**Superset ID:6373322**

**COGNITZANT DIGITAL NURTURE 4.0 JAVA FSE**

**WEEK-1: ALGORITHMS\_DATA STRUCTURES**

**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.

**CODE:**

**ECommerceSearchComparision.java**

package week1;

import java.util.\*;

//Product class with attributes

class Product {

int productId;

String productName;

String category;

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

this.productId = id;

this.productName = name.toLowerCase();

this.category = category.toLowerCase();

}

*@Override*

public String toString() {

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

}

}

public class ECommerceSearchComparision {

// Linear search by name

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

for (Product product : products) {

if (product.productName.contains(keyword.toLowerCase())) {

return product;

}

}

return null;

}

// Binary search by name

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

int left = 0;

int right = products.length - 1;

keyword = keyword.toLowerCase();

while (left <= right) {

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

int cmp = products[mid].productName.compareTo(keyword);

if (products[mid].productName.equals(keyword)) {

return products[mid];

} else if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

// Sample product list

Product[] products = {

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

new Product(2, "Speaker", "Audio"),

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

new Product(4, "Keyboard", "Electronics"),

new Product(5, "Laptop", "Computers")

};

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter product name to search: ");

String keyword = sc.nextLine();

// Linear Search

long startTime = System.*nanoTime*();

Product linearResult = *linearSearch*(products, keyword);

long endTime = System.*nanoTime*();

System.***out***.println("\nLinear Search Result:");

System.***out***.println(linearResult != null ? linearResult : "Product not found.");

System.***out***.println("Time Taken: " + (endTime - startTime) + " ns");

// Sort for Binary Search

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

// Binary Search

startTime = System.*nanoTime*();

Product binaryResult = *binarySearch*(products, keyword);

endTime = System.*nanoTime*();

System.***out***.println("\nBinary Search Result:");

System.***out***.println(binaryResult != null ? binaryResult : "Product not found.");

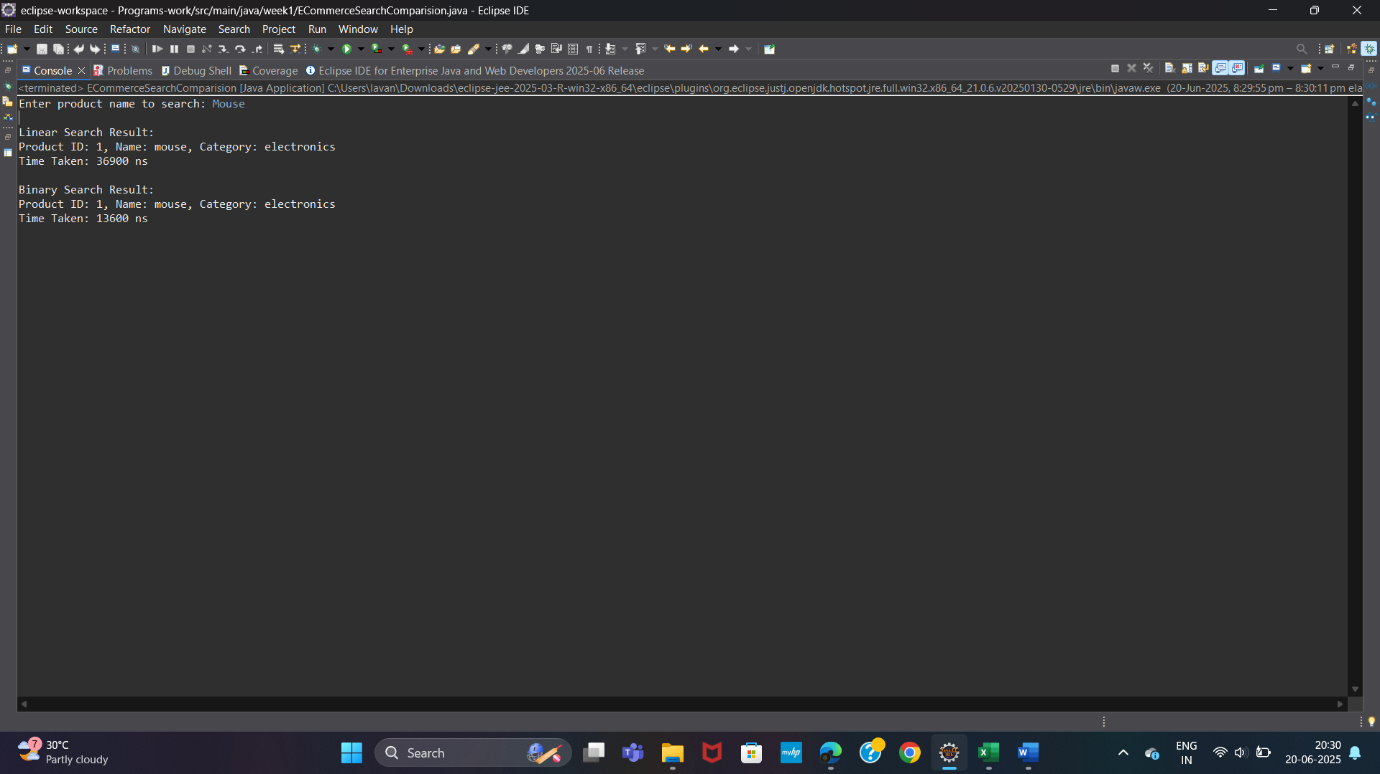
System.***out***.println("Time Taken: " + (endTime - startTime) + " ns");

sc.close();

}

}

Output:



**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.

**CODE**:

**FinancialForecastingRecursive.java**

package week1;

import java.util.Scanner;

public class FinancialForecastingRecursive {

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

for (int i = 0; i < years; i++) {

currentValue \*= (1 + growthRate);

}

return currentValue;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

// Input

System.***out***.print("Enter current value (e.g., revenue): ₹");

double currentValue = scanner.nextDouble();

System.***out***.print("Enter annual growth rate (in %, e.g., 5): ");

double ratePercent = scanner.nextDouble();

double growthRate = ratePercent / 100;

System.***out***.print("Enter number of years to forecast: ");

int years = scanner.nextInt();

// Recursive Forecast

double futureValue = *forecastIterative*(currentValue, growthRate, years);

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

scanner.close();

}

}

Output:

