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

1. What is Big O Notation?  
   Big O notation expresses the **upper bound** of an algorithm's running time. It helps developers:

* Compare performance
* Predict scalability
* Choose efficient algorithms

TIME COMPLEXITY

Best Case Average Case Worst Case

Linear Search- O(1) O(n) O(n)

Binary Search O(1) O(log n) O(log n)

**Code**

import java.util.Arrays;

class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String cat) {

this.productId = id;

this.productName = name;

this.category = cat;

}

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

class SearchEngine {

// Linear Search

public static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Binary Search (requires sorted array)

public static Product binarySearch(Product[] products, int targetId) {

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

if (products[mid].productId == targetId) {

return products[mid];

} else if (products[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

}

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(102, "Laptop", "Electronics"),

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

new Product(309, "Smartphone", "Electronics"),

new Product(401, "Book", "Education")

};

// Sort products by productId before binary search

Arrays.sort(products, (a, b) -> a.productId - b.productId);

// Linear Search

Product result1 = SearchEngine.linearSearch(products, 205);

System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));

// Binary Search

Product result2 = SearchEngine.binarySearch(products, 205);

System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));

}

}

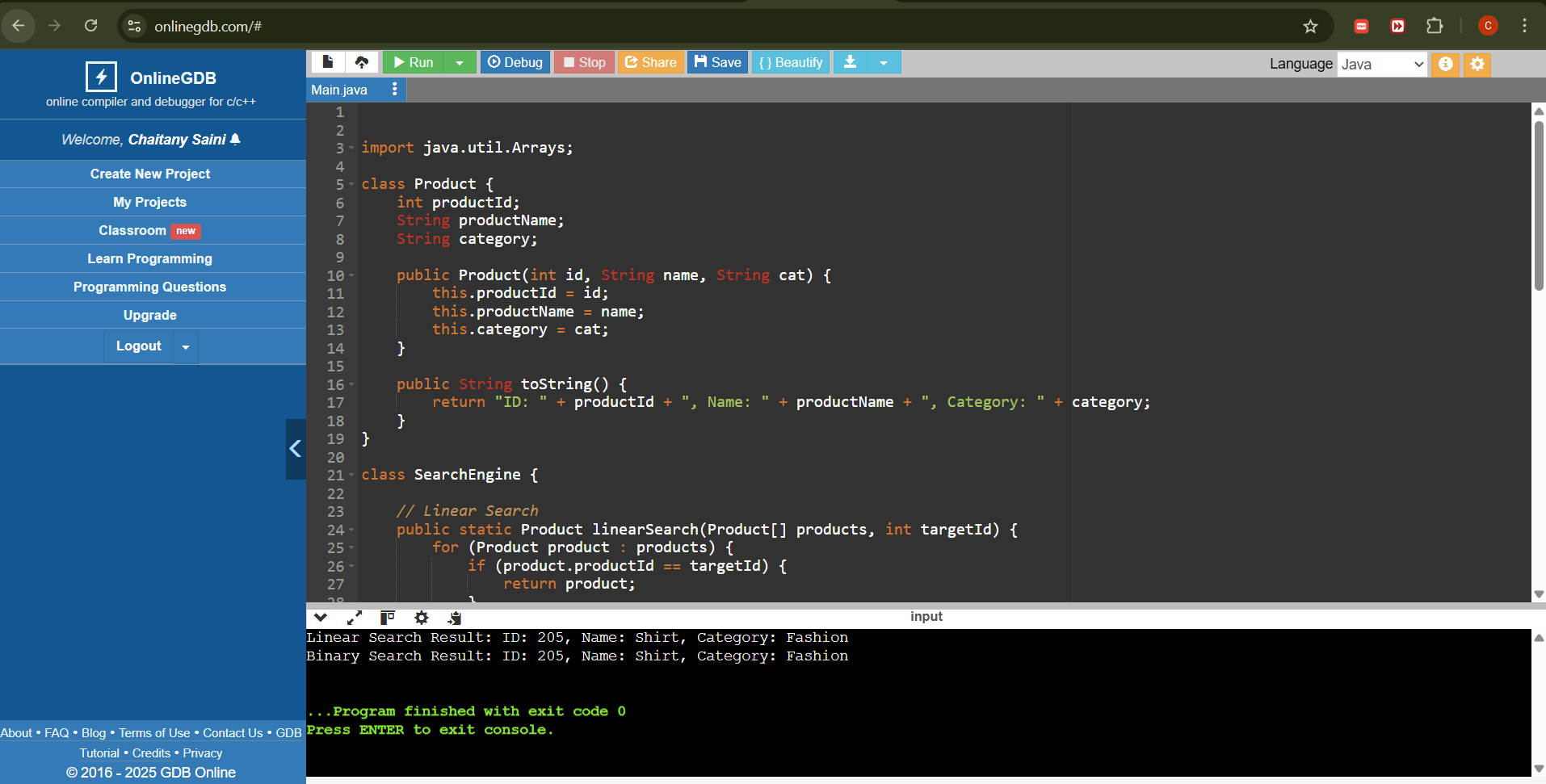
Time Complexity

Linear Search-O(n)

Binary Search-O(log n)

Note-For an e-commerce platform Binary Search is more efficient due to large product listings.

***OUTPUT***

******

***Exercise 7: Financial Forecasting***

***Recursion***Recursion is a method where a function **calls itself** to solve smaller instances of the same problem.  
It’s especially useful in problems that have **repetitive substructure**

CODE

// Financial Forecasting Tool using Recursion

public class Main {

// Recursive method to calculate future value

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

if (years == 0) {

return currentValue;

}

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

}

// Optimized recursive method using memoization

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

if (years == 0) {

return currentValue;

}

if (memo[years] != 0) {

return memo[years];

}

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

return memo[years];

}

public static void main(String[] args) {

double initialValue = 10000.0; // Starting amount

double growthRate = 0.08; // 8% annual growth

int years = 10; // Forecast for 10 years

// Standard Recursive Forecast

double futureValue = forecastValueRecursive(initialValue, growthRate, years);

System.out.printf("Recursive Forecast (10 years): ₹%.2f\n", futureValue);

// Optimized Forecast with Memoization

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

double optimizedFuture = forecastValueMemo(initialValue, growthRate, years, memo);

System.out.printf("Optimized Forecast with Memoization: ₹%.2f\n", optimizedFuture);

}

}

|  | ***Time Complexity*** |  |
| --- | --- | --- |

Recursive -O(n)

After optimizing it with memorization- O(n)

* + Memoization ensures efficiency of the recursive solution to avoid excessive computation.

