**What is Big O Notation?**

**Big O Notation** describes the **worst-case** time or space complexity of an algorithm as the input size (n) grows. It helps evaluate **efficiency** regardless of hardware.

* It **abstracts away constants** and focuses on how runtime or memory usage **scales**.
* Used to **compare algorithms** and decide which is better for **large inputs**.

Best, Average, and Worst-Case Scenarios (Search Operations)

| **Scenario** | **Description** | **Example** |
| --- | --- | --- |
| **Best Case** | The item is found early (e.g., at the start) | O(1) for linear search |
| **Average Case** | The item is somewhere in the middle | O(n/2) for linear search ≈ O(n) |
| **Worst Case** | The item is not found or at the end | O(n) for linear, O(log n) for binary |

Linear vs Binary Search – Time Complexity

| **Feature** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Time Complexity** | O(n) (worst case) | O(log n) (worst case) |
| **Best Case** | O(1) | O(1) (middle element) |
| **Data Type** | Works on unsorted data | Requires sorted data |
| **Implementation** | Simple to implement | Requires extra steps (sort) |

**Which Algorithm is More Suitable for Your Platform (Inventory)?**

**Binary Search is more suitable if the data is sorted, because:**

* It’s **much faster** than linear search as the dataset grows.
* Scales well for **large inventories** (e.g., in e-commerce platforms).

**Use Linear Search when:**

* The data is **unsorted** and small.
* Quick implementation is more important than speed.