## **What’s Big O Notation?**

Big O notation is like a speedometer for algorithms. It tells us how slow or fast an algorithm gets as we throw more data at it, like more products in our store. It focuses on the worst-case scenario to give us a clear idea of performance, ignoring small details like exact counts of operations.

## **How Do Our Search Methods Perform?**

### **Linear Search**

This is like checking every product on the shelf one by one until you find the right one.

* **Best Case**: You’re super lucky, and the product is the first one you check. Takes just one look (O(1)).
* **Average Case**: On average, you’ll need to check about half the products, so it scales with the number of products (O(n)).
* **Worst Case**: The product is at the very end, or it’s not there at all, so you check every single one (O(n)).

### **Binary Search**

This is like playing a smart guessing game with a sorted list. You keep splitting the list in half to narrow down where the product is.

* **Best Case**: You nail it on the first guess, finding the product right in the middle (O(1)).
* **Average Case**: You keep halving the list, so it takes about log n steps, where n is the number of products (O(log n)).
* **Worst Case**: Even if the product is at the edge or missing, you still only need log n steps (O(log n)).

## 

## 

## **Comparing the Two Searches**

### **Linear Search**

* **Speed**: Takes O(n) time for both average and worst cases, meaning it gets slower as your product list grows.
* **Memory**: Uses almost no extra space (O(1)), just checks products one by one.
* **Pros**: Super simple and works even if products aren’t sorted.
* **Cons**: Painfully slow for a big store with thousands of products.

### **Binary Search**

* **Speed**: Way faster at O(log n) for average and worst cases, so it stays quick even with tons of products.
* **Memory**: Also uses minimal space (O(1)) in our version.
* **Pros**: Blazing fast for large catalogs.
* **Cons**: Needs products sorted first, which might take O(n log n) effort upfront or when adding new items.

## **Which One’s Better for Our E-commerce Platform?**

For our online store, binary search is the clear winner. With a big product catalog, we need searches to be snappy to keep customers happy. Binary search’s O(log n) speed means it can handle thousands of products without breaking a sweat. Since product IDs are unique and don’t change much, we can keep them sorted by ID, making the initial sorting cost worth it for fast searches. Linear search’s O(n) speed just can’t keep up—it’d make customers wait too long. If our product list changes a lot, we could use a sorted structure like a balanced binary search tree to make binary search even smoother.