**Exercise 6: Library Management System**

Explain linear search and binary search algorithms.

**Linear Search**

**Definition:**

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

**How It Works:**

1. Start from the first element of the array or list.
2. Compare each element with the target value.
3. If a match is found, return the index or confirm the value is present.
4. If the end of the list is reached without a match, return that the value is not found.

**Example Scenario:**

Searching for the number 15 in the list: [4, 7, 15, 20, 33]

* Start at index 0: 4 → not a match.
* Index 1: 7 → not a match.
* Index 2: 15 → match found.

**Time Complexity:**

|  |  |
| --- | --- |
| **Case** | **Complexity** |
| Best Case | O(1) |
| Average Case | O(n) |
| Worst Case | O(n) |

**Space Complexity:**

* O(1) — No additional memory is needed beyond the input list.

**Advantages:**

* Works on **unsorted data**.
* Simple to implement.
* No additional memory required.

**Disadvantages:**

* Inefficient for large datasets.
* Checks every element even if the list is sorted.

**Binary Search**

**Definition:**

**Binary Search** is an efficient search algorithm that works on **sorted arrays** by repeatedly dividing the search interval in half.

**Precondition:**

The data **must be sorted** in ascending or descending order.

**How It Works:**

1. Start with two pointers: low (start of the list) and high (end of the list).
2. Find the **middle index**.

* Compare the middle element with the target:
  + If equal, the search is complete.
  + If the target is smaller, repeat the search in the **left half**.
  + If the target is larger, repeat the search in the **right half**.

1. Repeat steps 2–3 until the element is found or the interval becomes empty.

**Example Scenario:**

Searching for 15 in [4, 7, 15, 20, 33]

* Start with middle element (index 2): 15 → match found.

**Time Complexity:**

| **Case** | **Complexity** |
| --- | --- |
| Best Case | O(1) |
| Average Case | O(log n) |
| Worst Case | O(log n) |

**Space Complexity:**

* O(1) for iterative version.
* O(log n) for recursive version (due to function call stack).

**Advantages:**

* Very fast for large **sorted** datasets.
* Reduces time complexity significantly compared to linear search.

**Disadvantages:**

* Only works on **sorted** data.
* Requires more complex logic than linear search.

Compare the time complexity of linear and binary search.

**1. Linear Search**

* **Best Case (O(1))**: The target is found at the first index.
* **Average Case (O(n))**: The target is somewhere in the middle; about half of the elements are checked.
* **Worst Case (O(n))**: The target is at the last index or not present at all; every element is checked.

**2. Binary Search**

* **Best Case (O(1))**: The target is at the middle of the array on the first check.
* **Average Case (O(log n))**: The list is halved at each step; the target is found in one of the sublists.
* **Worst Case (O(log n))**: The target is not found, but all levels of division are traversed.

**Conclusion:**

* **Binary search is significantly more efficient** than linear search for large, sorted datasets due to its logarithmic time complexity.
* **Linear search** is more flexible and works on unsorted data but becomes inefficient as the dataset grows.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| **Linear Search** | O(1) | O(n) | O(n) |
| **Binary Search** | O(1) | O(log n) | O(log n) |

Discuss when to use each algorithm based on the data set size and order.

**When to Use Linear Search**

**1. Unsorted Data**

We should use linear search when the data is not sorted. Since linear search does not depend on the order of elements, it works on any type of dataset without preprocessing.

**2. Small Data Sets**

Linear search is suitable for small datasets where the time difference between linear and binary search is minimal. In such cases, the simplicity of linear search often outweighs the slight performance gain of binary search.

**3. Few or One-Time Searches**

If we need to perform a search only once or very infrequently, linear search is often more practical, especially if the cost of sorting the data (required for binary search) is not justified.

**4. Frequently Changing Data**

When the dataset is frequently updated with insertions and deletions, maintaining a sorted list can be inefficient. In these situations, using linear search avoids the overhead of constantly re-sorting the data.

**When to Use Binary Search**

**1. Sorted Data**

Binary search should be used when the data is already sorted. It is designed to take advantage of order by repeatedly dividing the search space in half.

**2. Large Data Sets**

For large datasets, binary search is significantly faster than linear search. Its logarithmic time complexity makes it highly efficient as the size of the dataset increases.

**3. Repeated Searches**

When we need to perform multiple searches on the same dataset, it may be worthwhile to sort the data once and then use binary search for all subsequent lookups.

**4. Performance-Critical Applications**

In applications where search speed is crucial, such as in large databases or real-time systems, binary search provides better performance than linear search, provided the data is sorted or can be sorted efficiently.

**Summary Table**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Use Linear Search** | **Use Binary Search** |
| Data is unsorted | Yes | No |
| Data is sorted | No | Yes |
| Data set is small | Yes | Optional |
| Data set is large | No | Yes |
| Few or one-time searches | Yes | Optional |
| Many repeated searches | No | Yes |
| Data changes frequently (unsorted) | Yes | No |
| High performance required | No | Yes |

**Conclusion**

We should use linear search for small or unsorted datasets, or when simplicity is more important than performance. Binary search is more appropriate for large, sorted datasets where performance matters or searches are frequent. Each algorithm has its strengths depending on the context and nature of the data.