**WEEK 1: Algorithms and Data Structures (Solutions)**

**Exercise 1: 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.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.

**Answer:**

**Big O Notation** is a mathematical way to describe the time or space complexity of an algorithm as the input size grows. It helps estimate **performance** regardless of hardware or language.

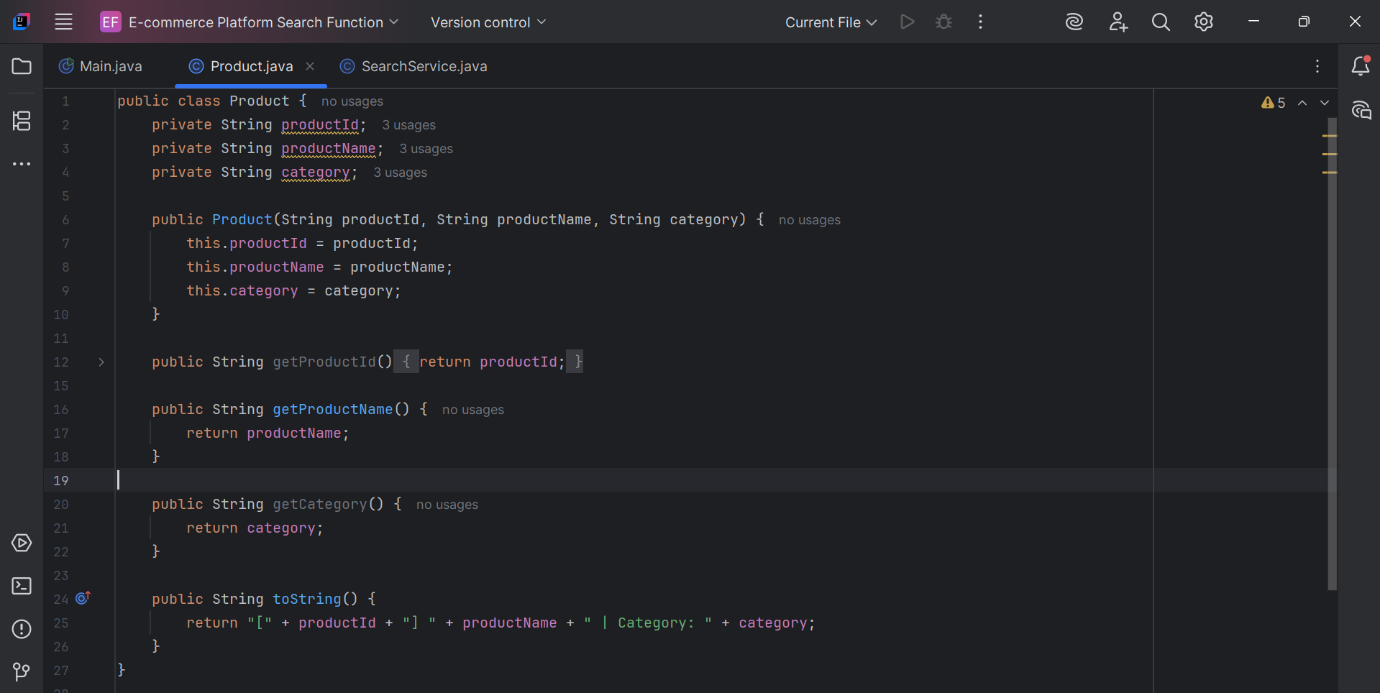
| **Notation** | **Description** | **Example** |
| --- | --- | --- |
| O(1) | Constant time | Accessing an array element by index |
| O(n) | Linear time | Linear search |
| O(log n) | Logarithmic time | Binary search |
| O(n²) | Quadratic time | Nested loops |

* + Describe the best, average, and worst-case scenarios for search operations.

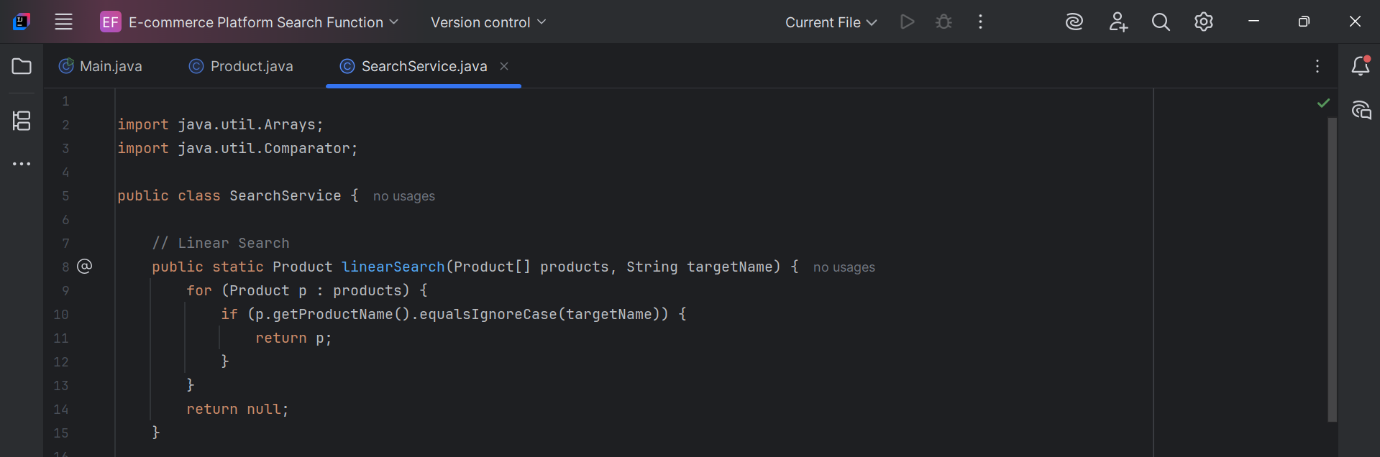
**Answer:**

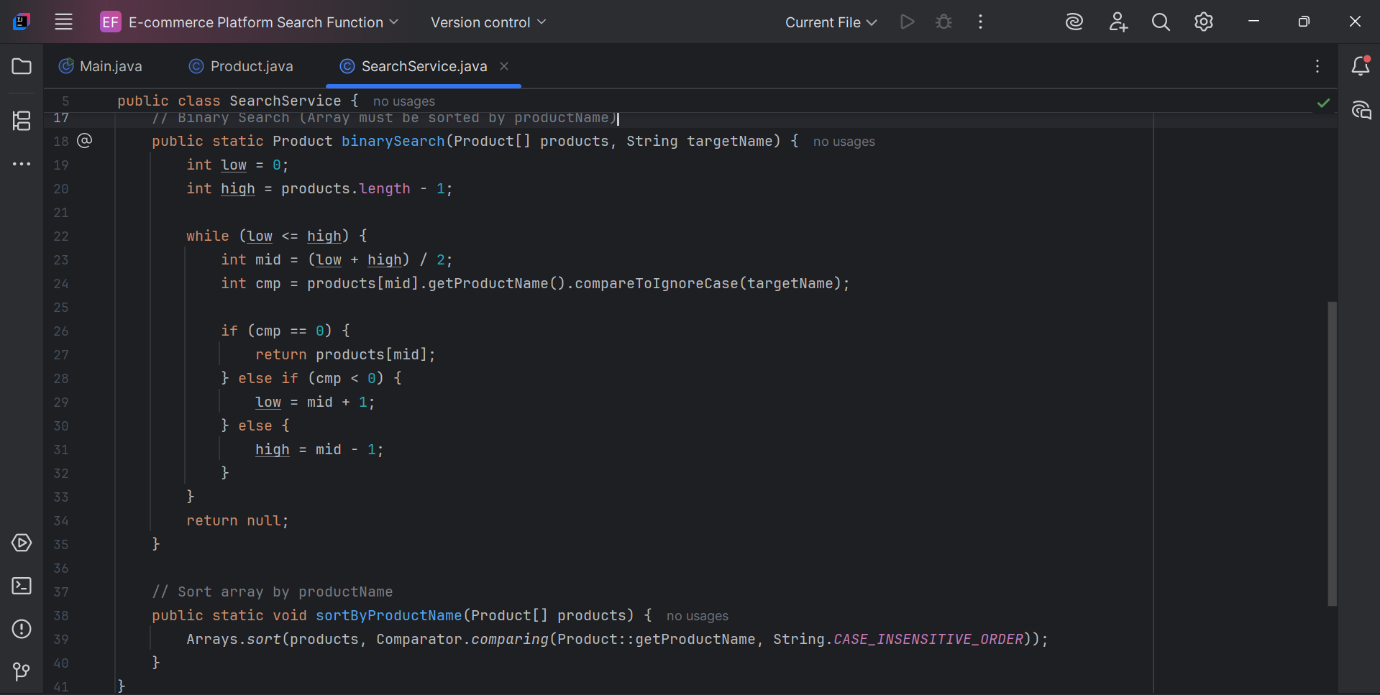
|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Best Case | Average Case | Worst Case |
| Linear Search | |  | | --- | |  |  |  | | --- | | O(1) – first match | | O(n/2) ≈ O(n) | O(n) – last or not found |
| Binary Search | |  | | --- | |  |  |  | | --- | | O(1) – first match | | |  | | --- | |  |  |  | | --- | | O(log n) | | O(log n) |

1. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.

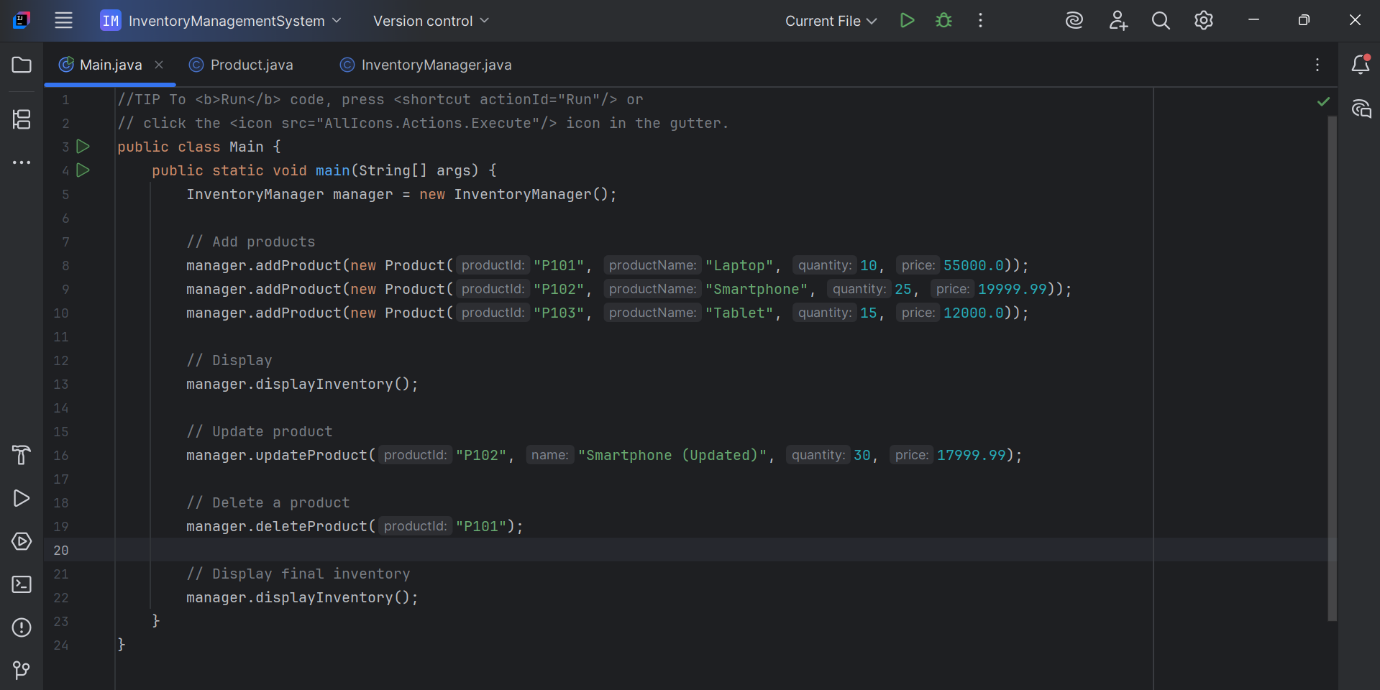


1. **Implementation:**
   * Implement linear search and binary search algorithms.

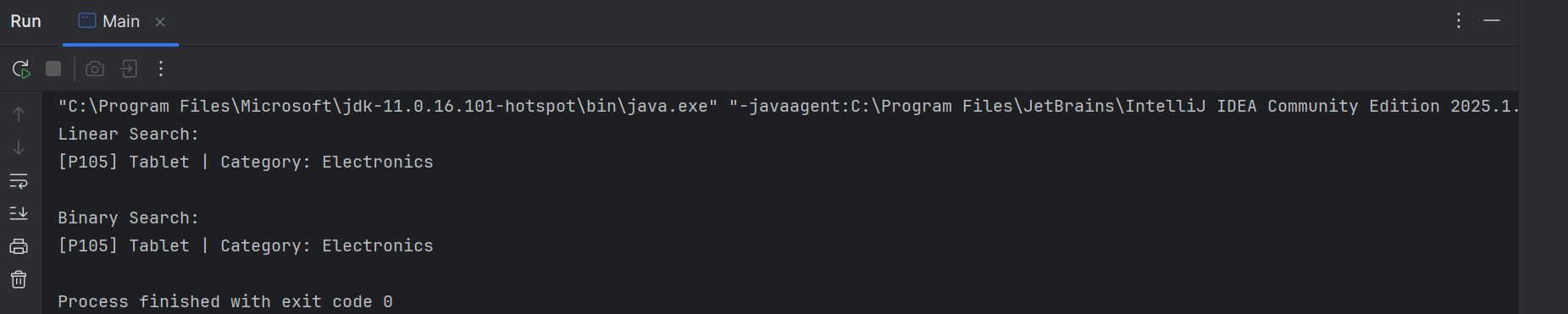




* + Store products in an array for linear search and a sorted array for binary search.



OUTPUT



1. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.

**Answer:**

Time Complexity Comparison:

| **Operation** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Time Complexity | O(n) | O(log n) |
| Requirements | No sort needed | Sorted array required |
| Suitable for | Small/unsorted datasets | Large/sorted datasets |

* + Discuss which algorithm is more suitable for your platform and why.

**Answer:**

* **Linear Search** is simple and works on unsorted arrays, but is slower (O(n)).
* **Binary Search** is much faster (O(log n)), but requires a sorted array.
* For an **E-commerce platform** with frequent searches:
* Maintain a sorted collection (e.g., using TreeMap or sort once then cache).
* Use binary search or index structures (like HashMap or Tries for strings) for best performance.

============

**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

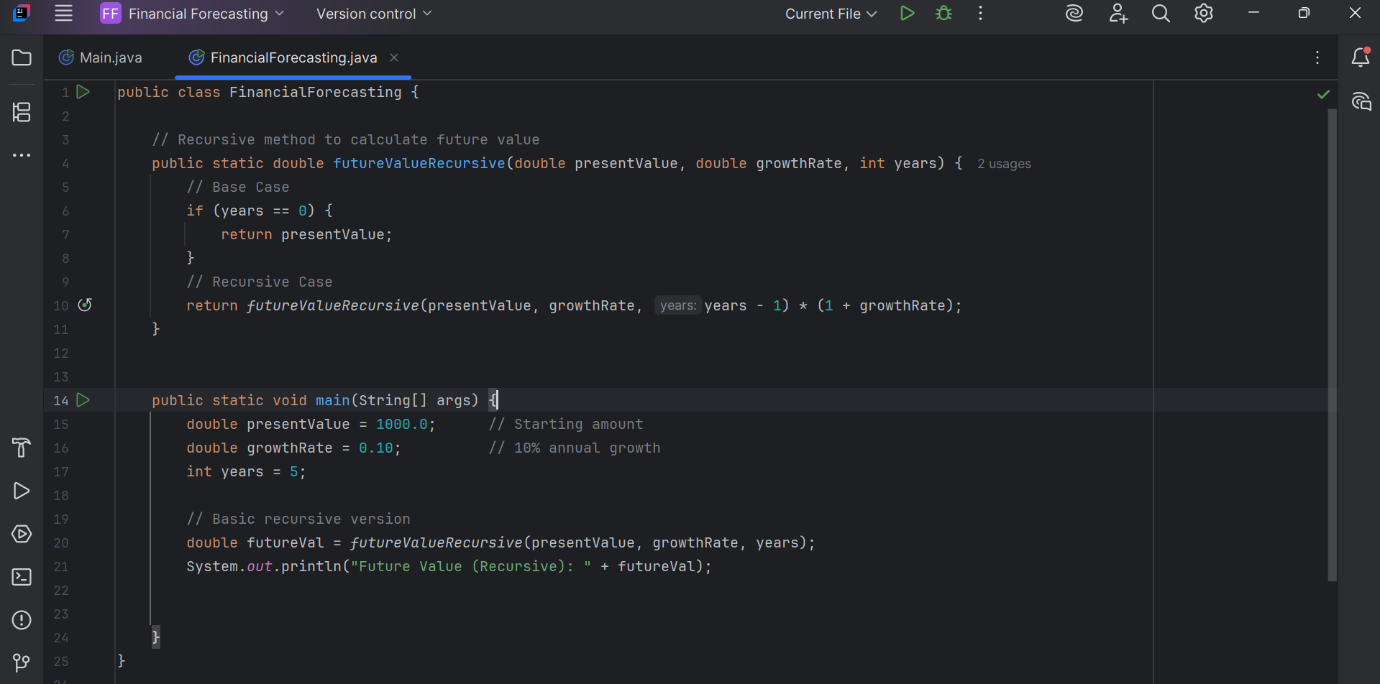
**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.

**Answer:**

* **Recursion** is a method where a function calls itself to solve a problem by breaking it down into smaller sub-problems.
* This approach simplifies complex problems by applying the same logic repeatedly. Each recursive call moves toward a base case, which ends the recursion. In financial forecasting, recursion can model repeated processes like compound interest or growth trends over time.
* By defining a clear base case (e.g., year 0) and a recursive rule (e.g., future value = current value × growth rate), one can simulate future values without manually iterating, leading to cleaner, more readable, and modular code.

1. **Setup:**
   * Create a method to calculate the future value using a recursive approach.

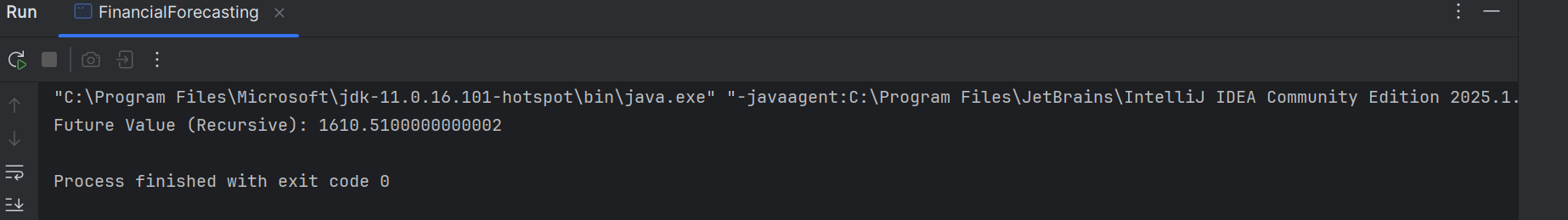


1. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates

**Recurrence Relation:**

**FV(n) = FV(n-1) \* (1 + r)**

**Base Case: FV(0) = Present Value**



1. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.

**Answer:**

Time Complexity:

* + - Basic Recursive Approach:
* **Time: O(n)** — Each recursive call computes one year.
* **Space: O(n)** — Call stack depth is n.
  + Explain how to optimize the recursive solution to avoid excessive computation.

**Answer:**

Optimization:

* **Memoization** — Avoid redundant calculations.
* **Iterative Approach** (if recursion is too deep).
* **Use BigDecimal** if high precision is required.

============