| ***Exercise 2: E-commerce Platform Search Function*** |
| --- |

## **Understand Asymptotic Notation**

## **Asymptotic Analysis?**

**Asymptotic analysis** is the process of describing the behavior of an algorithm as the input size n approaches infinity. It is a fundamental tool in the study of algorithms because it allows developers and computer scientists to evaluate the efficiency and scalability of algorithms **independent of machine-specific constants and implementation details**.

Rather than calculating exact run times, we measure how fast the runtime (or space) grows with input size. This analysis is especially useful when comparing multiple algorithms or choosing data structures for large-scale systems, like search engines or e-commerce platforms.

## **Big-O Notation?**

**Big-O Notation (O-notation)** provides an **upper bound** on the growth rate of a function. If an algorithm has a time complexity of f(n) = O(g(n)), it means that, for **sufficiently large values of n**, the function f(n) grows at most as fast as some constant multiple of g(n).

Are there cases other than worst case ?

**Yes, but performance is generally measured in worst case rather than best, or average case. It's an engineering convention to test the ready product in their worst so they perform well in average cases.**

## **Class Setup ->**

The following class was implemented to model a product:

class Product {

private String productId;

private String productName;

private String category;

// Constructors, Getters, and toString()

}

**Implementation**

**SOURCE CODE : *https://github.com/AnujMishra-0/WeekWise-Learning-Solutions-Digital-Nurture-4.0-JavaFSE-2025/blob/main/Week%201%20Algorithms\_Data%20Structures/Exercise%202%3A%20E-commerce%20Platform%20Search%20Function/code(s)/ECommerceSearch.java***

### **🔹 Linear Search:**

public Product linearSearch(String searchValue, String field) →

It iterates over the array productsArray and compares the target value based on the given field (productId, productName, or category).

### **🔹 Binary Search:**

public Product binarySearch(String searchValue, String field) →

It uses a **pre-sorted array** (productsSortedById, productsSortedByName, or productsSortedByCategory) and performs binary search based on the specified field.

## **Analysis**

| **Search Type** | **Time Complexity** |  |  |
| --- | --- | --- | --- |
| **Linear Search** | O(n) |  |  |
| **Binary Search** | O(log n) in one search query |  |  |

### **🔹 Which is More Suitable?**

For an **e-commerce platform**:

**Binary search is much more efficient** but comes with a key requirement: the data must be **sorted**. Instead of checking every element, it smartly goes to the middle of the array, checks whether the target is on the left or right half, and repeats this process — effectively halving the search space each time.In terms of time complexity:

* **Best case** is O(1) — if the middle element is the match.
* **Worst and average cases** are both O(log n), since we divide the problem in half each step.

So binary search is dramatically faster for large datasets. To put it in perspective, even for a million products, it would take at most ~20 comparisons(open source study).

However, the downside is that it requires **sorted arrays** — and maintaining that sorted order can be costly if the dataset changes frequently.

**Linear search is the simplest approach**: it starts from the beginning of the array and checks each product one by one until it either finds the match or reaches the end. It's intuitive and works on *any* dataset, even if it's unsorted. From a performance perspective:

* **Best case** is O(1) — for example, if the target product is the first item.
* **Worst case** is O(n) — the product is either at the end or not present at all.
* **Average case** is also O(n) — because on average, we might have to scan about half the list.

It uses **constant space**, so O(1) space complexity.

But while it's easy to implement and flexible (no need to sort the data), it becomes inefficient when the list grows large. That’s why, for platforms dealing with thousands of products, it’s not ideal for customer-facing search.

LINEAR SEARCH IMPLEMENTATION 

BINARY SEARCH IMPLEMENTATION USING COMPARATORS’ SORTING