**Understanding- Big O Notation**

The Big O Notation basically helps us understand how efficient an algorithm is when the size of the input grows. It gives a rough idea of how much time or space an algorithm will take in the worst-case scenario, which is important when working with large datasets, like in an e-commerce platform that could have thousands of products.

So instead of focusing on exact time, it focuses on how the performance scales. This helps us compare algorithms and figure out which one is better suited for different situations.

**Best, Average, and Worst Case (in Search):**

|  |  |  |
| --- | --- | --- |
| **Case** | **Linear Search** | **Binary Search** |
| **Best** | O(1) | O(1) |
| **Average** | O(n/2) ~ O(n) | O(log n) |
| **Worst** | O(n) | O(log n) |

**Analysis-**  
  
**Time Complexity Comparison:**

|  |  |
| --- | --- |
| **Algorithm** | **Time Complexity** |
| **Linear Search** | O(n) |
| **Binary Search** | O(log n) |

For an e-commerce platform, we care about speed. Users won’t wait for results to load. That’s why binary search is a better fit, as long as the product list is kept sorted. But if the product list is frequently changing or unsorted (maybe during some backend updates), then linear search can be a quick and temporary solution.