**Data structures and algorithms**

**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.

**Algorithm:**

1. **Define products** with ID, name, category, and price
2. **Create test data** of 1000 sample products
3. **Implement linear search** that checks each product sequentially
4. **Implement binary search** that requires sorted data and splits search range
5. **Compare performance** by timing searches at different positions
6. **Display results** showing binary search's speed advantage.

Code:

import java.util.Arrays;

import java.util.Comparator;

class Product {

int id;

String name;

String category;

int price;

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

this.id = id;

this.name = name;

this.category = category;

this.price = price;

}

@Override

public String toString() {

return String.format("Product ID: %d, Name: %s, Category: %s, Price: %d",

id, name, category, price);

}

}

public class ProductSearch {

private static final int WARMUP = 10;

private static final int TEST\_RUNS = 100;

private static double[] linearTimes = new double[3];

private static double[] binaryTimes = new double[3];

public static Product searchLinear(Product[] products, int searchId, int index) {

long totalTime = 0;

Product found = null;

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

for (Product p : products) {

if (p.id == searchId) {

found = p;

break;

}

}

}

for (int i = 0; i < TEST\_RUNS; i++) {

long start = System.nanoTime();

for (Product p : products) {

if (p.id == searchId) {

found = p;

break;

}

}

totalTime += System.nanoTime() - start;

}

double avgTime = (totalTime / TEST\_RUNS) / 1000.0;

linearTimes[index] = avgTime;

System.out.printf("Linear: %.3f μs%n", avgTime);

return found;

}

public static Product searchBinary(Product[] products, int searchId, int index) {

long totalTime = 0;

Product found = null;

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

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

while (left <= right) {

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

if (products[mid].id == searchId) {

found = products[mid];

break;

}

if (products[mid].id < searchId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

for (int i = 0; i < TEST\_RUNS; i++) {

long start = System.nanoTime();

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

while (left <= right) {

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

if (products[mid].id == searchId) {

found = products[mid];

break;

}

if (products[mid].id < searchId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

totalTime += System.nanoTime() - start;

}

double avgTime = (totalTime / TEST\_RUNS) / 1000.0;

binaryTimes[index] = avgTime;

System.out.printf("Binary: %.3f μs%n", avgTime);

return found;

}

public static void printAnalysis() {

System.out.println("\n=== Performance Analysis ===");

System.out.println("Search Type | First Item | Middle Item | Last Item");

System.out.printf("Linear | %7.3f μs | %8.3f μs | %7.3f μs%n",

linearTimes[0], linearTimes[1], linearTimes[2]);

System.out.printf("Binary | %7.3f μs | %8.3f μs | %7.3f μs%n",

binaryTimes[0], binaryTimes[1], binaryTimes[2]);

double linearAvg = (linearTimes[0] + linearTimes[1] + linearTimes[2]) / 3;

double binaryAvg = (binaryTimes[0] + binaryTimes[1] + binaryTimes[2]) / 3;

System.out.printf("%nAverage Times: Linear %.3f μs vs Binary %.3f μs%n", linearAvg, binaryAvg);

System.out.println("Binary search is " + String.format("%.1f", linearAvg/binaryAvg) + "x faster on average");

}

public static void main(String[] args) {

Product[] products = new Product[1000];

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

String cat = (i % 5 == 0) ? "Gadgets" : "Fashion";

products[i] = new Product(i+1, "Item-"+(i+1), cat, 100+(i\*10));

}

Product[] sorted = Arrays.copyOf(products, products.length);

Arrays.sort(sorted, Comparator.comparingInt(p -> p.id));

int[] testIds = {1, 500, 999};

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

System.out.println("\nSearch ID: " + testIds[i]);

Product linearResult = searchLinear(products, testIds[i], i);

if (linearResult != null) {

System.out.println(linearResult);

}

Product binaryResult = searchBinary(sorted, testIds[i], i);

if (binaryResult != null) {

System.out.println(binaryResult);

}

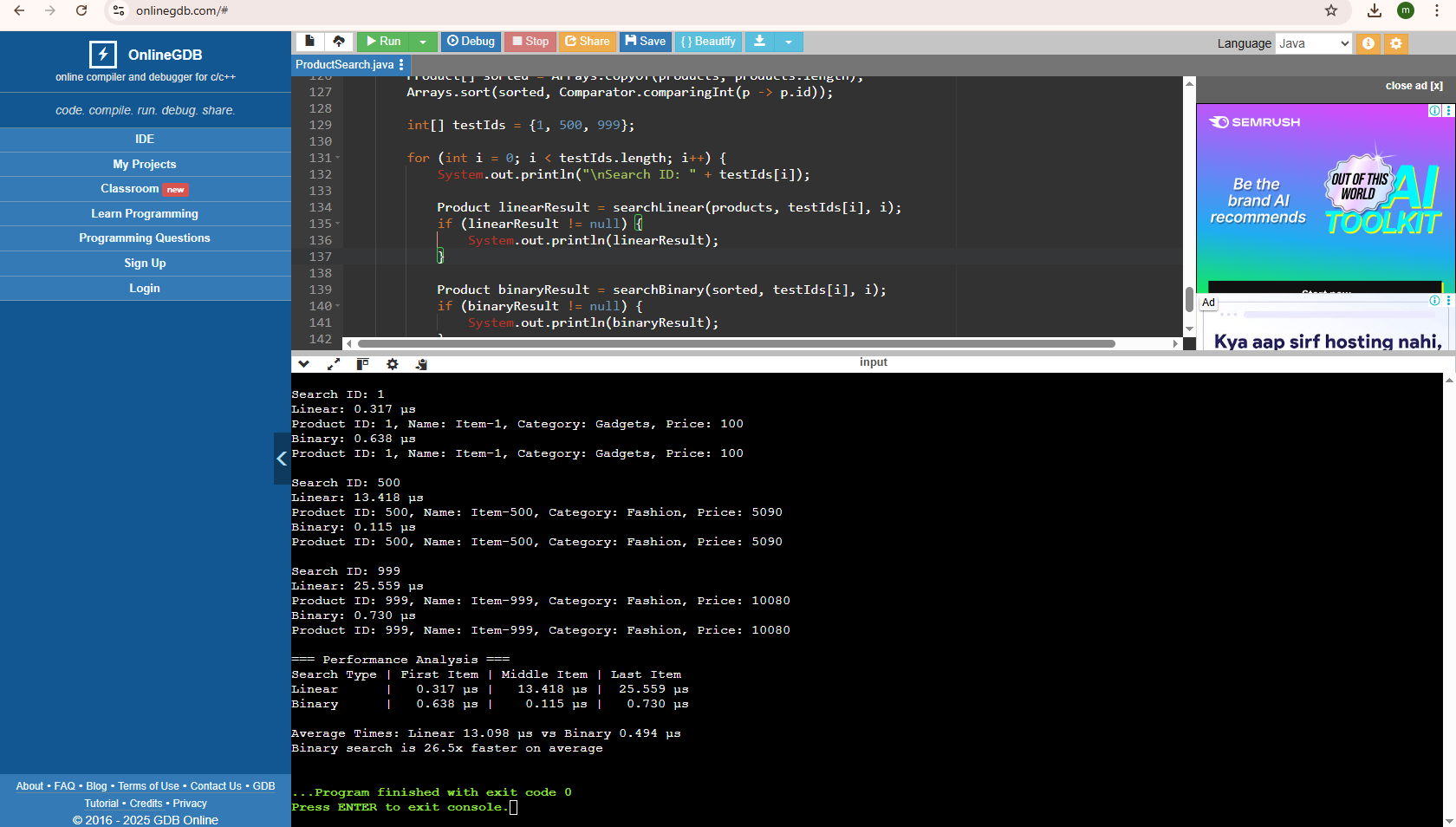
}

printAnalysis();

}

}

**Output:**

****

**Data structures and algorithms**

**Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Algorithm:**

1. Define method that takes current value, growth rate, and years
2. Check base case - return current value if years = 0
3. Recursive case - multiply value by (1 + growth rate) and reduce years by 1
4. Initialize parameters with sample investment values
5. Call recursive method to calculate future value
6. Print formatted result showing growth projection

**Code:**

public class FinancialForecast {

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 void main(String[] args) {

double initialAmount = 5000;

double annualGrowth = 0.05;

int projectionYears = 8;

double futureValue = predictFutureValue(initialAmount, annualGrowth, projectionYears);

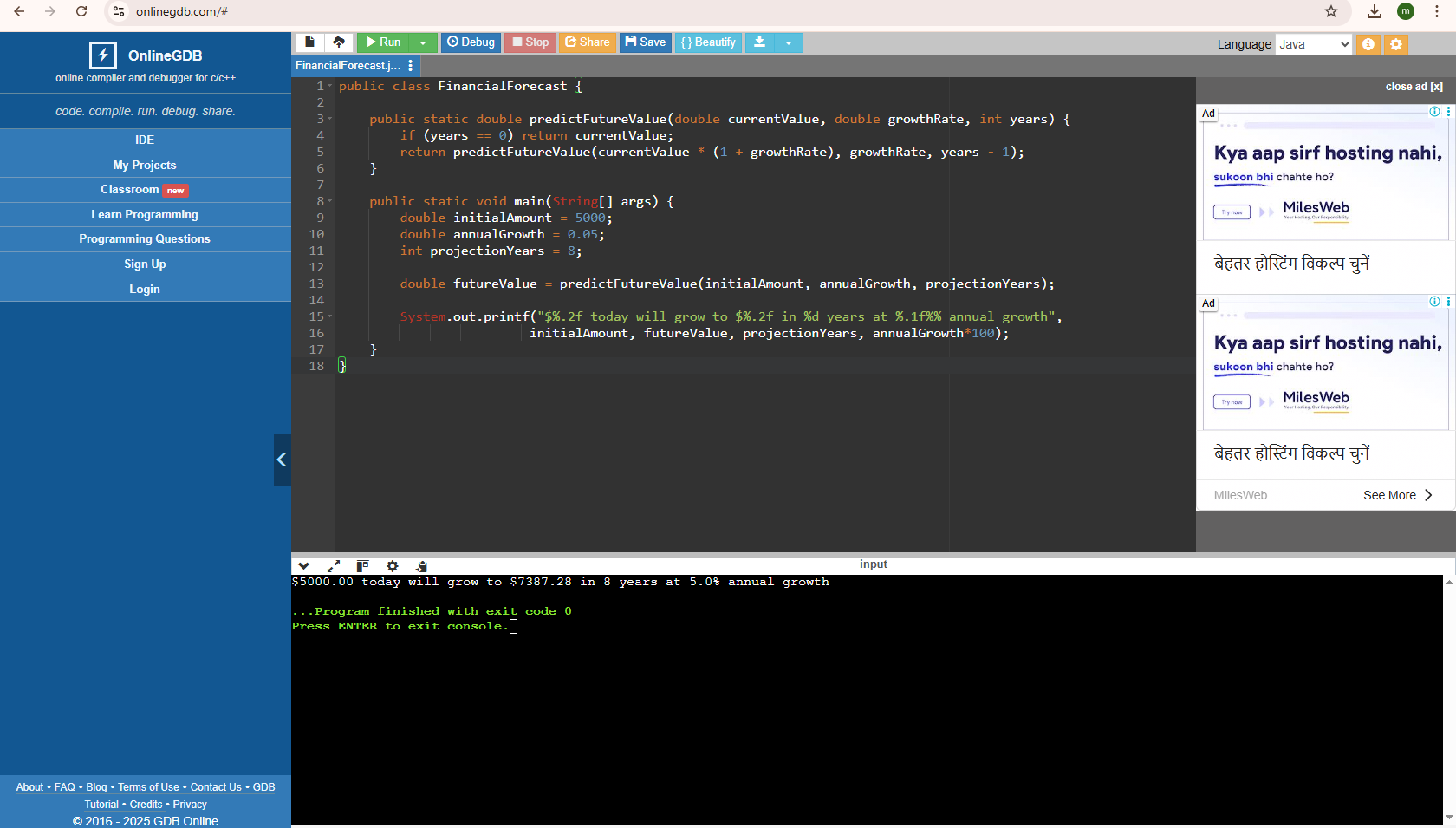
System.out.printf("$%.2f today will grow to $%.2f in %d years at %.1f%% annual growth",

initialAmount, futureValue, projectionYears, annualGrowth\*100);

}

}

**Output:**

****