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

import java.util.\*;  
  
class Product implements Comparable<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 int compareTo(Product other) {  
 return Integer.*compare*(this.productId, other.productId);  
 }  
  
 @Override  
 public String toString() {  
 return "Product{id=" + productId + ", name='" + productName + "', category='" + category + "'}";  
 }  
}  
  
public class ECommerceSearch {  
  
 public static Product linearSearch(Product[] products, int targetId) {  
 for (Product product : products) {  
 if (product.productId == targetId) {  
 return product;  
 }  
 }  
 return null;  
 }  
  
 public static Product binarySearch(Product[] products, int targetId) {  
 int left = 0, right = products.length - 1;  
 while (left <= right) {  
 int mid = left + (right - left) / 2;  
 if (products[mid].productId == targetId) {  
 return products[mid];  
 } else if (products[mid].productId < targetId) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 return null;  
 }  
  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(105, "Laptop", "Electronics"),  
 new Product(101, "Phone", "Electronics"),  
 new Product(103, "Shoes", "Fashion"),  
 new Product(104, "Watch", "Accessories"),  
 new Product(102, "Bag", "Travel")  
 };  
  
 *// Linear Search (Unsorted)* System.*out*.println("Linear Search Result:");  
 long startLinear = System.*nanoTime*();  
 Product foundLinear = *linearSearch*(products, 103);  
 long endLinear = System.*nanoTime*();  
 System.*out*.println(foundLinear);  
 System.*out*.println("Time taken for Linear Search: " + (endLinear - startLinear) + " ns");  
  
 *// Binary Search (Sorted)* Arrays.*sort*(products);  
 System.*out*.println("\nBinary Search Result:");  
 long startBinary = System.*nanoTime*();  
 Product foundBinary = *binarySearch*(products, 103);  
 long endBinary = System.*nanoTime*();  
 System.*out*.println(foundBinary);  
 System.*out*.println("Time taken for Binary Search: " + (endBinary - startBinary) + " ns");  
 }  
}

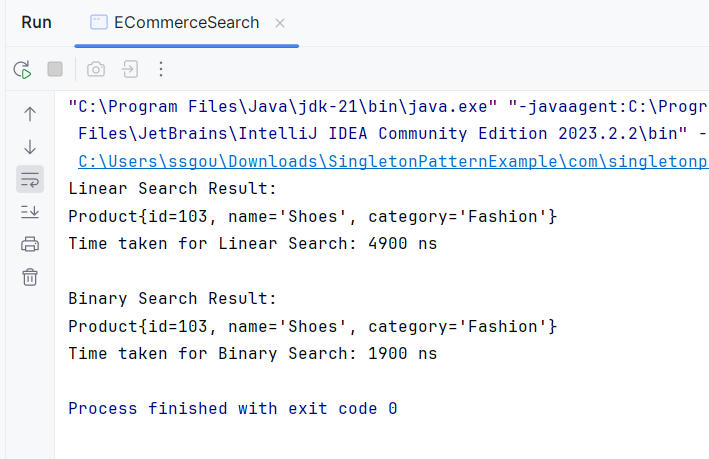
**Analysis:**

**Linear Search: O(n) - Best: O(1), Worst: O(n)**

**Binary Search: O(log n) - Requires sorted array**

**Binary Search is preferred for large, sorted datasets.**

**Output:**



**Exercise 7: Financial Forecasting**

public class FinancialForecasting {

// Recursive method to calculate future value

public static double forecastValue(int year, double[] pastData) {

if (year < pastData.length) {

return pastData[year];

}

double lastGrowth = (pastData[pastData.length - 1] - pastData[pastData.length - 2]) / pastData[pastData.length - 2];

double nextYearValue = forecastValue(year - 1, pastData) \* (1 + lastGrowth);

return nextYearValue;

}

// Optimized version using memoization

public static double forecastValueMemo(int year, double[] pastData, Double[] memo) {

if (year < pastData.length) {

return pastData[year];

}

if (memo[year] != null) {

return memo[year];

}

double lastGrowth = (pastData[pastData.length - 1] - pastData[pastData.length - 2]) / pastData[pastData.length - 2];

memo[year] = forecastValueMemo(year - 1, pastData, memo) \* (1 + lastGrowth);

return memo[year];

}

public static void main(String[] args) {

double[] pastData = {100, 110, 121}; // Simulated data for 3 years

int targetYear = 6;

// Forecast using naive recursion

long startRecursive = System.nanoTime();

double resultRecursive = forecastValue(targetYear, pastData);

long endRecursive = System.nanoTime();

System.out.println("Forecast for Year " + targetYear + " (Recursion): " + resultRecursive);

System.out.println("Time taken (Recursive): " + (endRecursive - startRecursive) + " ns");

// Forecast using memoization

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

long startMemo = System.nanoTime();

double resultMemo = forecastValueMemo(targetYear, pastData, memo);

long endMemo = System.nanoTime();

System.out.println("Forecast for Year " + targetYear + " (Memoized): " + resultMemo);

System.out.println("Time taken (Memoized): " + (endMemo - startMemo) + " ns");

}

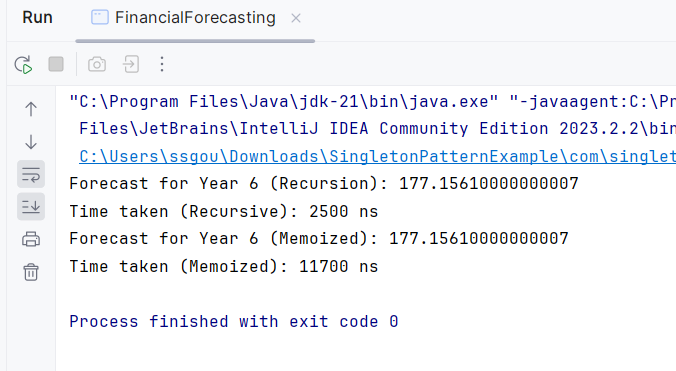
}

**Analysis**

**Time Complexity (Recursive): Exponential (O(2^n)) without memoization**

**Time Complexity (Memoized): Linear (O(n))**

**Output**

****