Exercise 2: E-commerce Platform Search Function

**1. Understanding Asymptotic Notation**

Asymptotic notation is used to describe how fast or slow an algorithm is when the input gets bigger. It helps in understanding how much time or memory an algorithm will need, when working with large amounts of data.

For search operations, we look at three main cases:

* **Best Case:** This is when the item we’re searching for is the first one in the list, so it’s found immediately.
* **Average Case:** This is the typical situation where the item could be anywhere in the list, so it takes a few steps to find it.
* **Worst Case:** This happens when the item is at the very end or isn’t there at all, meaning the whole list has to be searched.

CODE

import java.util.ArrayList;

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 int getproductid() {

return productid;

}

public String getproductname() {

return productname;

}

public String getcategory() {

return category;

}

}

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

ArrayList<Product> products = new ArrayList<>();

products.add(new Product(1, "Asus Rog", "Laptop"));

products.add(new Product(2, "Macbook m2", "Laptop"));

products.add(new Product(3, "Samsung S8", "Tablet"));

products.add(new Product(4, "Sherlock Holmes", "Books"));

products.add(new Product(5, "Bloodline", "Book"));

products.add(new Product(6, "Dell Alienware", "Laptop"));

products.add(new Product(7, "NOKIA", "Mobile"));

products.add(new Product(8, "Jordans", "Shoes"));

products.add(new Product(9, "Ring", "Accessories"));

products.add(new Product(10, "Blue Pen", "Stationery"));

Product foundLinear = linearSearch(products, "Macbook m2");

if (foundLinear != null) {

System.out.println("Linear Search Found: " + foundLinear.getproductid() + " | "

+ foundLinear.getproductname() + " | " + foundLinear.getcategory());

} else {

System.out.println("Product not found (Linear Search).");

}

Product foundBinary = binarySearch(products, "Macbook m2");

if (foundBinary != null) {

System.out.println("Binary Search Found: " + foundBinary.getproductid() + " | "

+ foundBinary.getproductname() + " | " + foundBinary.getcategory());

} else {

System.out.println("Product not found (Binary Search).");

}

}

public static Product linearSearch(ArrayList<Product> products, String target) {

for (Product p : products) {

if (p.getproductname().equalsIgnoreCase(target)) {

return p;

}

}

return null;

}

public static Product binarySearch(ArrayList<Product> products, String target) {

Collections.sort(products, new Comparator<Product>() {

public int compare(Product p1, Product p2) {

return p1.getproductname().compareToIgnoreCase(p2.getproductname());

}

});

int left = 0;

int right = products.size() - 1;

while (left <= right) {

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

Product midProduct = products.get(mid);

int cmp = midProduct.getproductname().compareToIgnoreCase(target);

if (cmp == 0) {

return midProduct;

} else if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

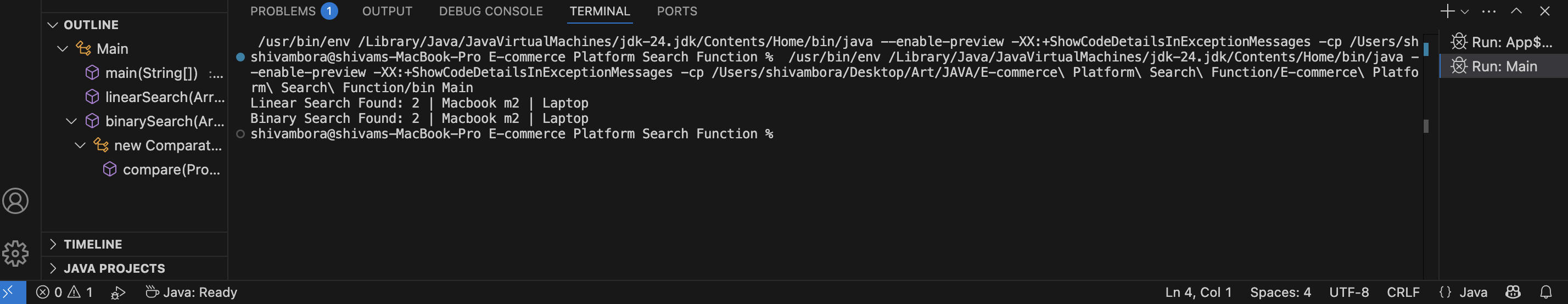
}

}

return null;

}

}

OUTPUT

4.Analysis

| **Search Method** | **Time Complexity** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- | --- |
| Linear Search | O(N) | O(1) | O(N/2) | O(N) |
| Binary Search | O(log N) | O(1) | O(log N) | O(log N) |

Linear search is fine for small lists or when the products aren’t sorted. But for larger e-commerce platforms with lots of products, **binary search** is a better choice because it’s faster.

Binary search will be the best option for an E-commerce platform search function

**Exercise 7: Financial Forecasting**

**1. Understanding Recursive Algorithms**

Recursion is a way of solving problems where a method calls itself to break down a bigger problem into smaller parts. It helps simplify problems that have repetitive steps or patterns. Instead of using loops, recursion solves a smaller piece of the problem with each step until it reaches a point where the result is known (called the base case).

Recursion is useful when the solution to a problem depends on solving smaller versions of the same problem.

2.CODE

public class FinancialForecasting {

public static double calculateFutureValue(double presentValue, double rate, int years) {

if (years == 0) return presentValue;

return (1 + rate) \* calculateFutureValue(presentValue, rate, years - 1);

}

public static double calculateFutureValueMemo(double presentValue, double rate, int years, Double[] memo) {

if (years == 0) return presentValue;

if (memo[years] != null) return memo[years];

memo[years] = (1 + rate) \* calculateFutureValueMemo(presentValue, rate, years - 1, memo);

return memo[years];

}

public static void main(String[] args) {

double pv = 134567;

double rate = 0.05;

int years = 17;

double resultRecursive = calculateFutureValue(pv, rate, years);

System.out.printf("Future Value (Recursive): %.2f%n", resultRecursive);

Double[] memo = new Double[years + 1];

double resultMemo = calculateFutureValueMemo(pv, rate, years, memo);

System.out.printf("Future Value (Memoized): %.2f%n", resultMemo);

}

}

**A computer screen shot of a black screen

AI-generated content may be incorrect.3**.OUTPUT

4.ANALYSIS

|  |  |
| --- | --- |
| **Time Complexity** | O(N), where N = number of years |
| **Space Complexity** | O(N), because of recursion stack |

The algorithm is simple but has one drawback: for every year, a new recursive call is made, which can slow things down if the number of years is very large.

**Optimization:**  
One way to make it more efficient is by using **memoization**. With memoization, the results of previous calculations are saved in memory. If the same calculation is needed again, the program uses the stored result instead of recalculating it. This avoids repeating the same work and saves time.