**Big O Notation**

**Big O notation** is used to describe the **upper bound** of an algorithm’s runtime as a function of the input size (denoted by *n*). It provides a way to evaluate the **efficiency** of algorithms in terms of **time** and **space**.

**Common Complexities:**

* **O(1):** Constant time (e.g., accessing an element in an array).
* **O(log n):** Logarithmic time (e.g., binary search).
* **O(n):** Linear time (e.g., linear search).
* **O(n log n):** Linearithmic time (e.g., merge sort).
* **O(n²):** Quadratic time (e.g., nested loops).

**Search Operation Scenarios**

* **Best Case:** The item is found at the first position (O(1) for linear search).
* **Average Case:** The item is in the middle or somewhere not at the start (O(n/2) ≈ O(n) for linear search).
* **Worst Case:** The item is not in the list, or at the last position (O(n) for linear search, O(log n) for binary search).

**Time Complexity**

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

**Which is Better for an E-commerce Platform?**

* **Linear Search:**
  + Pros: Simple, works on unsorted data.
  + Cons: Inefficient on large datasets (e.g., millions of products).
* **Binary Search:**
  + Pros: Very fast on sorted datasets.
  + Cons: Requires maintaining a sorted list or using a sorted index (overhead in insertions/deletions).

**Output:**

Linear Search Result:

Product ID: 3, Name: Shoes, Category: Footwear

Binary Search Result:

Product ID: 3, Name: Shoes, Category: Footwear

**Recommendation:**

For an e-commerce platform with potentially **millions of products**, **binary search (O(log n))** is significantly more efficient. In real-world systems, search is often handled by optimized data structures or search engines (e.g., **Elasticsearch**, **Lucene**), but algorithmically, binary search is the preferred method when searching sorted in-memory data.