**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:**

SearchMethods.java

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** SearchMethods {

**static** **class** ProductDetails {

**int** productId;

String productName;

String category;

**public** ProductDetails(**int** id, String name, String category) {

**this**.productId = id;

**this**.productName = name;

**this**.category = category;

}

**public** String toString() {

**return** "The ID of the product is " + productId + ",Name of the product is " + productName + ",product belogs to " + category;

}

}

**public** **static** ProductDetails linearSearch(ProductDetails[] products, String searchElement) {

**int** count = 0;

**for** (**int** index = 0; index < products.length; index++) {

count++;

**if** (products[index].productName.equals(searchElement)) {

System.***out***.println("The number of times the loop rotated in linear search: " + count);

**return** products[index];

}

}

System.***out***.println("The number of times the loop rotated in linear search: " + count);

**return** **null**;

}

**public** **static** ProductDetails binarySearch(ProductDetails[] products, String searchElement) {

**int** start = 0, end = products.length - 1;

**int** c2 = 0;

**while** (start <= end) {

c2++;

**int** middle = start + (end - start) / 2;

**int** cmp = products[middle].productName.compareToIgnoreCase(searchElement);

**if** (cmp == 0) {

System.***out***.println("The number of times the loop rotated in binary search: " + c2);

**return** products[middle];

} **else** **if** (cmp < 0) {

start = middle + 1;

} **else** {

end = middle - 1;

}

}

System.***out***.println("The number of times the loop rotated in binary search: " + c2);

**return** **null**;

}

**public** **static** **void** sortProducts(ProductDetails[] products) {

Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName.toLowerCase()));

}

**public** **static** **void** main(String[] args) {

ProductDetails[] products = {

**new** ProductDetails(1, "product1", "clothes"),

**new** ProductDetails(2, "product2", "craft"),

**new** ProductDetails(3, "product3", "books"),

**new** ProductDetails(4, "Product4", "electronics")

};

ProductDetails result1 = *linearSearch*(products, "product3");

System.***out***.println("Linear Search Result: " + (result1 != **null** ? result1 : "Product not found"));

*sortProducts*(products);

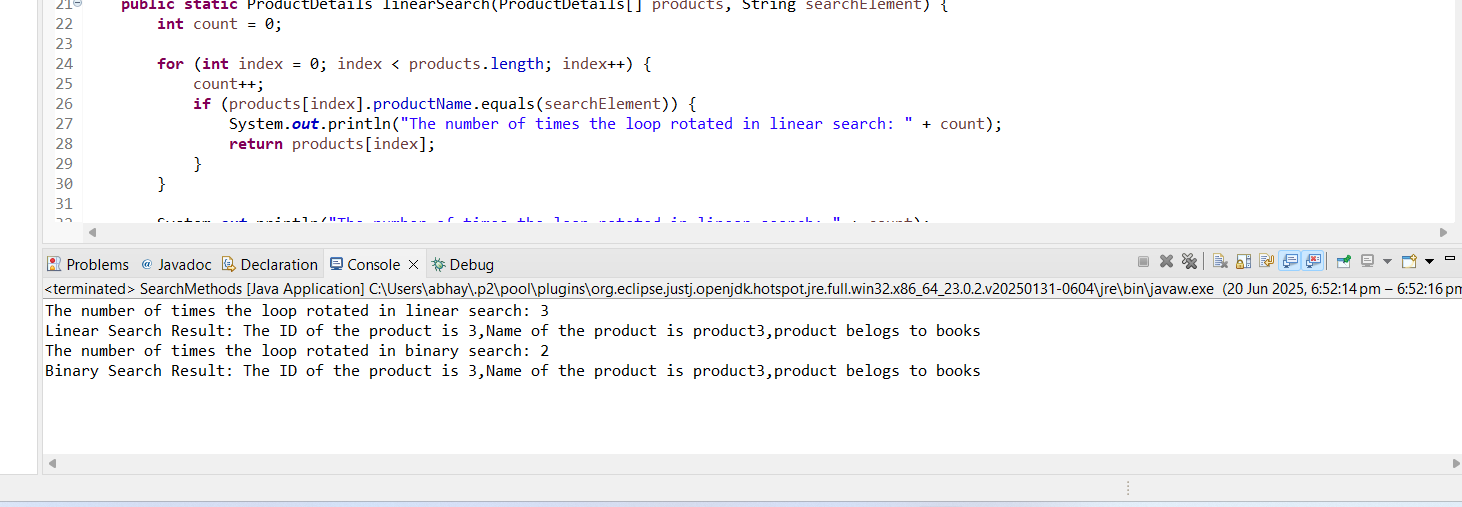
ProductDetails result2 = *binarySearch*(products, "product3");

System.***out***.println("Binary Search Result: " + (result2 != **null** ? result2 : "Product not found"));

}

}

Output:



**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:

**public** **class** FinancialRecursive {

**public** **static** **double** valueInFuture(**double** valueInPresent, **double** rateOfInterest, **int** noOfYears) {

**if** (noOfYears == 0) {

**return** valueInPresent;

}

**return** *valueInFuture*(valueInPresent \* (1 + rateOfInterest), rateOfInterest, noOfYears - 1);

}

**public** **static** **void** main(String[] args) {

**double** presentValue = 11345.0;

**double** rateOfInterest = 0.07;

**int** noOfyears = 10;

**double** FutureValue = *valueInFuture*(presentValue, rateOfInterest, noOfyears);

System.***out***.printf("Future Value after %d years: %.2f\n", noOfyears, FutureValue);

}

}

Outputs:

