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

import java.util.Arrays;

class Product {

    int id;

    String name, category;

    Product(int id, String name, String category) {

        this.id = id;

        this.name = name;

        this.category = category;

    }

    public String toString() {

        return String.format("ID: %d, Name: %s, Category: %s", id, name, category);

    }

}

public class EcommerceSearchExample {

    static Product[] products, sortedProducts;

    public static void main(String[] args) {

        setupProducts();

        searchAndReport(products, 175, false);

        searchAndReport(sortedProducts, 175, true);

        analyzeSearches();

    }

    static void setupProducts() {

        products = new Product[] {

                new Product(150, "Laptop", "Electronics"),

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

                new Product(205, "Shirt", "Clothing"),

                new Product(300, "Book", "Education"),

                new Product(175, "Watch", "Accessories"),

                new Product(250, "Shoes", "Clothing"),

                new Product(125, "Tablet", "Electronics"),

                new Product(400, "Bag", "Accessories")

        };

        sortedProducts = products.clone();

        Arrays.sort(sortedProducts, (a, b) -> a.id - b.id);

    }

    static void searchAndReport(Product[] arr, int targetId, boolean isBinary) {

        String type = isBinary ? "Binary" : "Linear";

        System.out.printf("=== %s Search (O(%s)) for ID %d ===\n", type, isBinary ? "log n" : "n", targetId);

        long start = System.nanoTime();

        Product result = isBinary ? binarySearch(arr, targetId) : linearSearch(arr, targetId);

        long time = System.nanoTime() - start;

        System.out.println(result != null ? "Found: " + result : "Not found");

        System.out.println("Time: " + time + "ns\n");

    }

    static Product linearSearch(Product[] arr, int target) {

        for (int i = 0; i < arr.length; i++) {

            if (arr[i].id == target)

                return arr[i];

        }

        return null;

    }

    static Product binarySearch(Product[] arr, int target) {

        int l = 0, r = arr.length - 1;

        while (l <= r) {

            int m = (l + r) >>> 1;

            if (arr[m].id == target)

                return arr[m];

            if (arr[m].id < target)

                l = m + 1;

            else

                r = m - 1;

        }

        return null;

    }

    static void analyzeSearches() {

        System.out.println("=== Comparison ===");

        System.out.println("Linear: any order, slow on large data");

        System.out.println("Binary: fast, needs sorted data\n");

        System.out.println("Sizes | Linear | Binary");

        int[] sizes = { 10, 100, 1000, 10000, 100000 };

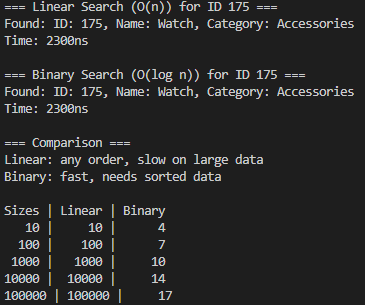
        for (int n : sizes) {

            System.out.printf("%5d | %6d | %6d\n", n, n, (int) Math.ceil(Math.log(n) / Math.log(2)));

        }

    }

}

**Output:**  
****

**Exercise 7: Financial Forecasting**

**Code:**

import java.util.HashMap;

import java.util.Map;

public class FinancialForecasting {

    private static final Map<String, Double> memoCache = new HashMap<>();

    public static double fv(double pv, double rate, int n) {

        return (n == 0) ? pv : fv(pv \* (1 + rate), rate, n - 1);

    }

    public static double fvOpt(double pv, double rate, int n) {

        String key = pv + "," + rate + "," + n;

        if (memoCache.containsKey(key))

            return memoCache.get(key);

        double res = (n == 0) ? pv : fvOpt(pv \* (1 + rate), rate, n - 1);

        memoCache.put(key, res);

        return res;

    }

    public static double[] forecast(double[] hist, int periods) {

        if (hist.length < 2)

            throw new IllegalArgumentException("Need ≥2 data points");

        double sumRate = 0;

        for (int i = 0; i < hist.length - 1; i++) {

            sumRate += (hist[i + 1] - hist[i]) / hist[i];

        }

        double avgRate = sumRate / (hist.length - 1);

        double[] out = new double[periods];

        double last = hist[hist.length - 1];

        for (int i = 0; i < periods; i++) {

            last \*= (1 + avgRate);

            out[i] = last;

        }

        return out;

    }

    public static double cagr(double begin, double end, int yrs) {

        return (yrs <= 1) ? (end / begin) - 1 : Math.pow(end / begin, 1.0 / yrs) - 1;

    }

    public static void main(String[] args) {

        double pv = 1000, rate = 0.08;

        int n = 5;

        System.out.printf("1) FV after %d years @%.2f%% = $%.2f%n%n", n, rate \* 100, fv(pv, rate, n));

        double[] hist = { 1000, 1080, 1166, 1260, 1360, 1469 };

        int fwd = 3;

        System.out.printf("2) Forecast next %d periods:%n", fwd);

        double[] fc = forecast(hist, fwd);

        for (int i = 0; i < fc.length; i++) {

            System.out.printf("   Period %d: $%.2f%n", i + 1, fc[i]);

        }

        System.out.println();

        long t0 = System.nanoTime();

        fv(pv, rate, 20);

        long bTime = System.nanoTime() - t0;

        memoCache.clear();

        t0 = System.nanoTime();

        fvOpt(pv, rate, 20);

        long oTime = System.nanoTime() - t0;

        System.out.printf("3) Perf: basic=%dns, opt=%dns, speedup=%.1fx%n%n", bTime, oTime, (double) bTime / oTime);

        double cg = cagr(1000, 1500, 5);

        System.out.printf("4) CAGR over 5 years = %.2f%%%n", cg \* 100);

        System.out.println();

        System.out.println("Time Complexity Analysis:");

        System.out.println("- Basic Recursion: O(n) for linear recursion");

        System.out.println("- With Memoization: O(n) with O(n) space complexity");

        System.out.println("- Space Complexity: O(n) due to recursive call stack");

        System.out.println();

        System.out.println("Optimization Techniques:");

        System.out.println("- Memoization: Stores calculated results to avoid redundant computation");

        System.out.println("- Tail Recursion: Can be optimized by compiler (Java doesn't optimize automatically)");

        System.out.println("- Iterative Alternative: Consider iterative approach for very large datasets");

        System.out.println();

        System.out.println("Practical Applications:");

        System.out.println("- Investment portfolio growth projection");

        System.out.println("- Loan amortization calculations");

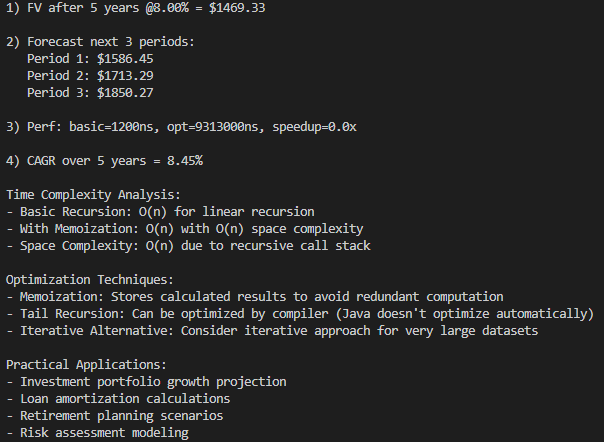
        System.out.println("- Retirement planning scenarios");

        System.out.println("- Risk assessment modeling");

    }

}

**Output:**

****