**Linear Search**

* **Description:** Scans each element in a list or array sequentially to find a target value.
* **Algorithm:**
  1. Check each element from start to end.
  2. Return index if the target is found.
  3. If the end is reached, the target is not present.
* **Time Complexity:**
  1. **Best Case:** O(1)
  2. **Average Case:** O(n)
  3. **Worst Case:** O(n)
* **Usage:** Ideal for unsorted lists.

**Binary Search**

* **Description:** Efficiently searches a sorted list by repeatedly dividing the search interval in half.
* **Algorithm:**
  1. Set pointers low and high.
  2. Compare target with the middle element.
  3. Adjust pointers based on comparison and repeat.
* **Time Complexity:**
  1. **Best Case:** O(1)
  2. **Average Case:** O(log n)
  3. **Worst Case:** O(log n)
* **Usage:** Suitable for sorted lists.

**Summary**

* **Linear Search:** Simple, works on unsorted data, but slower (O(n)).
* **Binary Search:** Fast (O(log n)), but requires sorted data.

**Analysis:**

### Time Complexity Comparison of Linear and Binary Search

### Linear Search:-

**Linear search** examines each element in the array sequentially until the target element is found or the end of the array is reached.

* **Best-Case:** The target element is the first element.  
  **Time Complexity:** O(1)
* **Average-Case:** The target element is expected to be in the middle of the array.  
  **Time Complexity:** O(n/2) ≈ O(n)
* **Worst-Case:** The target element is the last element or not present.  
  **Time Complexity:** O(n)

**Binary Search:-**

**Binary search** repeatedly divides the sorted array in half, eliminating half of the remaining elements with each comparison.

* **Best-Case:** The target element is the middle element.  
  **Time Complexity:** O(1)
* **Average-Case:** The target element is generally found after log(n) comparisons.  
  **Time Complexity:** O(log n)
* **Worst-Case:** The target element is not present, and the array is reduced to a single element.  
  **Time Complexity:** O(log n)

### When to Use Each Search Algorithm:

### Linear Search:-

**When to Use:**

* **Data Set Size:** Small to moderate size where performance isn't a critical concern.
* **Data Order:** Works with unsorted or unordered data.

**Advantages:**

* Simple and easy to implement.
* No need for data to be sorted.

**Disadvantages:**

* Inefficient for large datasets due to O(n) time complexity.

**Ideal Use Cases:**

* Quick searches in small lists or arrays.
* Situations where sorting the data is impractical or unnecessary.

**Binary Search:-**

**When to Use:**

* **Data Set Size:** Large datasets where efficiency is important.
* **Data Order:** Requires sorted data.

**Advantages:**

* Much faster than linear search for large datasets with O(log n) time complexity.
* Efficiently narrows down the search space.

**Disadvantages:**

* Requires data to be sorted beforehand.
* More complex to implement compared to linear search.

**Ideal Use Cases:**

* Searching in large, pre-sorted arrays or lists.
* Scenarios where data can be sorted or is already sorted.

So, in essence linear search is used for small or unsorted datasets where simplicity is preferred and binary search is used for large, sorted datasets where performance is critical.