple to write and understand.
* Works on all kinds of data—sorted or unsorted.

**Disadvantages:**

* Slower on large datasets since every element might need to be checked.

**When to Use Binary Search:**

* **Sorted Arrays/Lists:** Works only if the data is already sorted.
* **Large Datasets:** Much more efficient for bigger lists due to log-based complexity.
* **Frequent Searches:** Best for scenarios with repeated search queries.

**Advantages:**

* Far more efficient on larger, ordered datasets.
* Fewer comparisons due to reduced search space.

**Disadvantages:**

* Cannot be applied to unsorted data directly.
* Slightly more complex to implement compared to linear search.