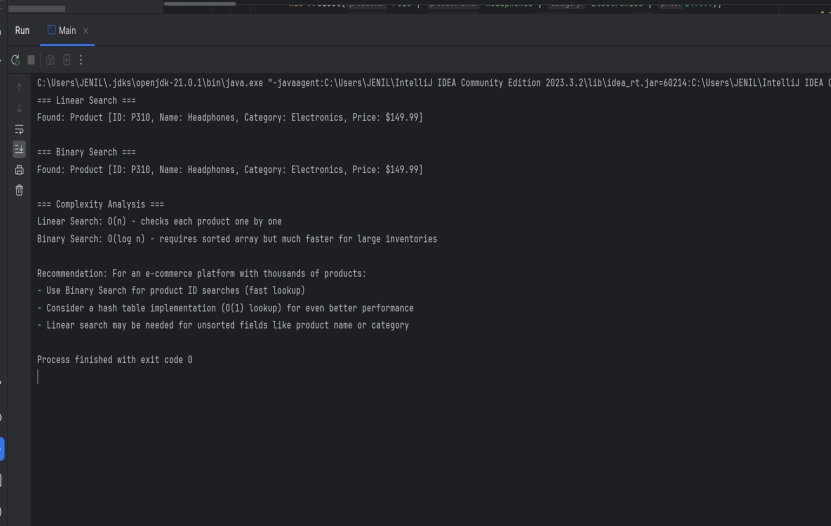
Algorithms\_Data Structures

Exercise 2: E-commerce Platform Search Function

import java.util.Arrays;  
import java.util.Comparator;  
  
class Product {  
 private String productId;  
 private String productName;  
 private String category;  
 private double price;  
  
 public Product(String productId, String productName, String category, double price) {  
 this.productId = productId;  
 this.productName = productName;  
 this.category = category;  
 this.price = price;  
 }  
  
   
 public String getProductId() { return productId; }  
 public String getProductName() { return productName; }  
 public String getCategory() { return category; }  
 public double getPrice() { return price; }  
  
 public String toString() {  
 return "Product [ID: " + productId + ", Name: " + productName +  
 ", Category: " + category + ", Price: $" + price + "]";  
 }  
}  
  
public class Main {  
  
   
 public static Product linearSearch(Product[] products, String targetId) {  
 for (Product product : products) {  
 if (product.getProductId().equals(targetId)) {  
 return product;  
 }  
 }  
 return null;  
 }  
  
   
 public static Product binarySearch(Product[] sortedProducts, String targetId) {  
 int left = 0;  
 int right = sortedProducts.length - 1;  
  
 while (left <= right) {  
 int mid = left + (right - left) / 2;  
 int comparison = sortedProducts[mid].getProductId().compareTo(targetId);  
  
 if (comparison == 0) {  
 return sortedProducts[mid];  
 } else if (comparison < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 return null;  
 }  
  
 public static void main(String[] args) {  
   
 Product[] products = {  
 new Product("P100", "Smartphone", "Electronics", 599.99),  
 new Product("P205", "Laptop", "Electronics", 999.99),  
 new Product("P310", "Headphones", "Electronics", 149.99),  
 new Product("P420", "T-Shirt", "Clothing", 19.99),  
 new Product("P530", "Jeans", "Clothing", 49.99)  
 };  
  
   
 Product[] sortedProducts = Arrays.*copyOf*(products, products.length);  
 Arrays.*sort*(sortedProducts, Comparator.*comparing*(Product::getProductId));  
  
   
 String targetId = "P310";  
  
 System.*out*.println("=== Linear Search ===");  
 Product resultLinear = *linearSearch*(products, targetId);  
 System.*out*.println(resultLinear != null ? "Found: " + resultLinear : "Product not found");  
  
 System.*out*.println("\n=== Binary Search ===");  
 Product resultBinary = *binarySearch*(sortedProducts, targetId);  
 System.*out*.println(resultBinary != null ? "Found: " + resultBinary : "Product not found");  
  
   
 System.*out*.println("\n=== Complexity Analysis ===");  
 System.*out*.println("Linear Search: O(n) - checks each product one by one");  
 System.*out*.println("Binary Search: O(log n) - requires sorted array but much faster for large inventories");  
 System.*out*.println("\nRecommendation: For an e-commerce platform with thousands of products:");  
 System.*out*.println("- Use Binary Search for product ID searches (fast lookup)");  
 System.*out*.println("- Consider a hash table implementation (O(1) lookup) for even better performance");  
 System.*out*.println("- Linear search may be needed for unsorted fields like product name or category");  
 }  
}

**output:**



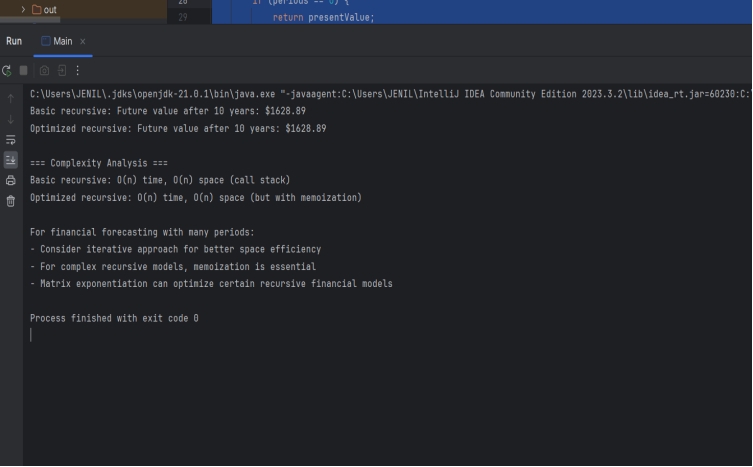
------------------------------------------------------------------------------------

Algorithms\_Data Structures

Exercise 7: Financial Forecasting

import java.util.\*;  
import java.util.Arrays;  
  
public class Main{  
  
  
 public static double calculateFutureValue(double presentValue, double growthRate, int periods) {  
  
 if (periods == 0) {  
 return presentValue;  
 }  
  
  
 return (1 + growthRate) \* *calculateFutureValue*(presentValue, growthRate, periods - 1);  
 }  
  
  
 private static double[] *memo*;  
  
 public static double calculateFutureValueOptimized(double presentValue, double growthRate, int periods) {  
 *memo* = new double[periods + 1];  
 Arrays.*fill*(*memo*, -1);  
 return *calculateFutureValueMemo*(presentValue, growthRate, periods);  
 }  
  
 private static double calculateFutureValueMemo(double presentValue, double growthRate, int periods) {  
 if (periods == 0) {  
 return presentValue;  
 }  
  
 if (*memo*[periods] != -1) {  
 return *memo*[periods];  
 }  
  
 *memo*[periods] = (1 + growthRate) \* *calculateFutureValueMemo*(presentValue, growthRate, periods - 1);  
 return *memo*[periods];  
 }  
  
 public static void main(String[] args) {  
 double pv = 1000.0;  
 double rate = 0.05;   
 int years = 10;   
  
  
 double fv = *calculateFutureValue*(pv, rate, years);  
 System.*out*.printf("Basic recursive: Future value after %d years: $%.2f%n", years, fv);  
  
  
 double fvOpt = *calculateFutureValueOptimized*(pv, rate, years);  
 System.*out*.printf("Optimized recursive: Future value after %d years: $%.2f%n", years, fvOpt);  
  
  
 System.*out*.println("\n=== Complexity Analysis ===");  
 System.*out*.println("Basic recursive: O(n) time, O(n) space (call stack)");  
 System.*out*.println("Optimized recursive: O(n) time, O(n) space (but with memoization)");  
 System.*out*.println("\nFor financial forecasting with many periods:");  
 System.*out*.println("- Consider iterative approach for better space efficiency");  
 System.*out*.println("- For complex recursive models, memoization is essential");  
 System.*out*.println("- Matrix exponentiation can optimize certain recursive financial models");  
 }  
}

**Output:**



--------------------------------------------------------------------------------------------------------------------------------------