**Module 1: ALGORITHMS\_DATA STRUCTURES**

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

**Step 1:** Understand Asymptotic Notation

Big O Notation

* Big O describes how the runtime or space grows as the input size increases.
* Helps compare algorithm efficiency without running code.

**Step 2:** Setup – Product Class

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

}

**Step 3:** Implementation – Linear & Binary Search

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

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

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

};

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

Product result1 = linearSearch(products, 103);

display(result1);

java.util.Arrays.sort(products, (a, b) -> Integer.compare(a.productId, b.productId));

System.out.println("\nBinary Search:");

Product result2 = binarySearch(products, 103);

display(result2);

}

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

for (Product p : products) {

if (p.productId == targetId)

return p;

}

return null;

}

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

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

while (left <= right) {

int mid = (left + right) / 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 static void display(Product p) {

if (p != null) {

System.out.println("Found Product:");

System.out.println("ID: " + p.productId + ", Name: " + p.productName + ", Category: " + p.category);

} else {

System.out.println("Product not found.");

}

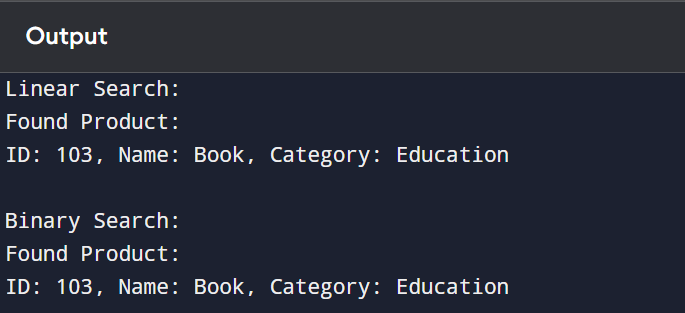
}

}

**Step 4:** Analysis – Time Complexity

* Linear Search: O(n) – Slower for large datasets; checks each item one by one.
* Binary Search: O(log n) – Much faster but requires a sorted list.

**Output:**

****

**Exercise 7: Financial Forecasting**

**Step 1:** Recursive Algorithms

Recursion is a programming technique where a method calls itself to solve smaller subproblems of a bigger task. It simplifies problems like computing future value over time, especially when the process repeats at regular intervals (e.g., compound growth).

**Step 2 & 3:** Recursive Implementation – Future Value Prediction

Formula:

Future Value = Present Value × (1 + rate)^n

Code:

public class Main {

public static void main(String[] args) {

double presentValue = 1000.0;

double rate = 0.10; // 10% growth rate

int years = 5;

double futureValue = forecast(presentValue, rate, years);

System.out.printf("Future Value after %d years = %.2f\n", years, futureValue);

}

public static double forecast(double value, double rate, int years) {

if (years == 0)

return value;

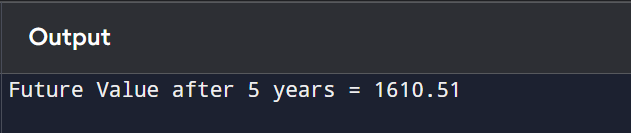
return forecast(value \* (1 + rate), rate, years - 1);

} }

**Step 4:** Analysis – Time Complexity

* Time Complexity: O(n), where *n* is the number of years.
* Space Complexity: O(n) due to recursive call stack.

**Output:**

****