1. **E-commerce Platform Search Function**
2. **Big O notation**

Big O notation is used to describe how the performance of an algorithm changes as the size of the input increases. It helps us understand whether an algorithm is efficient or not, especially when we’re dealing with large datasets like in an e-commerce platform.

Different cases:

* Best Case: The minimum time an algorithm might take.
* Average Case: Expected time for typical inputs.
* Worst Case: The maximum time it might take.

1. **Code:**

**SearchFunction.java**

import java.util.Arrays;

public class SearchFunction {

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(2, "Laptop", "Electronics"),

            new Product(3, "Book", "Stationery"),

            new Product(4, "Shoes", "Footwear"),

            new Product(5, "Shirt", "Clothing")

        };

        // Linear Search

        int index1 = linearSearch(products, "Shoes");

        System.out.println("Linear Search: Found at index " + index1);

        // Binary Search

        Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

        int index2 = binarySearch(products, "Shoes");

        System.out.println("Binary Search: Found at index " + index2);

    }

    public static int linearSearch(Product[] arr, String target) {

        for (int i = 0; i < arr.length; i++) {

            if (arr[i].productName.equalsIgnoreCase(target)) {

                return i;

            }

        }

        return -1;

    }

    public static int binarySearch(Product[] arr, String target) {

        int low = 0, high = arr.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int result = arr[mid].productName.compareToIgnoreCase(target);

            if (result == 0) return mid;

            else if (result < 0) low = mid + 1;

            else high = mid - 1;

        }

        return -1;

    }

}

class Product {

    int productId;

    String productName;

    String category;

    Product(int id, String name, String cat) {

        this.productId = id;

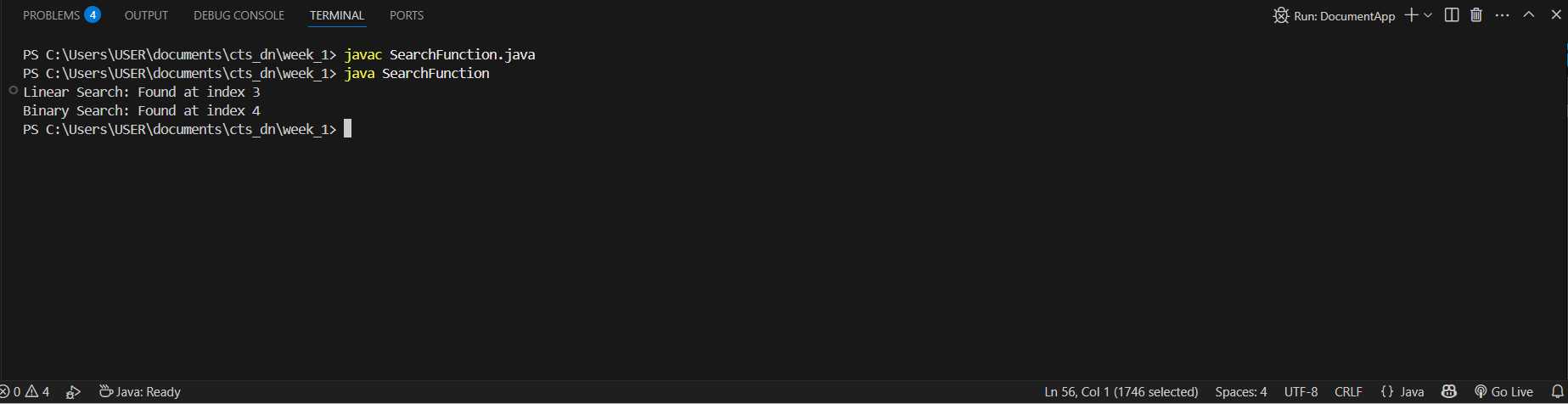
        this.productName = name;

        this.category = cat;

    }

}

**Output:**

****

1. **Time Complexity Analysis:**

Linear Search - O(n) This is used when data is unsorted or small.

Binary Search - O(log n) This is used when data is sorted and large.

**Conclusion:**

For an e-commerce platform where we deal with thousands of products, binary search is clearly more efficient as long as the product list is sorted. It's ideal for features like quick product lookup, filtering, or suggestions.

**2. Financial Forecasting**

1. **Recursive Algorithms:**

Recursion is a way of solving problems where a function calls itself to work on smaller pieces of the same problem. It helps break down repetitive problems into simple steps, which can make code easier to write and understand.

1. **Code:**

**Forecast.java**

public class Forecast {

    public static void main(String[] args) {

        double startingAmount = 15000;

        double growthRate = 0.10;

        int years = 4;

        double result = calculateFutureValue(startingAmount, growthRate, years);

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

    }

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

        if (years == 0) {

            return amount;

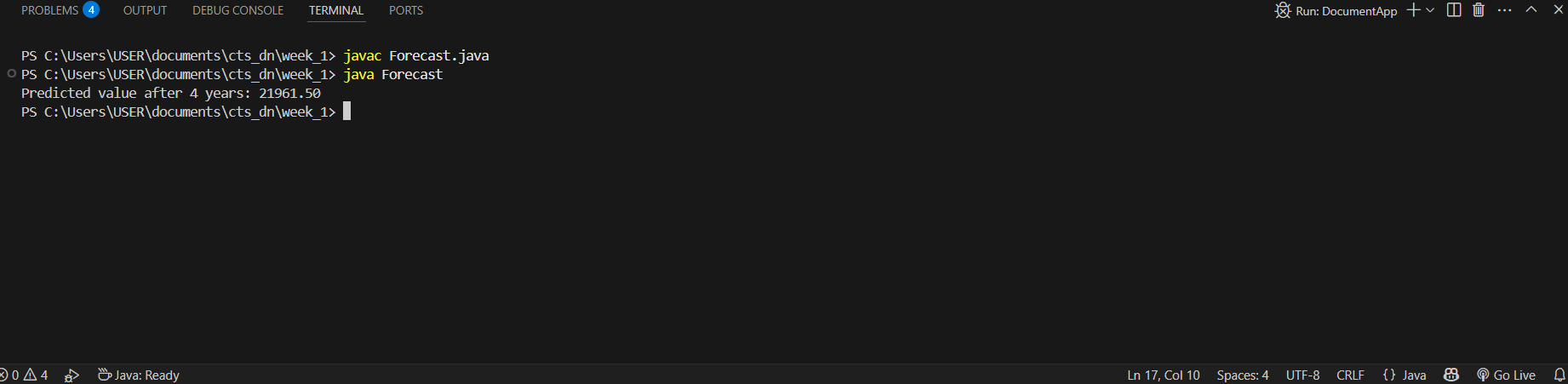
        }

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

    }

}

**Output:**

****

1. **Time Complexity:**

The time complexity of this recursive method is O(n), where n is the number of years. This is because the method runs once for each year.

1. **Optimization:**

Since each step only depends on the previous year's value, the recursion doesn’t need to repeat any calculations. But for larger or more complex cases, we can convert it into a loop to save memory and avoid stack overflow.