**WEEK 1- DSA Mandatory EXERCISES - 6428652**

**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.

**ANSWER:**

class Product {

    int productId;

    String productName;

    String category;

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

        this.productId = id;

        this.productName = name;

        this.category = category;

    }

}

// Main program class

public class EcommerceSearch {

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

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

            if (products[i].productName.equalsIgnoreCase(keyName)) {

                return i;

            }

        }

        return -1;

    }

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

        int left = 0, right = products.length - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            int compare = keyName.compareToIgnoreCase(products[mid].productName);

            if (compare == 0) return mid;

            else if (compare < 0) right = mid - 1;

            else left = mid + 1;

        }

        return -1;

    }

    public static void sortByName(Product[] products) {

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

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Shoes", "Footwear"),

            new Product(102, "Phone", "Electronics"),

            new Product(103, "T-shirt", "Clothing"),

            new Product(104, "Laptop", "Electronics"),

        };

        System.out.println("== Linear Search ==");

        int linearResult = linearSearch(products, "Phone");

        System.out.println(linearResult != -1 ? "Found at index " + linearResult : "Not found");

        System.out.println("\n== Binary Search ==");

        sortByName(products);

        int binaryResult = binarySearch(products, "Phone");

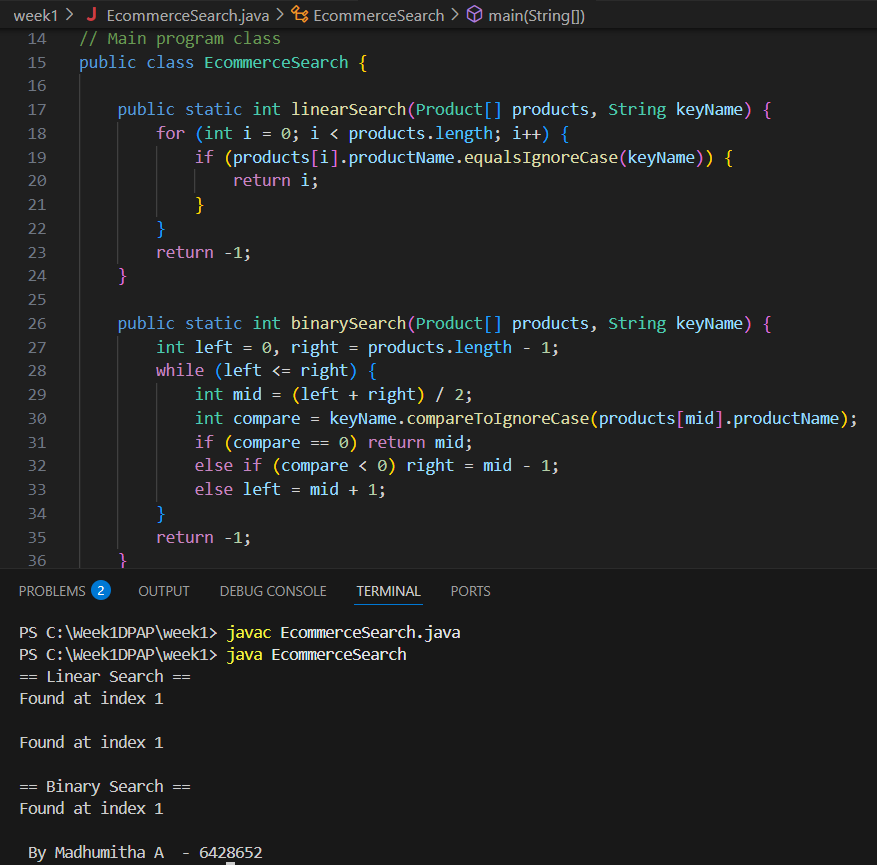
        System.out.println(binaryResult != -1 ? "Found at index " + binaryResult : "Not found");

        System.out.println("\n By Madhumitha A - 6428652");

    }

}

|  |  |  |
| --- | --- | --- |
| **Notation** | **Meaning** | **Example** |
| O(1) | Constant time | Accessing array element |
| O(n) | Linear time | Linear Search |
| O(log n) | Logarithmic time | Binary Search |
| O(n²) | Quadratic time | Bubble Sort |

****

**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.

**ANSWER:**

public class FinancialForecast {

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

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

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

}

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

if (years == 0) return presentValue;

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

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

return memo[years];

}

public static void main(String[] args) {

double presentValue = 10000; // ₹10,000 investment

double growthRate = 0.07; // 7% annual growth

int years = 5;

System.out.println("== Simple Recursive Forecast ==");

double result = forecastRecursive(presentValue, growthRate, years);

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

System.out.println("\n== Optimized Recursive Forecast (Memoized) ==");

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

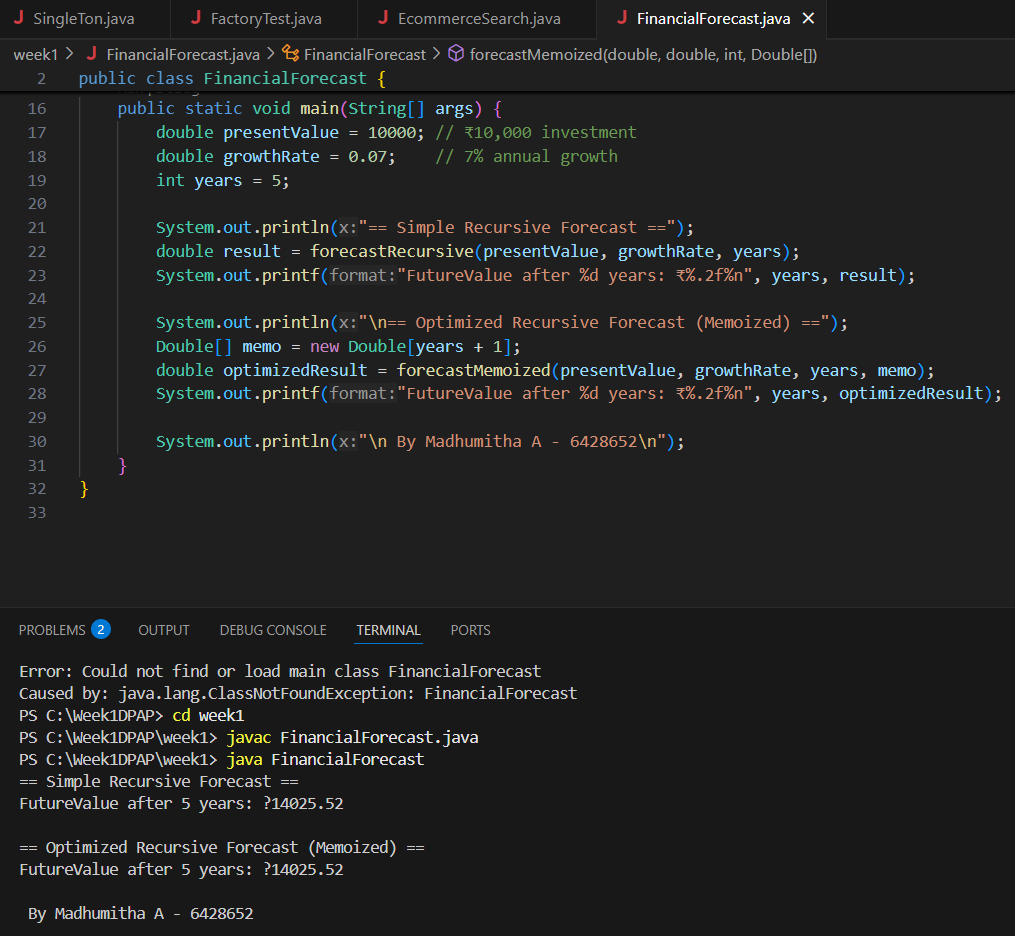
double optimizedResult = forecastMemoized(presentValue, growthRate, years, memo);

System.out.printf("FutureValue after %d years: ₹%.2f%n", years, optimizedResult);

System.out.println("\n By Madhumitha A - 6428652\n");

}

}



|  |  |
| --- | --- |
| **Method** | **Time Complexity** |
| Simple Recursion | O(n) |
| Memoized Version | O(n) |