**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.

**Solution-**

**1. Asymptotic Notation**

Big O Notation - It is a notation used to denote the time or space complexity of any program.

For example, in case of time complexity it tells how the time will vary with respect to input.

Analyzing Algorithms - O(n) tells the time vary with respect to the size of input while O(n^2) tells the time vary with respect to the square of size of input. In this way we can analyze difference between algorithms.

Search operation – Mainly linear and binary search operations are used.

For linear search: Best O(1), Average O(n), Worst O(n)

For binary search: Best O(1), Average O(log n), Worst O(log n) {The search space should be sorted}

**2. Setup**

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;

}

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

**3. Implementation**

We will search on the basis of productId

**Linear Search-**

public static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

**Binary Search (Array must be sorted)-**

public static Product binarySearch(Product[] products, int targetId) {

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (products[mid].productId == targetId) {

return products[mid];

} else if (products[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

**Main Function-**

public class SearchDemo {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(120, "Smartphone", "Electronics"),

new Product(150, "Book", "Education"),

new Product(205, "Shoes", "Fashion")

};

// Linear Search

Product result1 = linearSearch(products, 150);

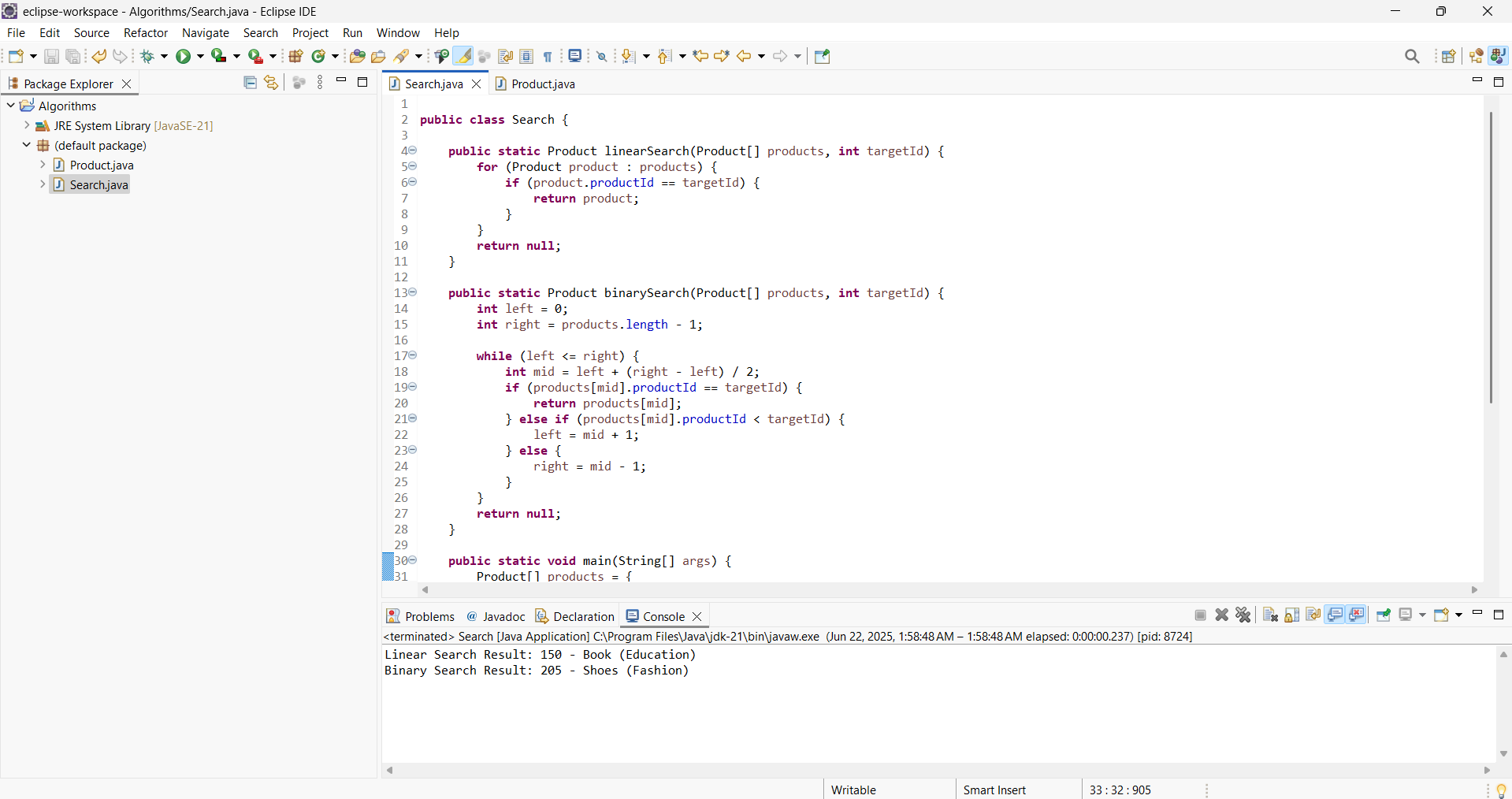
System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));

// Binary Search

Product result2 = binarySearch(products, 150);

System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));

}

**Output-**

**4. Analysis**

Time complexity of linear search is best for unsorted data. O(n)

For sorted data binary search is best. O(log n)

For our platform binary search is best as product catalogue may be large and we can easily sort the products through product\_id.

If the size of product catalogue is small we can also use linear search, not the best but still if size is small.

**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.

**Solution-**

**1. Recursive Algorithms**

Basically, a function calling itself is known as recursion.

Recursive Algorithms use recursion to minimize the problem by solving small instances of problem. A certain condition is used to stop recursion known as base condition.

It can simplify certain problems which depends on a sequence of solution. We can take example of divide and conquer algo using recursion. Due to the recursion stack the complexity of writing algo is reduced to a certain extent.

**2. Setup**

We will create a simple method to find compound interest where the principal investment amount can be taken as past data and the final amount is the future value of investment.

We will use the basic formula: F = P(1+r)^t

Where F is future value, P is principal, r is rate of interest per year, t is no. of years

**3. Implementation**

- futureValue is the recursive function.

public class Forecast {

public static double futureValue(double presentValue, double rate, int periods) {

// Base case

if (periods == 0) {

return presentValue;

}

return futureValue(presentValue \* (1 + rate), rate, periods - 1);

}

public static void main(String[] args) {

double P = 1000.0;

double rate = 0.05; // 5% growth rate

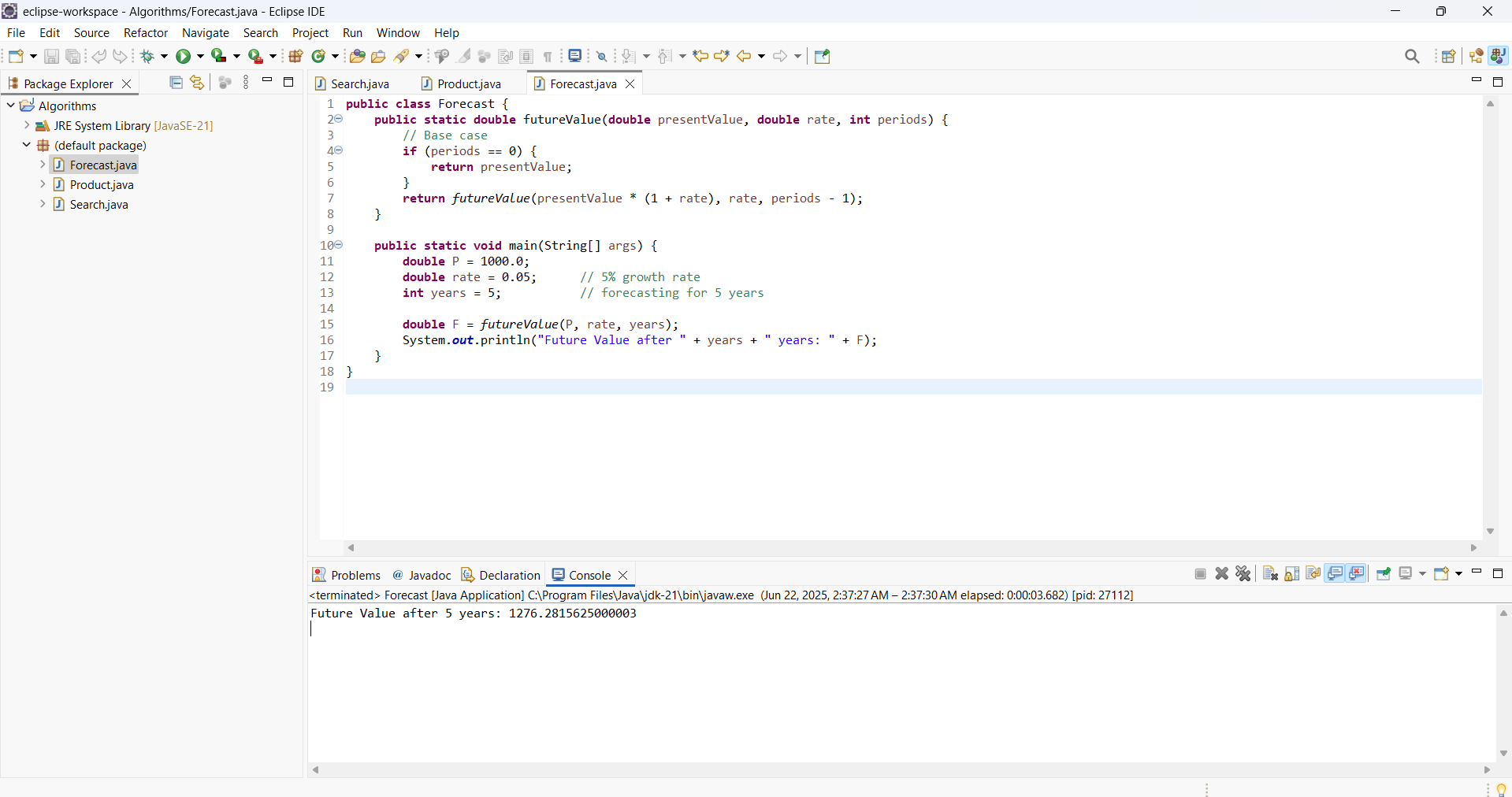
int years = 5; // forecasting for 5 years

double F = futureValue(P, rate, years);

System.out.println("Future Value after " + years + " years: " + F);

}

}

**Output-**

**4. Analysis**

**Time complexity:** O(t) ‘t’ is the number of years as ‘t’ calls will be made until the base condition as the above algo uses linear recursion.

**Optimization:** We can use memorization in case of overlapping problems as we use in DP problems mostly. It can reduce exponential time complexity to polynomial time. This can reduce the overall computations. In some cases iterative methods can also be used.