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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

Explanation of Big O notation and how it helps in analysing algorithms:

**Big O Notation** is a mathematical way to describe how the **time** or **space** needed by an algorithm grows **as the input size increases**. It gives us an upper bound on the growth rate, helping us analyze and compare algorithms efficiently.

1. Helps you **predict performance** before running the code.
2. Allows you to **compare multiple algorithms** regardless of hardware or programming language.
3. Useful for **choosing efficient algorithms**, especially for large inputs (e.g., thousands of products in an e-commerce app).

| **Big O** | **Name** | **Example** | **Performance** |
| --- | --- | --- | --- |
| O(1) | Constant Time | Accessing an element by index | Fastest |
| O(log n) | Logarithmic Time | Binary Search | Very Fast |
| O(n) | Linear Time | Linear Search, Loop over array | Moderate |
| O(n log n) | Log-linear Time | Merge Sort, Quick Sort (avg case) | Efficient |
| O(n²) | Quadratic Time | Bubble Sort, nested loops | Slower |
| O(2ⁿ) | Exponential Time | Brute force subset generation | Very Slow |

Analysis of time and space complexities of linear and binary search algorithms:

**Best Case**

* In **Linear Search**, the best-case scenario occurs when the target element is the **first element** in the list, resulting in only **one comparison** (Time Complexity: **O(1)**).
* In **Binary Search**, the best case occurs when the target element is found **exactly at the middle** of the sorted list in the **first comparison** (Time Complexity: **O(1)**).

**Average Case**

* In **Linear Search**, on average, the target element is found **somewhere in the middle** of the list, requiring **n/2 comparisons** (Time Complexity: **O(n)**).
* In **Binary Search**, the average case requires **log₂(n)** comparisons, as the list is divided in half with each step (Time Complexity: **O(log n)**).

**Worst Case**

* In **Linear Search**, the worst case occurs when the target element is the **last element** in the list or **not present at all**, requiring a check of **every element** (Time Complexity: **O(n)**).
* In **Binary Search**, the worst case occurs when the element is **not found**, requiring repeated division of the list until the search space is empty (Time Complexity: **O(log n)**).

Best Approach:

**Binary Search** is more suitable for our e-commerce platform **when the product data is sorted** (e.g., by productId, productName, or category), and we want **fast and scalable search performance**

**Faster Search Time (O(log n))**

* Binary Search reduces the time complexity significantly compared to Linear Search (O(n)), especially useful when the number of products is large.

**Efficient for Frequently Queried Data**

* In an e-commerce platform, where users search products frequently, fast response time is crucial for good user experience.

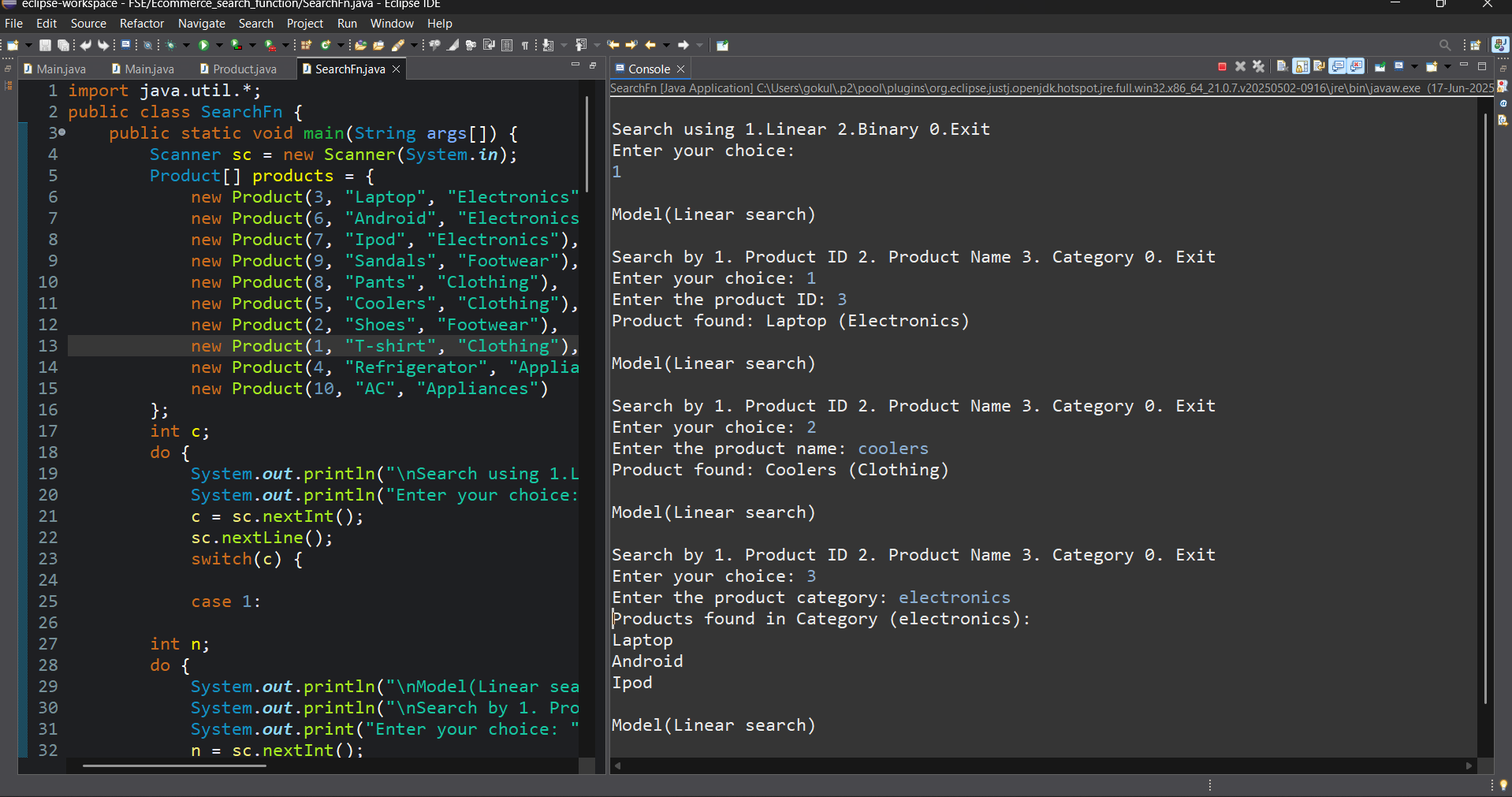
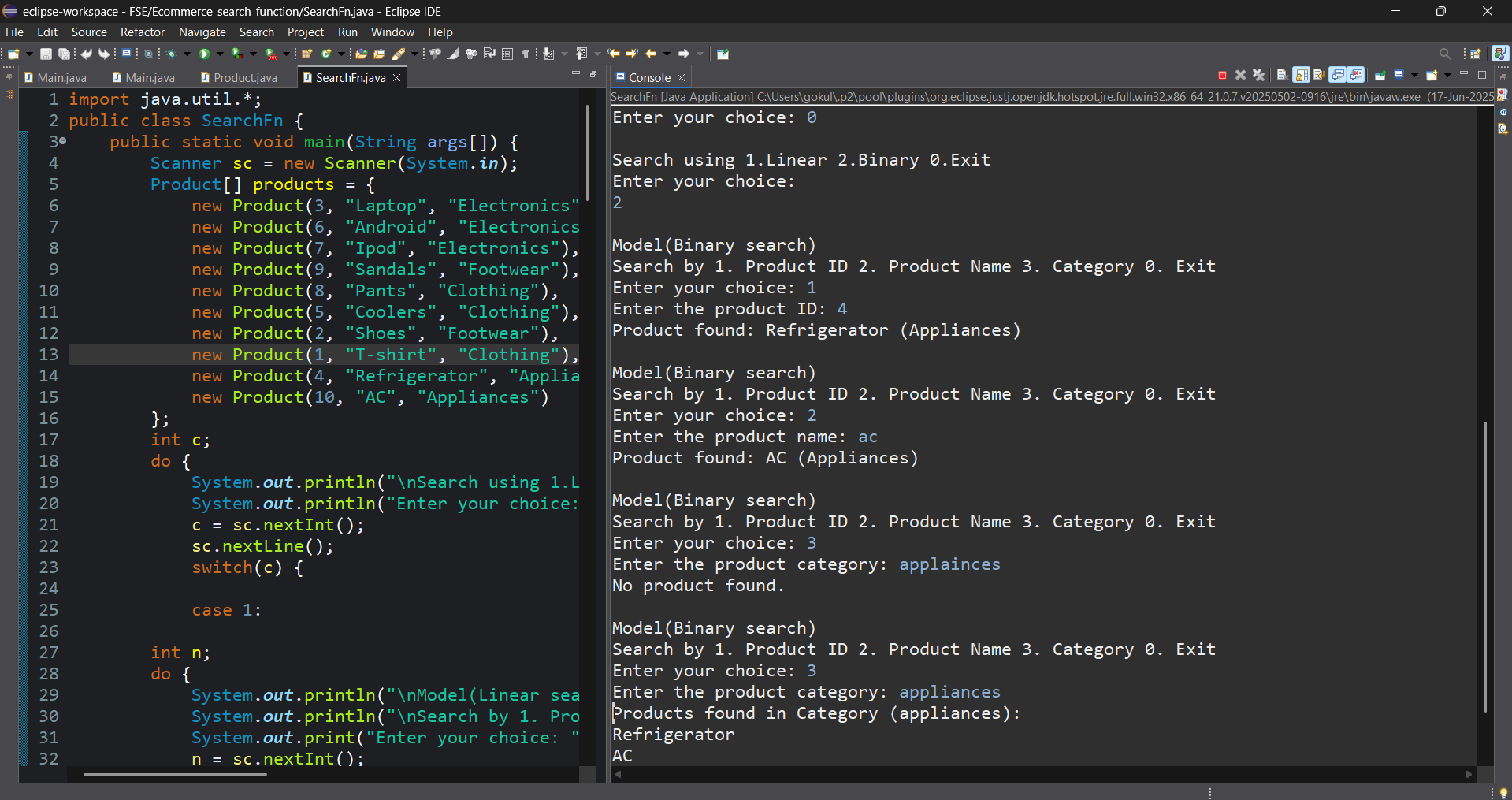
**Ideal for Sorted Data Structures**

* E-commerce platforms usually maintain sorted indexes (e.g., sorted by name or category) which makes Binary Search applicable.

Where as Linear search can be used when dataset is small and unsorted.

Output:

Linear search:

Binary Search: