**1.Big O Notation and How It Helps in Analyzing Algorithms**

Big O notation is a way to describe the efficiency of an algorithm, especially when the input size becomes very large. It shows how the time or space used by the algorithm increases with input. We mainly use Big O to focus on the **worst-case** scenario, which is important in real-world systems where speed and memory matter.

For example, in a **linear search**, we go through the list one by one to find an element. If the list has n elements, in the worst case, we might have to check all n items. So, the time complexity is **O(n)**.

But in a **binary search**, which only works on sorted arrays, we divide the array in half each time. This makes it much faster. Its time complexity is **O(log n)** in the worst case.

So, Big O helps us compare these algorithms and understand which one is better for large data sets.

**2. Best, Average, and Worst-Case Scenarios in Search Operations**

When analyzing search algorithms, we look at three cases:

**Best Case:**  
This is the fastest possible scenario.  
Example: In linear search, if the element we are looking for is at the first position, we find it immediately.  
**Time Complexity: O(1)** (constant time)

**Average Case:**  
This is the expected case when the input is random.  
Example: In linear search, we might find the element somewhere in the middle of the list.  
**Time Complexity: O(n)** (on average, n/2 elements checked)

**Worst Case:**  
This is the slowest scenario.  
Example: In linear search, the element is not in the list or is at the last position, so we have to check every item.  
**Time Complexity: O(n)**

For binary search:

**Best Case:** The element is exactly in the middle → **O(1)**

**Average and Worst Case:** We keep dividing until one element is left → **O(log n)**