**WEEK-1 DEEP SKILLING**

**2.DATA STRUCTURES AND ALGORITHMS**

**Exercise 2: 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.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**CODE:**

**package** week\_1;

**import** java.util.Arrays;

**import** java.util.Comparator;

**class** Product {

**int** productId;

String productName;

String category;

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

**this**.productId = productId;

**this**.productName = productName;

**this**.category = category;

}

**public** String toString() {

**return** "[" + productId + ", " + productName + ", " + category + "]";

}

}

**public** **class** ECommerceSearch {

**public** **static** **void** main(String[] args) {

Product[] products = {

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

**new** Product(105, "Shoes", "Fashion"),

**new** Product(103, "Book", "Stationery"),

**new** Product(102, "Mobile", "Electronics"),

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

};

System.***out***.println("Linear Search for productId 103:");

Product result1 = *linearSearch*(products, 103);

System.***out***.println(result1 != **null** ? result1 : "Not Found");

Arrays.*sort*(products, Comparator.*comparingInt*(p -> p.productId));

System.***out***.println("\nBinary Search for productId 103:");

Product result2 = *binarySearch*(products, 103);

System.***out***.println(result2 != **null** ? result2 : "Not Found");

}

**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** low = 0, high = products.length - 1;

**while** (low <= high) {

**int** mid = (low + high) / 2;

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

**return** products[mid];

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

low = mid + 1;

} **else** {

high = mid - 1;

}

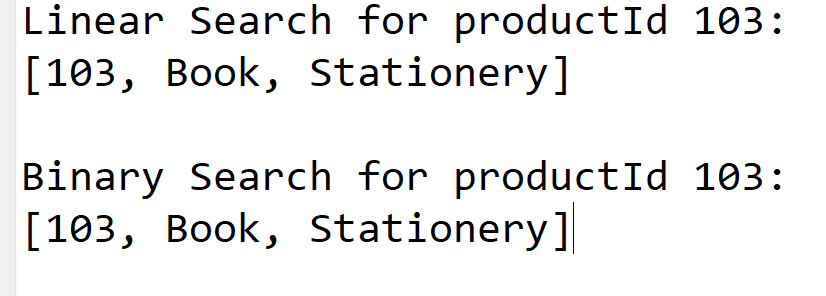
}

**return** **null**;

}

}

**OUTPUT:**

****

**Exercise 7: Financial Forecasting**

**Scenario:**

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

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**CODE:**

**package** week\_1;

**public** **class** FinancialForecasting {

**public** **static** **void** main(String[] args) {

**double** presentValue = 10000;

**double** growthRate = 0.10;

**int** years = 5;

**double** futureValue = *calculateFutureValue*(presentValue, growthRate, years);

System.***out***.println("Predicted future value after" + years + " years: ₹" + futureValue);

}

**public** **static** **double** calculateFutureValue(**double** value, **double** rate, **int** years) {

**if** (years == 0) {

**return** value;

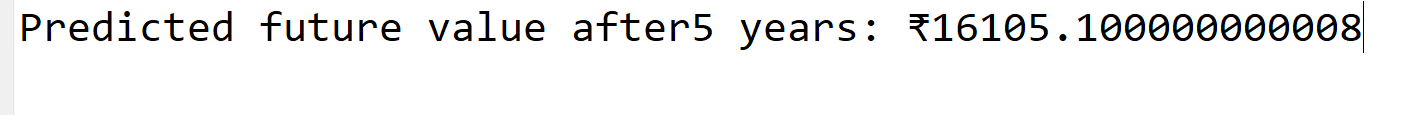
}

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

}

}

**OUTPUT:**

****