Exercise 1: 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.**
      Big O Notation:
      It describes how the runtime or space requirement of an algorithm grows with input size n.
      Helps to choose the most efficient algorithm based on worst case growth.

    - **Describe the best, average, and worst-case scenarios for search operations.**

Case	Description
1. Best Case	Fastest scenario
2. Average Case	Typical runtime over many inputs
3. Worst Case	Slower scenario

2. **Setup:**
    - **Create a class Product with attributes for searching, such as productId, productName, and category.**

      Product Class: Product.java file

```
public 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 + ")";
  }
}
```

3. **Implementation:**
    - Implement linear search and binary search algorithms.
    - Store products in an array for linear search and a sorted array for binary search.

ProductSearch.java file

```java
import java.util.Arrays;
import java.util.Comparator;

public class ProductSearch {

  // Linear Search
  public static Product linearSearch(Product[] products, int targetId) {
    for (Product product : products) {
      if (product.productId == targetId) {
        return product;
      }
    }
    return null;
  }

  // Binary Search
  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;
  }

  // Main method to test
  public static void main(String[] args) {
    Product[] products = {
      new Product(101, "Laptop", "Electronics"),
      new Product(205, "Shoes", "Fashion"),
      new Product(309, "Refrigerator", "Appliances"),
      new Product(150, "Watch", "Accessories"),
      new Product(120, "Mobile", "Electronics")
    };
```

```java
// Binary Search to Sort by productId
    Product[] sortedProducts = Arrays.copyOf(products, products.length);
    Arrays.sort(sortedProducts, Comparator.comparingInt(p -> p.productId));

    System.out.println("Linear Search for ID 205:");
    Product result1 = linearSearch(products, 205);
    System.out.println(result1 != null ? result1 : "Not Found");

    System.out.println("\nBinary Search for ID 205:");
    Product result2 = binarySearch(sortedProducts, 205);
    System.out.println(result2 != null ? result2 : "Not Found");
  }
}
```

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                Code - src
PS C:\Users\sanga\OneDrive\Desktop\Cognizant\VsCode\Week1_Data Structure and Algorithm\E-commerce Platform Search Function\src> javac Product.java ProductSearch.java
PS C:\Users\sanga\OneDrive\Desktop\Cognizant\VsCode\Week1_Data Structure and Algorithm\E-commerce Platform Search Function\src> java ProductSearch
Linear Search for ID 205:
[205] Shoes (Fashion)

Binary Search for ID 205:
[205] Shoes (Fashion)
PS C:\Users\sanga\OneDrive\Desktop\Cognizant\VsCode\Week1_Data Structure and Algorithm\E-commerce Platform Search Function\src>
```

4. **Analysis:**

- **Compare the time complexity of linear and binary search algorithms.**

Algorithm	Time Complexity	Sorting Requirement	Suitable
Linear Search	O(n)	No	For Small / Unsorted datasets
Binary Search	O(log n)	Yes	Large / Sorted datasets

- **Discuss which algorithm is more suitable for your platform and why.**
  For Large scale e-commerce platforms, where fast search is critical, Binary search is more suitable. But only if data is pre sorted or stored in a Tree / Index structure.

Exercise 2: 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.**

      Recursion:

      Recursion is when a function calls itself to solve smaller sub-problems of a larger task.

      Example:

      Financial Forecasting with recursion

      - Initial amount: P
      - Annual growth rate: r (e.g., 5% - 0.05)
      - Number of years: n

      Formula:

      FutureValue = P x (1 + r)^n

      Can return recursively,

      FV(n) = FV(n - 1) x (1 + r)

2. **Setup:**
    - **Create a method to calculate the future value using a recursive approach.**

```
// Recursive method to calculate future value
public class FinancialForecast
{
  public static double futureValue(double principal, double rate, int years)
  {
    if (years == 0)
    {
      return principal;  // base case
    }
    return futureValue(principal, rate, years - 1) * (1 + rate);
   }
```

3. **Implementation:**

   - **Implement a recursive algorithm to predict future values based on past growth rates.**

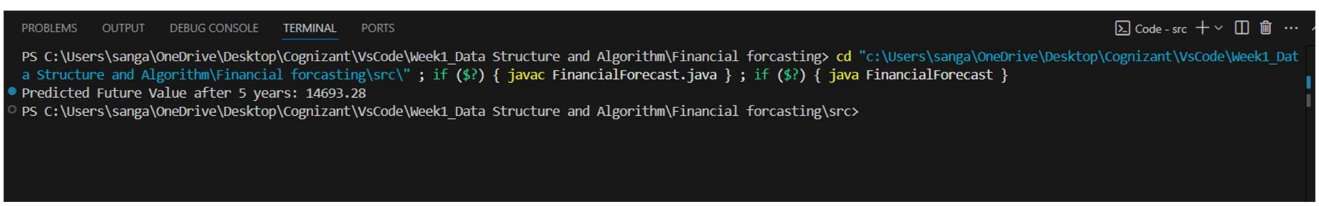
FinancialForcast.java

```
public class FinancialForecast {


  public static double futureValue(double principal, double rate, int years) {
    if (years == 0) {
      return principal; // Base case: No more growth
    }
    return futureValue(principal, rate, years - 1) * (1 + rate);
  }


  public static void main(String[] args) {
    double initialAmount = 10000.0;   // Starting amount
    double annualGrowthRate = 0.08;   // 8% growth rate
    int years = 5;                    // Forecast period
    double predictedValue = futureValue(initialAmount, annualGrowthRate, years);
    System.out.printf("Predicted value after %d years: ₹%.2f\n", years, predictedValue);
  }
}
```

Output:



4. **Analysis:**

   - **Discuss the time complexity of your recursive algorithm.**

Aspect	**Value**
Time Complexity	**O(n)** — one recursive call per year
Space Complexity	**O(n)** — due to recursive call stack

   - **Explain how to optimize the recursive solution to avoid excessive computation.**

     Can optimize using three methods,

     1. Iterative Approach [Recommended for Real Apps]

     2. Tail Recursion [Where Supported – java doesn't support tail call optimization]

     3. Memorization [For repeated Calculations]