**Understand Search Algorithms**

**Linear Search**

* **Algorithm**: A linear search algorithm sequentially checks each element in a list until the desired element is found or the list ends.
* **Time Complexity**:
  + Best Case: O(1)O(1)O(1) - When the element is at the beginning of the list.
  + Average and Worst Case: O(n)O(n)O(n) - When the element is in the middle, end, or not in the list.

**Binary Search**

* **Algorithm**: A binary search algorithm divides a sorted list into halves, compares the target value to the middle element, and then continues searching in the half where the target value is likely to be.
* **Time Complexity**:
  + Best, Average, and Worst Case: O(log⁡n)O(\log n)O(logn) - Because the list is halved with each step.

**Analysis**

**Time Complexity Comparison**

* **Linear Search**:
  + **Best Case**: O(1)O(1)O(1) - The desired book is at the beginning.
  + **Average and Worst Case**: O(n)O(n)O(n) - The desired book is in the middle, end, or not present in the list.
* **Binary Search**:
  + **Best, Average, and Worst Case**: O(log⁡n)O(\log n)O(logn) - The list is halved with each step, significantly reducing the number of comparisons.

**When to Use Each Algorithm**

* **Linear Search**:
  + **Use Case**: Suitable for small or unsorted data sets where the overhead of sorting the list is not justified.
  + **Advantage**: Simple to implement and does not require the list to be sorted.
  + **Disadvantage**: Inefficient for large data sets due to O(n)O(n)O(n) time complexity.
* **Binary Search**:
  + **Use Case**: Ideal for large, sorted data sets where the efficiency of search operations is crucial.
  + **Advantage**: Much faster for large data sets due to O(log⁡n)O(\log n)O(logn) time complexity.
  + **Disadvantage**: Requires the list to be sorted, adding overhead for maintaining the sorted order after insertions and deletions.