**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.

**Java Code :**import java.util.\*;

public class EcommerceSearchFunction {

    // Product class

    static class Product {

        int productId;

        String productName;

        String category;

        Product(int productId, String productName, String category) {

            this.productId = productId;

            this.productName = productName.toLowerCase();

            this.category = category.toLowerCase();

        }

        public String toString() {

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

        }

    }

    // Linear Search

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

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

    // Binary Search (products must be sorted by name)

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

        int left = 0;

        int right = products.length - 1;

        targetName = targetName.toLowerCase();

        while (left <= right) {

            int mid = (left + right) / 2;

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

            if (cmp == 0) return products[mid];

            else if (cmp < 0) left = mid + 1;

            else right = mid - 1;

        }

        return null;

    }

    // Main method

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Shoes", "Footwear"),

            new Product(102, "T-Shirt", "Apparel"),

            new Product(103, "Laptop", "Electronics"),

            new Product(104, "Watch", "Accessories")

        };

        // Sort for binary search

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

        String searchTerm = "Laptop";

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

        Product found1 = linearSearch(products, searchTerm);

        System.out.println(found1 != null ? found1 : "Product not found.");

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

        Product found2 = binarySearch(products, searchTerm);

        System.out.println(found2 != null ? found2 : "Product not found.");

        // Analysis

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

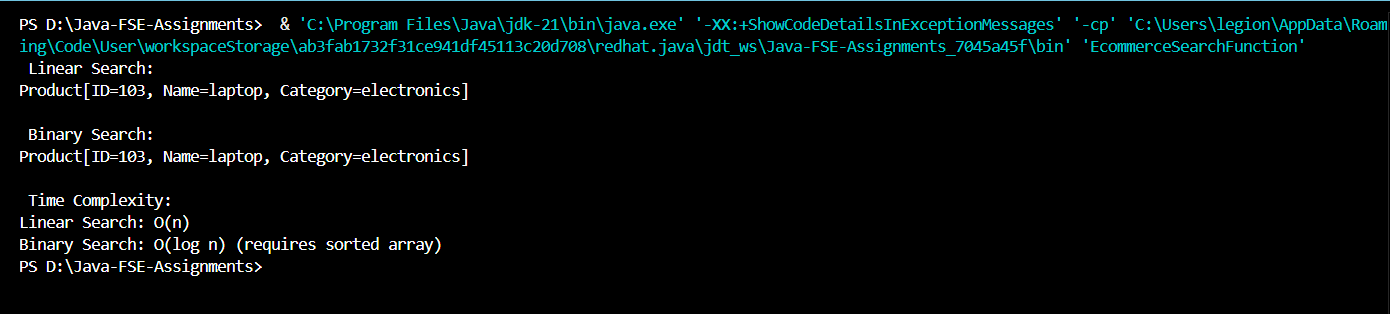
        System.out.println("Linear Search: O(n)");

        System.out.println("Binary Search: O(log n) (requires sorted array)");

    }

}

**Output :**

 **Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data  
  
**Java Code:**public class FinancialForecasting {

    public static double predictFutureValue(double currentValue, double growthRate, int years) {

        if (years == 0) {

            return currentValue;

        }

        return predictFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

    }

    public static double predictMemoized(double currentValue, double growthRate, int years, double[] memo) {

        if (years == 0) return currentValue;

        if (memo[years] != 0) return memo[years];

        return memo[years] = predictMemoized(currentValue \* (1 + growthRate), growthRate, years - 1, memo);

    }

    public static void main(String[] args) {

        double initialInvestment = 10000;

        double annualGrowthRate = 0.1;

        int futureYears = 5;

        double futureValue = predictFutureValue(initialInvestment, annualGrowthRate, futureYears);

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

        double[] memo = new double[futureYears + 1];

        double futureValueOptimized = predictMemoized(initialInvestment, annualGrowthRate, futureYears, memo);

        System.out.printf(" Predicted value after %d years (memoized): %.2f\n", futureYears, futureValueOptimized);

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

        System.out.println("Recursive: O(n) - one call per year");

        System.out.println("Memoized: O(n) - avoids redundant calculations");

    }

}

**Output :**