# E-COMMERCE PLATFORM SEARCH FUNCTION

Question:

**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.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.

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

# 1. Understanding Asymptotic Notation

## Big O Notation

Big O notation describes the time complexity of an algorithm — how the runtime increases as the input size grows. It helps us compare algorithms without implementation or hardware bias.  
  
Common complexities:  
- O(1): Constant time  
- O(n): Linear time  
- O(log n): Logarithmic time  
- O(n²): Quadratic time

## Best, Average, and Worst Case for Search

Linear Search:  
- Best Case: O(1) – match at first position  
- Average Case: O(n/2) -> O(n)  
- Worst Case: O(n)  
  
Binary Search:  
- Best Case: O(1) – match at middle  
- Average Case: O(log n)  
- Worst Case: O(log n)  
  
\* Binary Search requires sorted data.

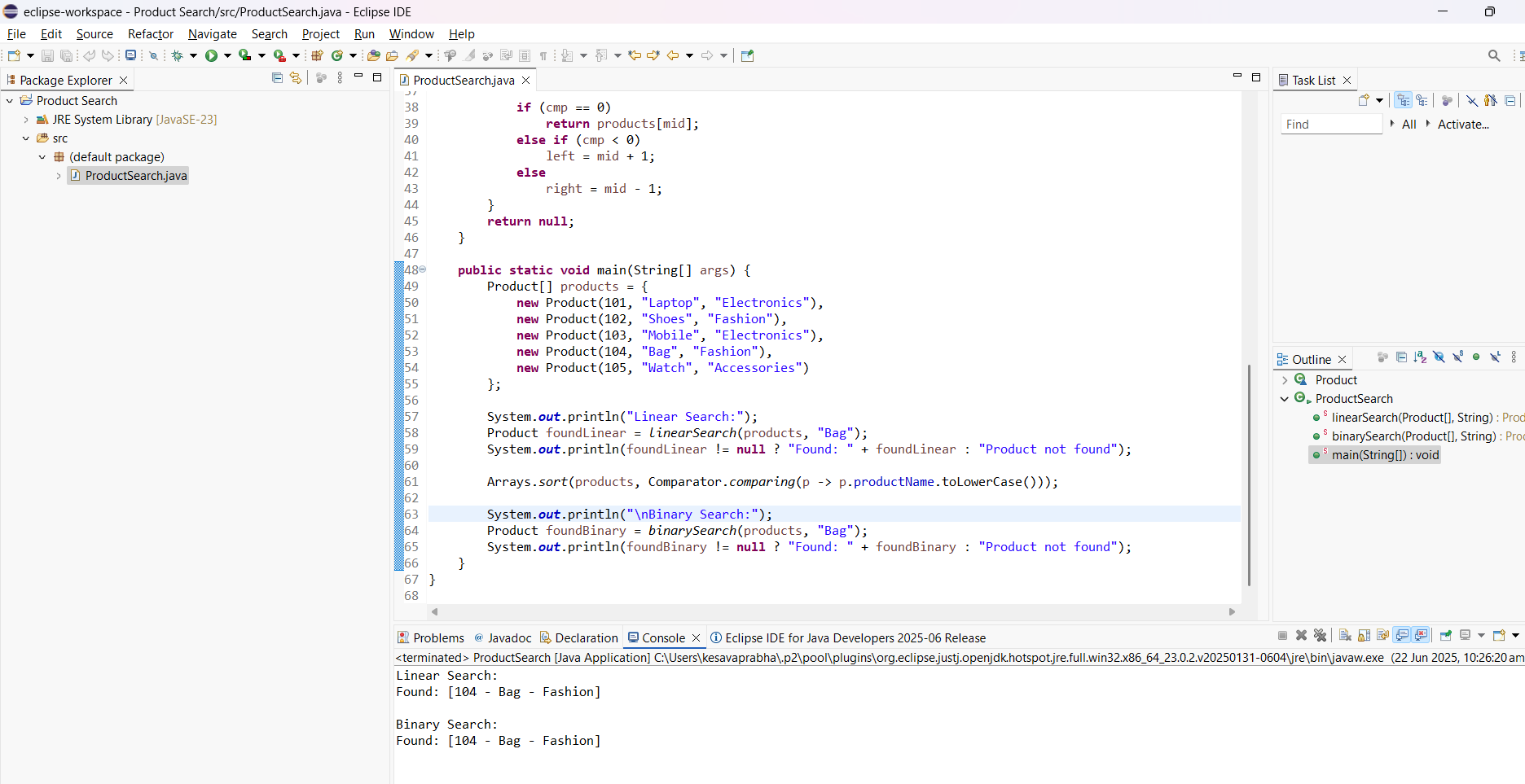
# 2. Setup: Product Class

The Product class contains attributes used for searching such as productId, productName, and category.  
  
public class Product {  
 int productId;  
 String productName;  
 String category;  
  
 public Product(int productId, String productName, String category) {  
 this.productId = productId;  
 this.productName = productName;  
 this.category = category;  
 }  
  
 @Override  
 public String toString() {  
 return "[" + productId + " - " + productName + " - " + category + "]";  
 }  
}

# 3. Implementation: Linear and Binary Search

Both Linear and Binary search methods are implemented. Linear search goes through each element, while Binary search uses divide-and-conquer on a sorted array.

Output:



# 4. Analysis: Time Complexity & Suitability

Time Complexity:  
Linear Search: O(n)  
Binary Search: O(log n)  
  
Suitability:  
- Use Linear Search for small or unsorted datasets.  
- Use Binary Search for large, sorted datasets for better performance.

# FINANCIAL FORECASTING

**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.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

# 1. Understanding Recursive Algorithms

Recursion is a programming technique where a method calls itself to solve the same problem. It simplifies complex problems like tree traversal, mathematical computations (e.g., factorial, Fibonacci), and divide-and-conquer problems.

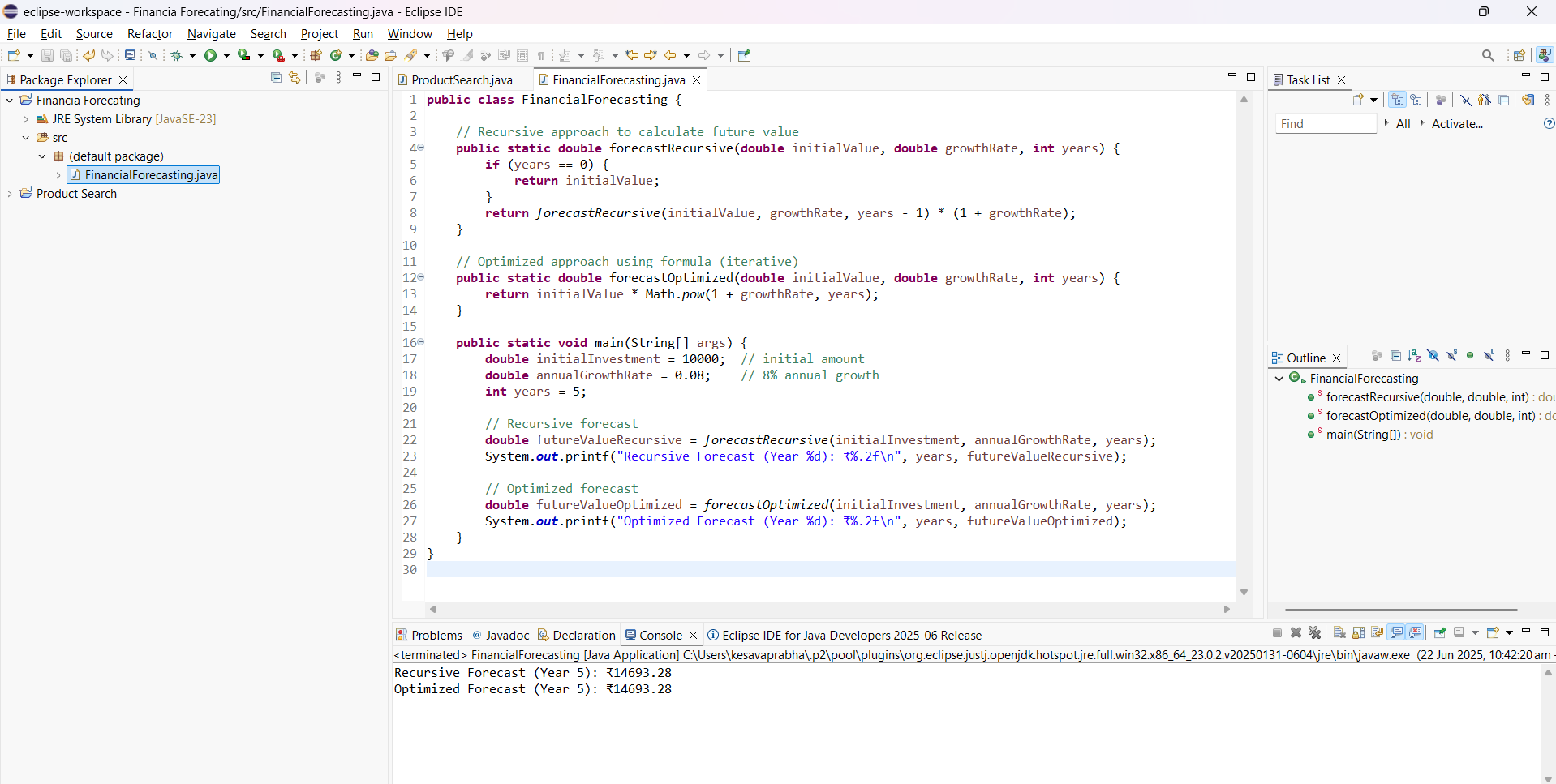
Advantages of recursion:  
- Cleaner, more intuitive code for problems that naturally fit a recursive pattern.  
- Reduces the need for loop constructs in some cases.

# 2. Setup

To forecast future financial values recursively, we define a method that takes:  
- initial value  
- growth rate  
- number of periods  
  
The method recursively calculates the future value using the formula:  
FV(n) = FV(n-1) \* (1 + growthRate)

# 3. Implementation: Recursive Forecasting public static double forecastValue(double initialValue, double growthRate, int years) { if (years == 0) { return initialValue; } return forecastValue(initialValue, growthRate, years - 1) \* (1 + growthRate); }

Output:



# 4. Analysis

Time Complexity:  
The recursive algorithm has a time complexity of O(n), where n is the number of years, because it makes a single recursive call per year.  
  
Optimization:  
- Use memoization or convert to an iterative approach to avoid redundant calls.  
- For financial computations, an iterative or closed-form solution is more efficient:  
 FV = initialValue \* Math.pow(1 + growthRate, years)