**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.

**SOURCE CODE :**

**Product.java**

package com.ecommerce.search;

public 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;

}

@Override

public String toString() {

return "Product [ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";

}

}

**SearchDemo.java**

package com.ecommerce.search;

import java.util.Arrays;

import java.util.Comparator;

public class SearchDemo {

// Linear Search by product name

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

for (Product product : products) {

if (product.productName.equalsIgnoreCase(productName)) {

return product;

}

}

return null;

}

// Binary Search by product name

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (result == 0) {

return products[mid];

} else if (result < 0) {

high = mid - 1;

} else {

low = mid + 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shirt", "Clothing"),

new Product(103, "Book", "Education"),

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

new Product(105, "Shoes", "Footwear")

};

String searchItem = "Book";

// === Linear Search ===

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

long startTimeLinear = System.nanoTime();

Product resultLinear = linearSearch(products, searchItem);

long endTimeLinear = System.nanoTime();

if (resultLinear != null)

System.out.println("Result: " + resultLinear);

else

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

System.out.println("Time Complexity: O(n)");

System.out.println("Execution Time: " + (endTimeLinear - startTimeLinear) + " ns");

// === Binary Search ===

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

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

long startTimeBinary = System.nanoTime();

Product resultBinary = binarySearch(products, searchItem);

long endTimeBinary = System.nanoTime();

if (resultBinary != null)

System.out.println("Result: " + resultBinary);

else

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

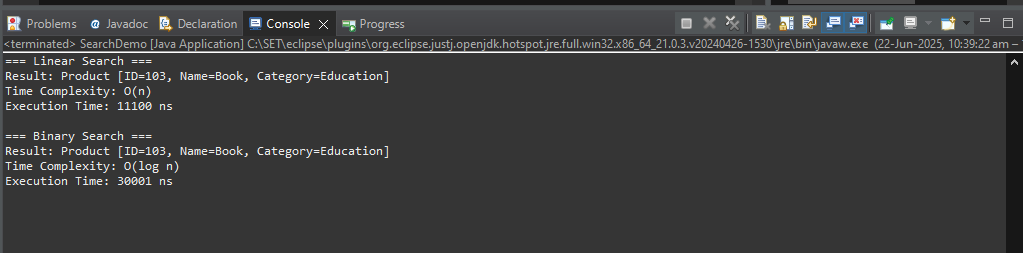
System.out.println("Time Complexity: O(log n)");

System.out.println("Execution Time: " + (endTimeBinary - startTimeBinary) + " ns");

}

}

**OUTPUT :**

****

**Exercise 7: Financial Forecasting**

**SCENARIO:**

You are developing a financial forecasting tool that predicts future values based on past data.

**SOURCE CODE :**

**FinancialForecast.java**

package com.forecasting.tool;

public class FinancialForecast {

// Recursive method to calculate future value

public static double calculateFutureValueRecursive(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue; // base case

}

return (1 + growthRate) \* calculateFutureValueRecursive(presentValue, growthRate, years - 1); // recursive case

}

// Optimized: Iterative version

public static double calculateFutureValueIterative(double presentValue, double growthRate, int years) {

double futureValue = presentValue;

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

futureValue \*= (1 + growthRate);

}

return futureValue;

}

public static void main(String[] args) {

double presentValue = 10000; // Starting amount

double annualGrowthRate = 0.08; // 8% growth rate

int years = 5;

// Recursive calculation

long startTimeRec = System.nanoTime();

double futureValueRec = calculateFutureValueRecursive(presentValue, annualGrowthRate, years);

long endTimeRec = System.nanoTime();

// Iterative calculation

long startTimeIter = System.nanoTime();

double futureValueIter = calculateFutureValueIterative(presentValue, annualGrowthRate, years);

long endTimeIter = System.nanoTime();

System.out.printf("Recursive Future Value after %d years: %.2f\n", years, futureValueRec);

System.out.printf("Iterative Future Value after %d years: %.2f\n", years, futureValueIter);

System.out.println("\n=== Time Complexity Analysis ===");

System.out.println("Recursive Method: O(n), Space: O(n) due to call stack");

System.out.println("Iterative Method: O(n), Space: O(1)");

System.out.println("\nExecution Time:");

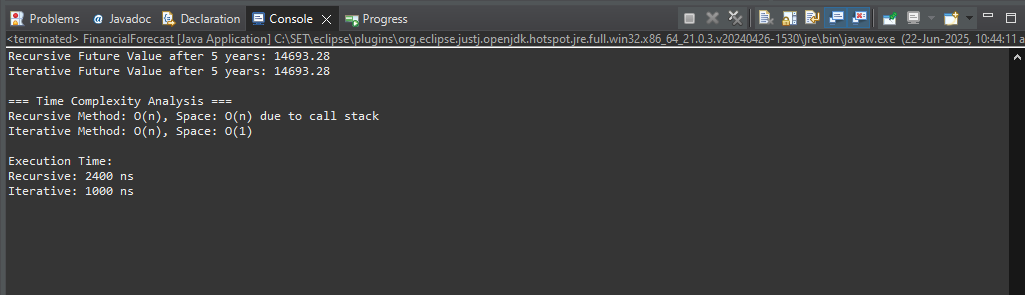
System.out.println("Recursive: " + (endTimeRec - startTimeRec) + " ns");

System.out.println("Iterative: " + (endTimeIter - startTimeIter) + " ns");

}

}

**OUTPUT :**

****