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

**1. Understanding Asymptotic Notation**

When analyzing algorithms, we often use **Big O notation** (also called *asymptotic notation*) to describe how the running time or space requirements grow as the size of the input increases. This helps developers and engineers compare algorithms and choose the one that performs best, especially on large datasets.

**Big O** expresses the *upper bound* of an algorithm’s time or space complexity — how slow it could get in the worst case. This ensures that systems will still perform acceptably even under heavy load.

**Search Operations: Best, Average, and Worst Cases**

For **search algorithms**, performance depends on how quickly the target element is found (or confirmed missing):

| **Scenario** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Best Case | O(1) — first element match | O(1) — middle element match |
| Average Case | O(n/2) → O(n) | O(log n) |
| Worst Case | O(n) — element not present | O(log n) |

**2. Setup**

In this exercise, I created a class called Product to represent items in an e-commerce platform’s catalog. Each product has the following attributes:

* productId: unique identifier for the product.
* productName: name of the product.
* category: product category (e.g., electronics, books, clothing).

The project also includes a main class, ProductSearch, to perform search operations on the product list.

**3. Implementation**

**Search Algorithms**

Two types of search algorithms were implemented:

* **Linear Search**:
  + Scans each product in the array one by one.
  + Does not require any special ordering of products.
* **Binary Search**:
  + Operates on a *sorted* array of products (sorted by productId).
  + Uses a divide-and-conquer approach to quickly eliminate half of the search space at each step.

**Data Storage**

* Products are stored in a **plain array (Product[])**.
* The array is sorted before applying **binary search** using Arrays.sort() and a Comparator<Product>.

**4. Analysis**

**Time Complexity Comparison**

| **Algorithm** | **Time Complexity** | **Notes** |
| --- | --- | --- |
| Linear Search | O(n) | No sorting needed; slow on large datasets |
| Binary Search | O(log n) | Requires sorted data; much faster on large datasets |

**Suitability for E-commerce Platform**

For a **small number of products** (few dozens or hundreds), linear search is acceptable because its simplicity outweighs the small performance hit.

However, for **large-scale e-commerce platforms** with thousands or millions of products, **binary search** (or even more advanced structures like hash tables or search trees) is far more suitable:

* Much faster lookups (O(log n) vs O(n)).
* Scales well as the catalog grows.
* Sorting overhead is negligible when products are sorted once and searched many times.

**Conclusion**

In this exercise, I implemented a simple search functionality for an e-commerce platform using both linear and binary search algorithms. The results clearly show that **binary search is superior** in terms of speed when the product list is large.

In a real-world e-commerce platform, we would likely combine **sorted structures** and/or **hash-based indexes** (e.g., using databases or search engines like Elasticsearch) to enable even faster and more flexible search capabilities.