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

**Code:**

**ECommerceSearch.java**

import java.util.Arrays;

import java.util.Scanner;

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 this.productName.compareToIgnoreCase(other.productName);

}

@Override

public String toString() {

return "[" + productId + ", " + productName + ", " + category + "]";

}

}

public class ECommerceSearch {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

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

while (left <= right) {

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

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

if (cmp == 0)

return products[mid];

else if (cmp < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Product[] products = {

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

new Product(102, "Shoes", "Footwear"),

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

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

new Product(105, "Watch", "Accessories")

};

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

String target = sc.nextLine();

long startLinear = System.nanoTime();

Product foundLinear = linearSearch(products, target);

long endLinear = System.nanoTime();

Arrays.sort(products);

long startBinary = System.nanoTime();

Product foundBinary = binarySearch(products, target);

long endBinary = System.nanoTime();

System.out.println("\n--- Search Results ---");

if (foundLinear != null)

System.out.println("Linear Search Found: " + foundLinear);

else

System.out.println("Linear Search: Product not found");

if (foundBinary != null)

System.out.println("Binary Search Found: " + foundBinary);

else

System.out.println("Binary Search: Product not found");

System.out.println("\n--- Time Comparison ---");

System.out.println("Linear Search Time: " + (endLinear - startLinear) + " ns");

System.out.println("Binary Search Time: " + (endBinary - startBinary) + " ns");

System.out.println("\n--- Analysis ---");

System.out.println("Linear Search: O(n) - Scans each element, slower for large datasets.");

System.out.println("Binary Search: O(log n) - Much faster but requires sorted data.");

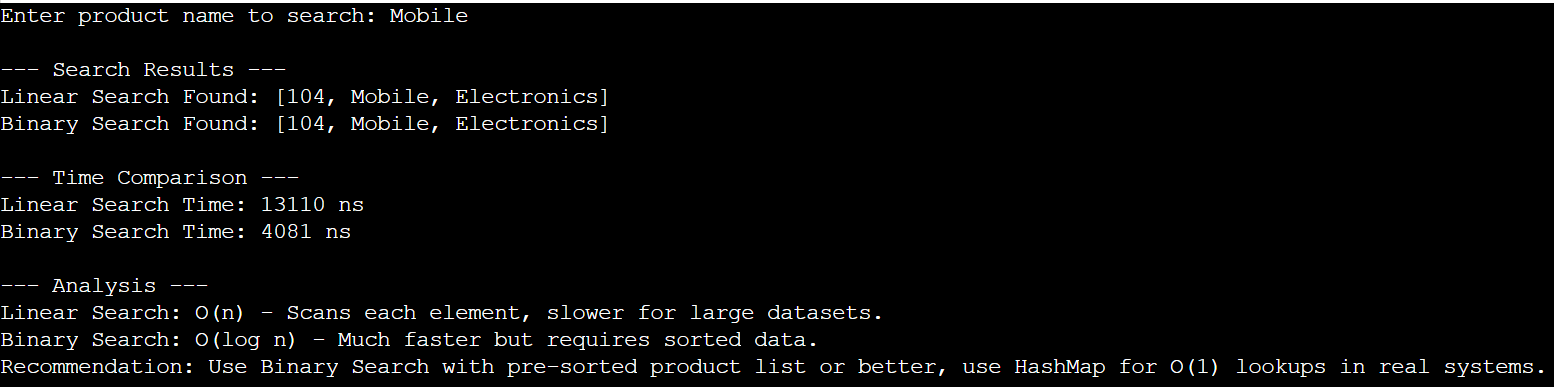
System.out.println("Recommendation: Use Binary Search with pre-sorted product list or better, use HashMap for O(1) lookups in real systems.");

sc.close();

}

}

**Output:**

****

**Exercise 7: Financial Forecasting**

**Code:**

**FinancialForecast.java**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class FinancialForecast {

static Map<Integer, Double> memo = new HashMap<>();

public static double forecastValue(double initialAmount, double growthRate, int years) {

if (years == 0)

return initialAmount;

if (memo.containsKey(years))

return memo.get(years);

double result = (1 + growthRate) \* forecastValue(initialAmount, growthRate, years - 1);

memo.put(years, result);

return result;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter initial amount: ");

double initialAmount = sc.nextDouble();

System.out.print("Enter annual growth rate (in %): ");

double rate = sc.nextDouble();

double growthRate = rate / 100.0;

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

int years = sc.nextInt();

double futureValue = forecastValue(initialAmount, growthRate, years);

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

System.out.println("\n--- Analysis ---");

System.out.println("Recursive approach time complexity: O(n)");

System.out.println("Without memoization, it would be exponential: O(2^n)");

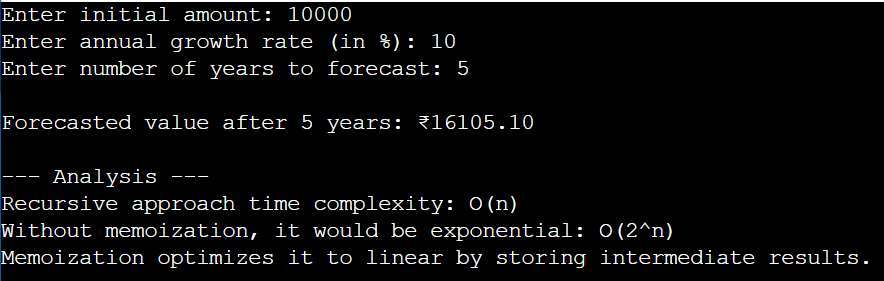
System.out.println("Memoization optimizes it to linear by storing intermediate results.");

sc.close();

}

}

**Output:**

****