**SEARCHING ALGORITHM**

**(LINEAR SEARCH & BINARY SEARCH)**

1. **What is the time complexity of Linear Search and Binary Search in the best, worst, and average cases?**

**Ans:**

**Linear Search:**

* **Best case:** O(1) (if the element is found at the first position).
* **Worst case:** O(N) (if the element is at the last position or not present).
* **Average case:** O(N).

**Binary Search:**

* **Best case:** O(1) (if the element is found in the first check).
* **Worst case:** O(log N) (keeps dividing the list into halves).
* **Average case:** O(log N).

1. **Given the array arr[] = {12, 3, 43, 5, 67, 34, 23, 45, 21, 4} and target element x = 5, perform a Linear Search and determine the number of comparisons required to find the target.**

**Ans:**

Step-by-step Linear Search:

Compare 5 with 12 → Not found.

Compare 5 with 3 → Not found.

Compare 5 with 43 → Not found.

Compare 5 with 5 → Found at index 3 (4th position).

Total comparisons required: 4.

1. **For an array of size N, calculate the worst-case time complexity of Linear and Binary Search.**

**Ans:**

**Linear Search:** In the worst case, it checks all N elements → O(N).

**Binary Search:** In the worst case, it keeps dividing by 2 until 1 element is left:

Number of steps ≈ log₂ N → O(log N).

**Example Calculation:**

For N = 1000,

**Linear Search:** Worst case = 1000 comparisons.

**Binary Search:** Worst case = log₂ 1000 ≈ 10 comparisons.

1. **Explain why Binary Search requires a sorted array.**

**Ans:**

* It checks the middle element and decides to search left or right.
* Unsorted Array can not determine the element's position, making Binary Search ineffective.

1. **If a list contains 1000 elements, estimate the number of comparisons Binary Search takes in the worst case.**

**Ans:**

Given,

Number of elements, N=1000.

**Formula:**

Number of comparisons in the worst case ≈ log2N.

Therefore, log21000 ≈ 10.

**Number of Comparisons:** Approximately 10.

1. **Given an array arr[] = {3, 4, 5, 12, 21, 23, 34, 43, 45, 67} and target x = 5, perform Binary Search step by step and find the number of comparisons needed.**

**Ans:**

**Step 1:** Middle index = (0+9)/2 = 4 → arr[4] = 21 (too large, search left).

**Step 2:** Middle index = (0+3)/2 = 1 → arr[1] = 4 (too small, search right).

**Step 3:** Middle index = (2+3)/2 = 2 → arr[2] = 5 (FOUND!).

**Total comparisons =** 3.

1. **Why is mid = L + (H - L) / 2 preferred over mid = (L + H) / 2 in Binary Search?**

**Ans:**

* **(L + H) / 2** can cause an integer overflow for very large values of L and H.
* **(L + (H - L) / 2)** prevents overflow by calculating the middle safely.

1. **What happens if coincidentally Linear Search and Binary Search require the same number of comparisons?**

**Ans:**

This can happen only in small datasets where both searches might take similar steps. However, as the list size grows, Binary Search always performs better than Linear Search because it reduces the search space faster.

**Example:**

* If the list has 4 elements, both Linear Search (worst case) and Binary Search (worst case) may take 2 comparisons.
* But for larger lists, Binary Search will always need fewer comparisons than Linear Search.

Thus, even if they are equal in rare cases, Binary Search remains the better choice for large datasets.

1. **Compare the number of comparisons required in the worst case for Linear Search and Binary Search when searching a list of 1,000,000 elements.**

**Ans:**

* **Linear Search:** Worst case = 1,000,000 comparisons.
* **Binary Search:** Worst case = log₂(1,000,000) ≈ 20 comparisons.
* Binary Search is much faster than Linear Search for large lists.

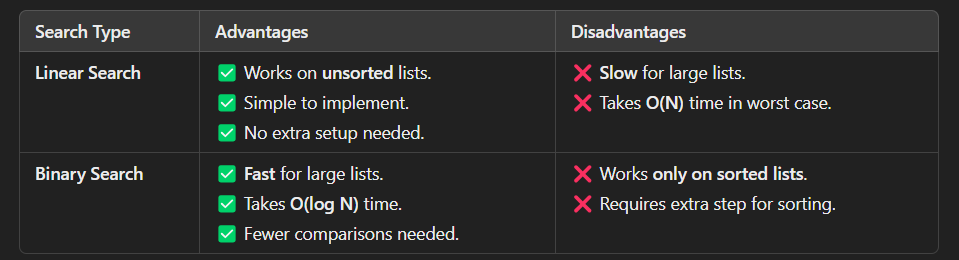
1. **What happens if the target element is not found in both Linear and Binary Search?**

**Ans:**

* **Linear Search:** It checks all elements and returns -1 (not found).
* **Binary Search:** It keeps dividing the list but fails to find the element, returning -1.

1. **Advantages and Disadvantages of Linear search and Binary search algorithm.**

**Ans:**



1. **Differentiate between Linear search and Binary search algorithm.**

**Ans:**

| **Feature** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Definition** | Searches for an element by checking each item one by one. | Searches by dividing the list into halves. |
| **Data Requirement** | Works on both sorted and unsorted lists. | Works only on **sorted** lists. |
| **Time Complexity (Best Case)** | **O(1)** (if the element is at the first position). | **O(1)** (if the element is at the middle). |
| **Time Complexity (Worst Case)** | **O(N)** (searches all elements). | **O(log N)** (keeps dividing by 2). |
| **Efficiency** | Slower for large datasets. | Much faster for large datasets. |
| **Method Used** | **Sequential search** (one by one). | **Divide and conquer** (splitting the list). |
| **Number of Comparisons (Worst Case)** | **N** (for N elements). | **log₂ N** (for N elements). |
| **Example Search for 5 in [3, 4, 5, 12, 21]** | Checks 3 → 4 → **5 (found)**. | Middle = 5 → **found immediately**. |
| **Best Use Case** | Small or **unsorted** lists. | Large and **sorted** lists. |

1. **If a Linear Search algorithm takes 0.01 seconds for an input size of 100, estimate the time required for an input size of 10,000 (assuming O(N) complexity).**

**Ans:**

* Time Complexity of Linear Search is O(N), so time increases linearly.
* If 100 elements → 0.01 sec, then 10,000 elements → (10,000 / 100) × 0.01 sec = 1 sec.
* **Formula:** T2/T1 = N2/N1