**1.Understand Asymptotic Notation:**

**Explain Big O notation and how it helps in analyzing algorithms.**

**Definition:**

Big O Notation is a mathematical representation used to describe the **upper bound of an algorithm’s time or space complexity** as the input size (n) increases. It gives a **general idea of the algorithm’s efficiency**, especially when the input becomes very large. Big O ignores constants and focuses only on the term that grows the fastest as n increases.

For example:

* If a function has complexity O(n + 5), we consider it as O(n).
* If a function has complexity O(3n² + 2n + 10), we simplify it to O(n²).

**How It Helps in Analyzing Algorithms:**

* **Performance Estimation**  
  Big O helps us estimate how much **time** or **memory** an algorithm will need, based on the input size. It tells us if the algorithm is practical for small, medium, or large datasets.
* **Scalability Check**  
  It lets developers see if an algorithm can handle large inputs efficiently. For instance, an algorithm with O(n log n) will usually scale much better than one with O(n²).
* **Comparison Tool**  
  Big O gives a standard way to **compare multiple algorithms**. Even before writing code, we can decide which one is more efficient by checking their time complexities.
* **Optimization Guide**  
  When trying to improve performance, Big O helps identify the **bottlenecks** and guides developers toward more optimized solutions.

**Describe the best, average, and worst-case scenarios for search operations.**

When analyzing a search algorithm, it’s important to look at how it performs under different conditions. These are generally classified as Best Case, Average Case, and Worst Case.

**Best Case**

**Definition:**

The best-case scenario occurs when the element is found immediately or with the least effort. This is the fastest possible outcome for the algorithm.

**Example (Linear Search):**

Searching for 3 in [3, 7, 9, 12, 15]

→ The element 3 is found at the first index (0).

Time Complexity:

O(1) – because only one comparison is needed.

**Average Case**

**Definition:**

The average case assumes that the input is random or typical, and the target element is located somewhere in the middle. It gives an idea of the expected performance over multiple inputs.

**Example (Linear Search):**

Searching for 12 in [3, 7, 9, 12, 15, 18]

→ The element is found at index 3.

Time Complexity:

O(n) – On average, half the elements will be checked before finding the item.

**Worst Case**

**Definition:**

The worst-case scenario happens when the element is at the last position or not present at all. This represents the maximum amount of work the algorithm will do.

**Example (Linear Search):**

Searching for 20 in [3, 7, 9, 12, 15, 18]

→ The element is not found after checking all 6 items.

Time Complexity:

O(n) – Every element must be checked.

**4.Analysis**

**Compare the time complexity of linear and binary search algorithms.**

|  |  |  |
| --- | --- | --- |
| **Best Case** | Found at the 1st index | O(1) |
| **Average Case** | |  | | --- | |  |  |  | | --- | | Found somewhere in the middle | | |  | | --- | |  |  |  | | --- | | O(n) | |
| **Worst Case** | Not found or found at the last index | O(n) |

Time Complexity Comparison:

**Linear Search:**

**Best Case:** O(1) (if the element is at the start)

**Average/Worst Case**: O(n)

**Binary Search:**

**Best Case:** O(1)

**Average/Worst Case**: O(log n)

**Discuss which algorithm is more suitable for your platform and why.**

**Binary search** is more suitable for an e-commerce platform where the product list is large and frequently searched. It offers much faster performance (**O(log n)**) compared to linear search, provided the data is sorted. For dynamic data, maintain sorted lists or use indexed structures like trees or hash maps for optimal results.