**Data Structures and Algorithms Solution**

**Exercise 2: E-commerce Platform Search Function**

**Code :**

**Product.java**

package dsa;

//Product.java

public class Product {

private int productId;

private String productName;

private 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;

}

@Override

public String toString() {

return "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

**SearchEngine.java**

package dsa;

import java.util.Arrays;

import java.util.Comparator;

public class SearchEngine {

// Linear Search by Product ID

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

for (Product product : products) {

if (product.getProductId() == targetId) {

return product;

}

}

return null;

}

// Binary Search by Product ID (requires sorted array)

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

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

while (left <= right) {

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

int midId = products[mid].getProductId();

if (midId == targetId) {

return products[mid];

} else if (midId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(101, "T-Shirt", "Clothing"),

new Product(103, "Shoes", "Footwear"),

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

};

// Sort for binary search

Arrays.*sort*(products, Comparator.*comparingInt*(Product::getProductId));

int searchId = 103;

// Linear Search

Product resultLinear = *linearSearch*(products, searchId);

System.*out*.println("Linear Search Result: " + (resultLinear != null ? resultLinear : "Product not found."));

// Binary Search

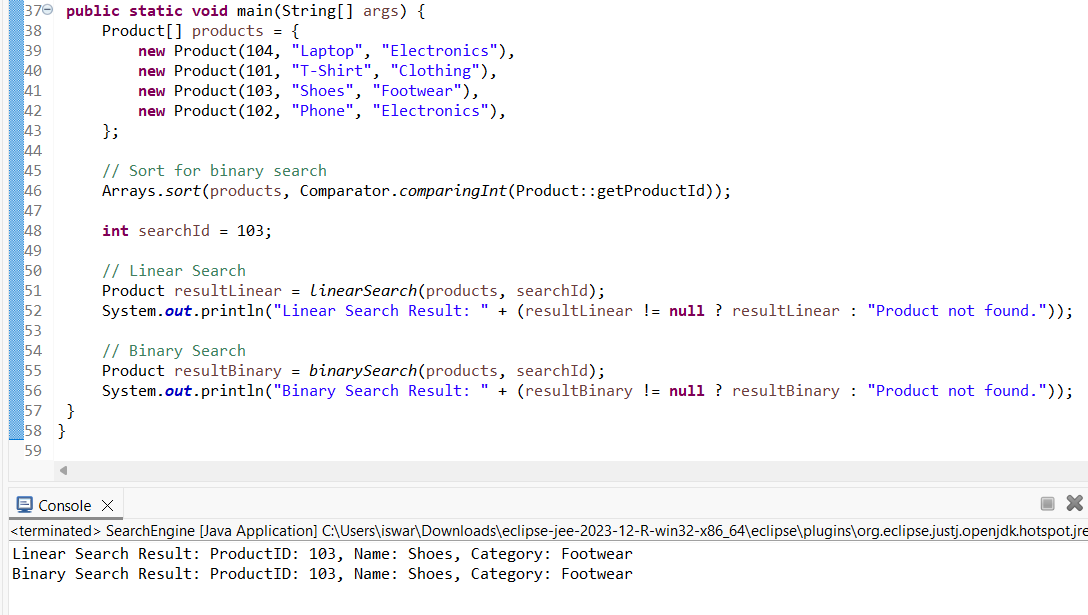
Product resultBinary = *binarySearch*(products, searchId);

System.*out*.println("Binary Search Result: " + (resultBinary != null ? resultBinary : "Product not found."));

}

}

**Output :**

****

**Analysis :**

I implemented both linear and binary search to find a product by its ID. While linear search works for unsorted data, binary search provides a much better time complexity (O(log n)) when data is sorted. For scalable systems like e-commerce, binary search or database indexing strategies are more suitable.

**Exercise 7: Financial Forecasting**

**Code:**

**FinancialForecast.java**

package dsa;

public class FinancialForecast {

public static double forecastRecursive(double baseValue, double growthRate, int years) {

if (years == 0) {

return baseValue; }

return *forecastRecursive*(baseValue, growthRate, years - 1) \* (1 + growthRate);

}

// Optimized with memoization

public static double forecastMemoized(double baseValue, double growthRate, int years, Double[] cache) {

if (years == 0) {

return baseValue; }

if (cache[years] != null) {

return cache[years]; }

cache[years] = *forecastMemoized*(baseValue, growthRate, years - 1, cache) \* (1 + growthRate);

return cache[years];

}

// Iterative version

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

double result = baseValue;

for (int i = 1; i <= years; i++) {

result \*= (1 + growthRate);

}

return result;

}

public static void main(String[] args) {

double base = 10000;

double rate = 0.05;

int years = 10;

System.*out*.println("Recursive Forecast: " + *forecastRecursive*(base, rate, years));

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

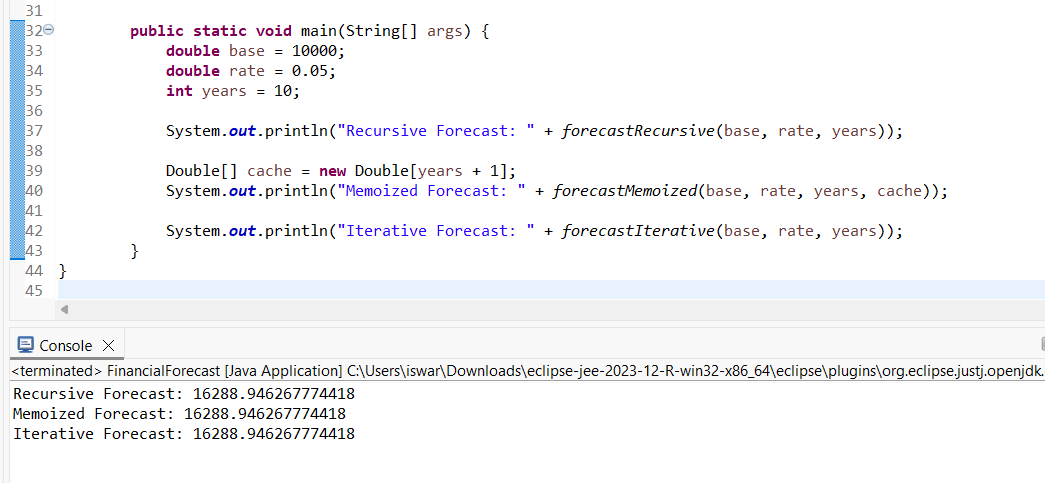
System.*out*.println("Memoized Forecast: " + *forecastMemoized*(base, rate, years, cache));

System.*out*.println("Iterative Forecast: " + *forecastIterative*(base, rate, years));

}

}

**Output :**



**Analysis :**

I created a recursive function to calculate future financial values using compound interest. The recursive method is elegant but can be inefficient for large n, so I optimized it using memoization and also provided an iterative version which is the most efficient. This shows how recursion can be expressive, but optimization is key in production-grade systems.