# Week1\_DSA\_HandsOn

## Exercise 2: E-commerce Platform Search Function

UNDERSTANDING ASYMPTOTIC NOTATION:

**What is Big O Notation?**

Big O notation is a way to describe how **fast or slow** an algorithm is, based on the size of the input (n).  
It tells us how the time or space **grows** as data grows, without worrying about exact numbers.

* **Example:**
* If an algorithm takes **2 seconds for 100 items** and **4 seconds for 200 items**, it is said to have **O(n)** time — linear growth.
* **Why is Big O Important?**
* It helps us compare different algorithms.
* We can **predict performance** and **choose the best approach** before coding.
* It helps avoid slow solutions that don’t scale when data gets bigger.

**Best, Average, and Worst-Case Scenarios for Search**

* **Linear Search**
* **Best Case:** O(1) → First item matches.
* **Average Case:** O(n/2) → On average, it checks half the list.
* **Worst Case:** O(n) → Target is at the end or not found at all.

**Binary Search**

* **Best Case:** O(1) → Middle item matches.
* **Average Case:** O(log n) → List is halved every step.
* **Worst Case:** O(log n) → Still halves until 1 element is left.

**JAVA CODE:**

import java.util.Arrays;  
  
public class ECommerceSearch {  
  
 static class Product {  
 int productId;  
 String productName;  
 String category;  
  
 Product(int productId, String productName, String category) {  
 this.productId = productId;  
 this.productName = productName;  
 this.category = category;  
 }  
  
 public String toString() {  
 return productId + " - " + productName + " (" + category + ")";  
 }  
 }  
  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(1, "Laptop", "Electronics"),  
 new Product(2, "Shoes", "Fashion"),  
 new Product(3, "Keyboard", "Electronics"),  
 new Product(4, "Book", "Education")  
 };

*// Linear search* System.*out*.println("Linear Search: Looking for 'Shoes'");  
 Product result1 = LinearSearch.*search*(products, "Shoes");  
 if (result1 != null) {  
 System.*out*.println("Found: " + result1);  
 } else {  
 System.*out*.println("Not found");  
 }  
  
 *// Binary search* System.*out*.println("Binary Search: Looking for 'Book'");  
 Product result2 = BinarySearch.*search*(products, "Book");  
 if (result2 != null) {  
 System.*out*.println("Found: " + result2);  
 } else {  
 System.*out*.println("Not found");  
 }  
 }  
}  
*//lINEAR*class LinearSearch {  
 public static ECommerceSearch.Product search(ECommerceSearch.Product[] products, String name) {  
 for (int i = 0; i < products.length; i++) {  
 String currentName = products[i].productName.toLowerCase();  
 String targetName = name.toLowerCase();  
  
 if (currentName.equals(targetName)) {  
 return products[i];  
 }  
 }  
 return null;  
 }  
}

*//BINARY*class BinarySearch {  
 public static ECommerceSearch.Product search(ECommerceSearch.Product[] products, String name) {  
 Arrays.*sort*(products, (a, b) -> a.productName.toLowerCase().compareTo(b.productName.toLowerCase()));  
  
 int left = 0;  
 int right = products.length - 1;  
  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 String midName = products[mid].productName.toLowerCase();  
 String targetName = name.toLowerCase();  
  
 if (midName.equals(targetName)) {  
 return products[mid];  
 } else if (midName.compareTo(targetName) < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
  
 return null;  
 }  
}

## OUTPUT:

"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea\_rt.jar=59558" -Dfile.encoding=UTF-8 -classpath "D:\DeepSkill Cognizant\ECommerceSearch.java\out\production\ECommerceSearch.java" ECommerceSearch

Linear Search: Looking for 'Shoes'

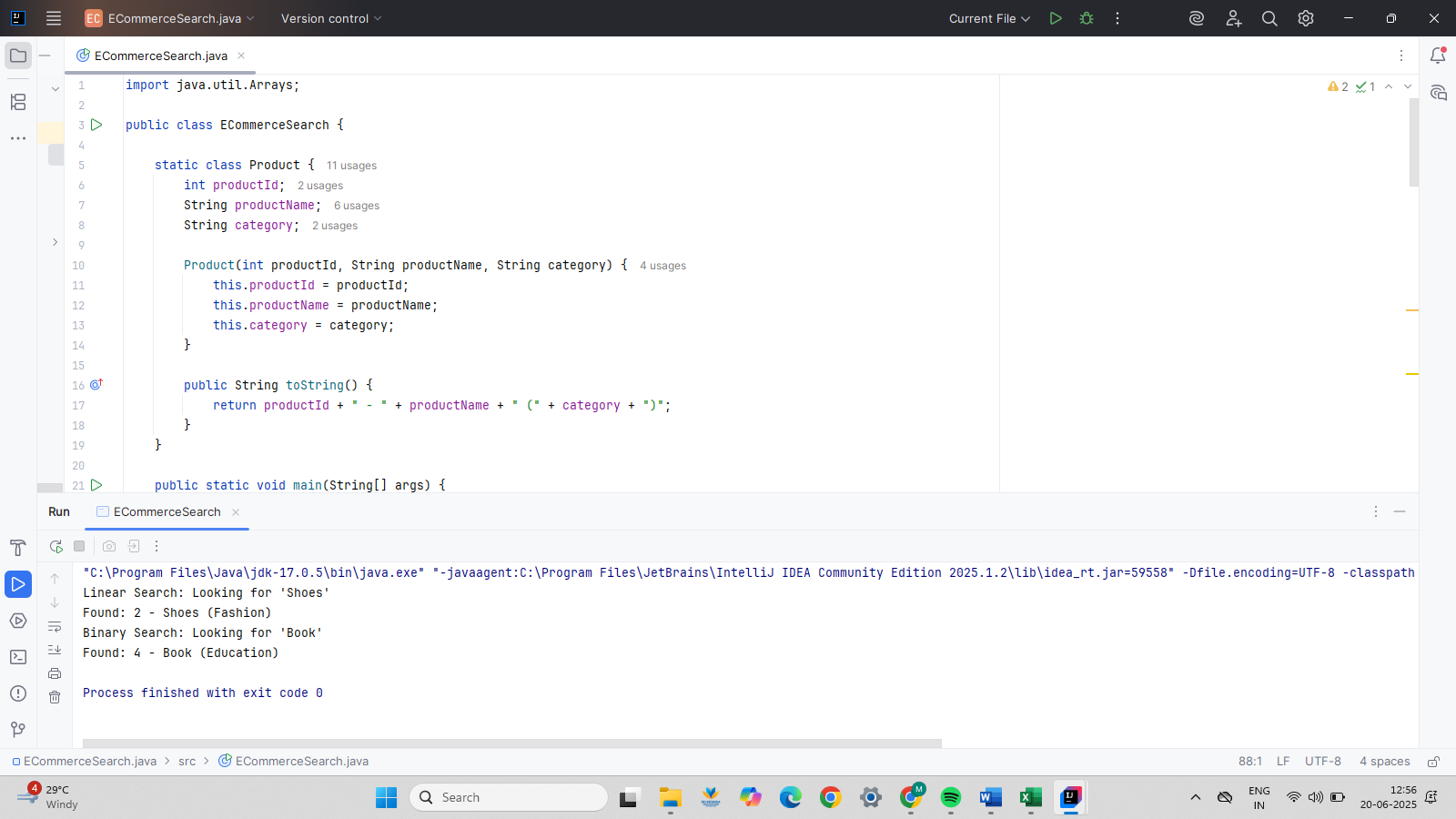
Found: 2 - Shoes (Fashion)

Binary Search: Looking for 'Book'

Found: 4 - Book (Education)

Process finished with exit code 0

## OUTPUT SCREENSHOT:



## FINAL ANALYSIS:

**Big O notation** helps us understand how fast or slow an algorithm is based on input size.

**Linear search** checks each item one by one.

* Best: O(1), Worst: O(n)

**Binary search** divides the list in half each time.

* Best: O(1), Worst: O(log n)

**Binary search is faster** but only works on sorted data.

**Linear search is simpler** and works on any data.

## EXERCISE 7: FINANCIAL FORECASTING

UNDERSTAND RECURSIVE ALGORITHMS

* Recursion is when a method calls itself to solve a smaller part of the same problem.
* It helps break down problems into simpler sub-problems.
* Recursive methods are often shorter and easier to understand than loops for certain tasks, like calculating repeated growth over years.

JAVA CODE:

public class FinancialForecast {  
  
 *// Recursive method* public static double calculateFutureValue(double presentValue, double growthRate, int years) {  
 if (years == 0) {  
 return presentValue;  
 } else {  
 return *calculateFutureValue*(presentValue, growthRate, years - 1) \* (1 + growthRate);  
 }  
 }  
  
 public static void main(String[] args) {  
 double presentValue = 10000.0; *// start amount* double growthRate = 0.05; *// 5% growth/year* int years = 5; *// No Of years* double futureValue = *calculateFutureValue*(presentValue, growthRate, years);  
 System.*out*.println("Future value after " + years + " years: ₹" + futureValue);  
 }  
}

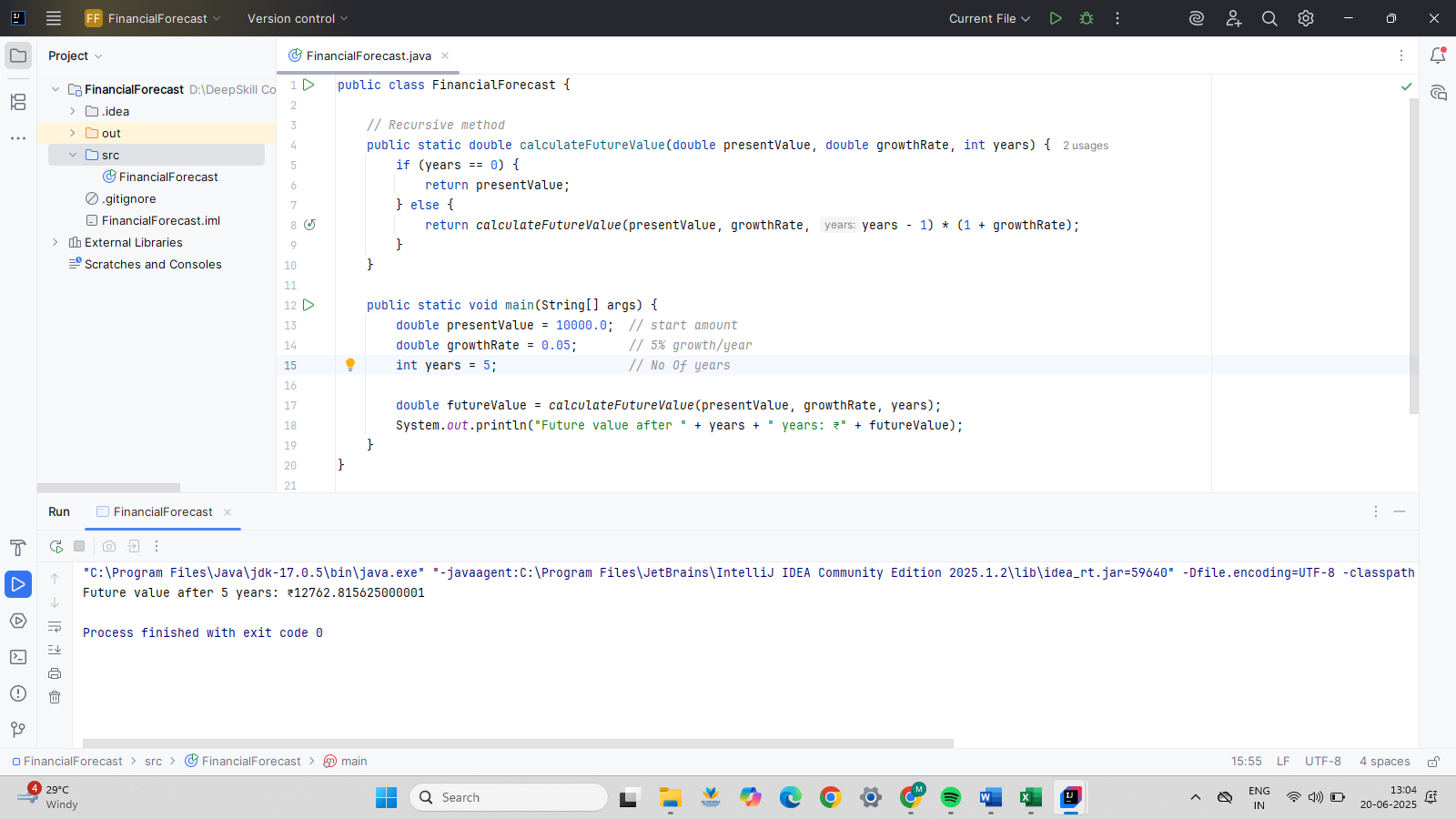
## OUTPUT:

"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea\_rt.jar=59640" -Dfile.encoding=UTF-8 -classpath "D:\DeepSkill Cognizant\FinancialForecast\out\production\FinancialForecast" FinancialForecast

Future value after 5 years: ₹12762.815625000001

Process finished with exit code 0

## OUTPUT SCREENSHOT:



## FINAL ANALYSIS:

* **Time Complexity:** The recursive function runs once for each year, so the time complexity is O(n), where n is the number of years.
* **Optimization:** To avoid too many recursive calls in large problems:
* Use a loop instead (iterative approach), or
* Store previously computed results using memoization.