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

**What is Big O Notation?**

**Big O Notation** describes the **performance or complexity** of an algorithm in terms of how it **scales** with input size n. It tells us how fast or slow an algorithm grows as input increases.

| **Case** | **Description** | **Example (Linear Search)** |
| --- | --- | --- |
| **Best** | Fastest time (ideal case) | O(1) — first element match |
| **Average** | Typical time over many inputs | O(n/2) ≈ O(n) |
| **Worst** | Slowest time (unlucky case) | O(n) — last element match or not found |

**Binary Search:**

* Works only on **sorted data**, **Bes**t: O(1),**Averag**e/Worst: O(log n)

**Step 2: Setup Product Class**

1.Product.java

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 "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

2.SearchService.java

import java.util.\*;

public class SearchService {

public static Product linearSearch(Product[] products, String targetName) {

for (Product product : products) {

if (product.productName.equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

public static Product binarySearch(Product[] products, String targetName) {

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(targetName);

if (cmp == 0)

return products[mid];

else if (cmp < 0)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

public static void sortProductsByName(Product[] products) {

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

}

}

3.Main.java

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Microwave", "Electronics"),

new Product(2, "Shirt", "Clothing"),

new Product(3, "Watch", "Accessories"),

new Product(4, "Book", "Education"),

new Product(5, "Phone", "Electronics")

};

Product result1 = SearchService.linearSearch(products, "Book");

System.out.println("Linear Search Result: " + result1);

SearchService.sortProductsByName(products);

Product result2 = SearchService.binarySearch(products, "Book");

System.out.println("Binary Search Result: " + result2);

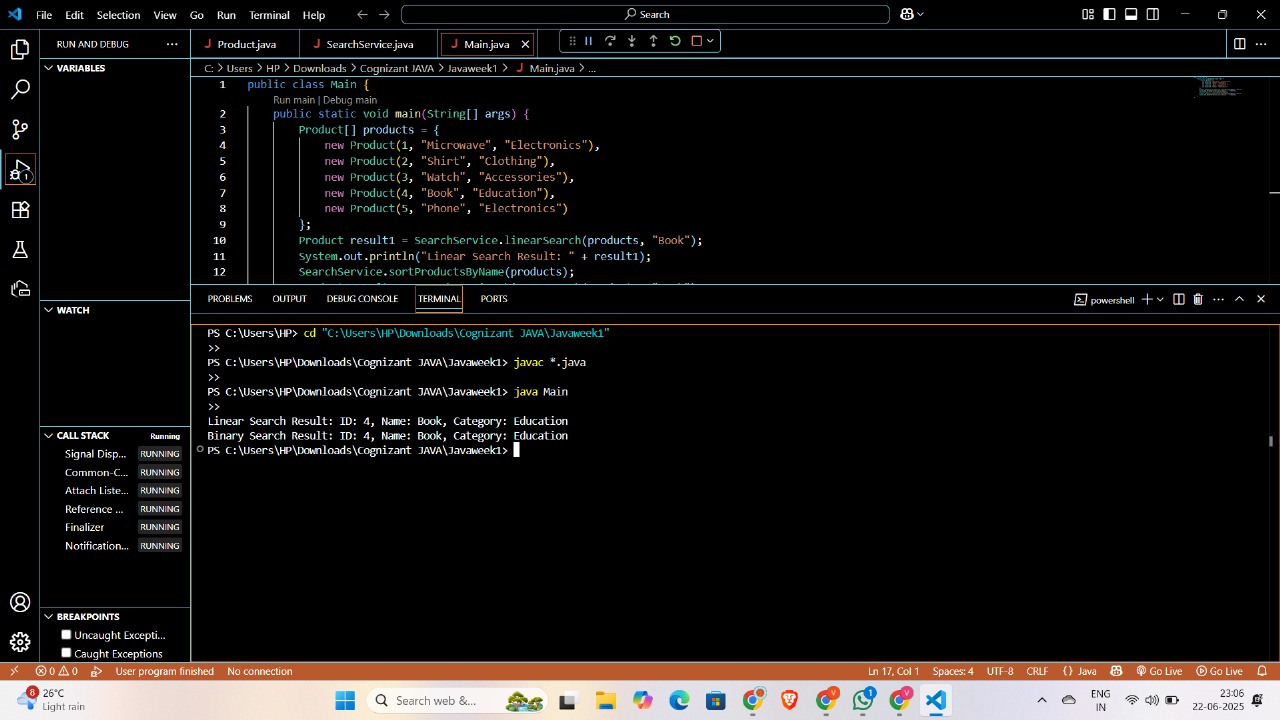
}

}

**Step 5: Analysis**

| **Feature** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Data Requirement** | Unsorted | Sorted |
| **Best Case** | O(1) | O(1) |
| **Average/Worst** | O(n) | O(log n) |
| **Implementation** | Simple | Requires sorting + more logic |
| **Flexibility** | Can be used on any data | Only on sorted data |

**Binary Search is more Efficient  
Output:**



**Exercise 7:**

**Financial Forecasting**

**Step 1: Understand Recursive Algorithms**

**What is Recursion?**

Recursion is when a function calls itself to solve a smaller version of a problem.

**Why is it useful?**  
It simplifies problems that have repeating patterns, like:

* Calculating compound interest
* Fibonacci numbers
* Tree traversal

**Example Idea for Financial Forecasting:**

If a value grows by a fixed percentage every year, then:

Where:

* FV(n) = value at year n
* r = growth rate (e.g., 0.10 for 10%)

**Step 2: Setup**

Create a class with a method to calculate future value recursively:

public class Forecast {

public static double futureValue(double currentValue, double growthRate, int years) {

if (years == 0) return currentValue; // base case

return futureValue(currentValue, growthRate, years - 1) \* (1 + growthRate);

}

}

**Step 3: Implementation**

Now, add a main method to test the recursive forecasting:

public class Main {

public static void main(String[] args) {

double initialValue = 30000;

double growthRate = 0.06;

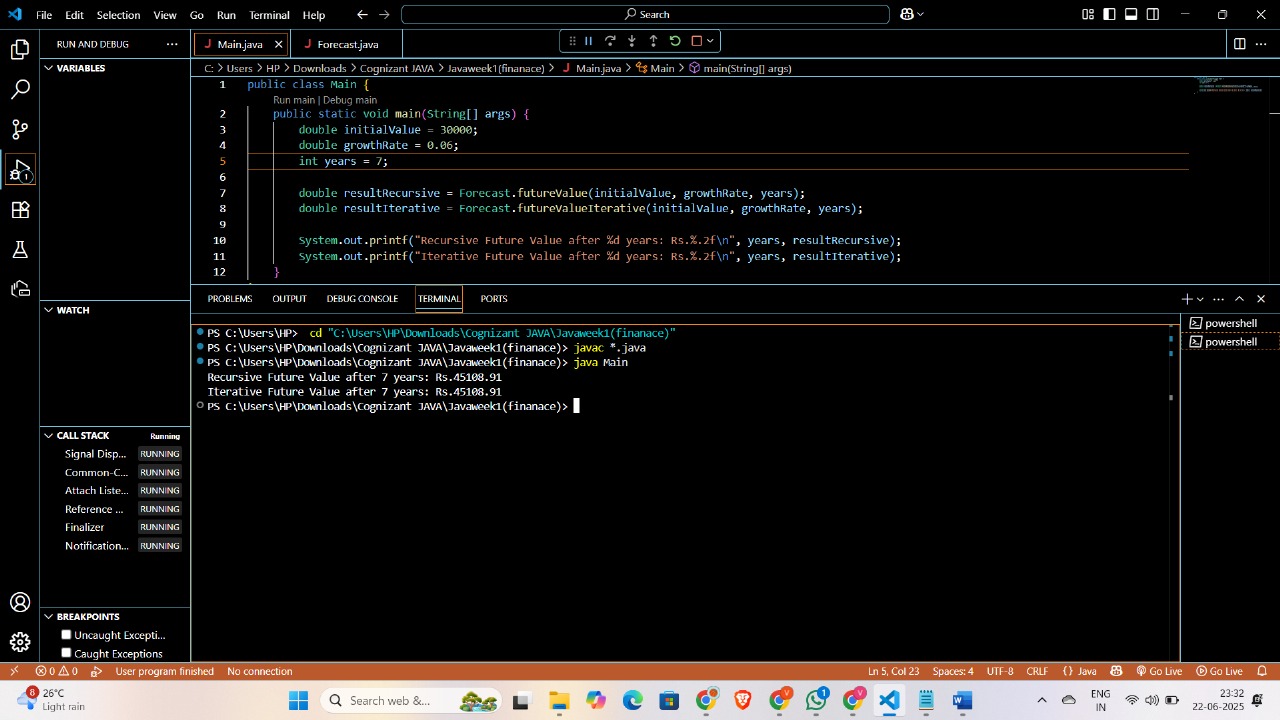
int years = 7;

double result = Forecast.futureValue(initialValue, growthRate, years);

System.out.printf("Future value after %d years: ₹%.2f\n", years, result);

}

}



Output:

Recursive Future Value after 7 years: Rs.45108.91

Iterative Future Value after 7 years: Rs. 45108.91  
**Optimization:**

Use iterative approach