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.

1. **Understand Asymptotic Notation**

   The efficiency of an algorithm depends on the amount of time, storage and other resources required to execute the algorithm. The efficiency is measured with the help of asymptotic notations. Asymptotic notation is a way to describe the performance of algorithms, specifically how their runtime or space usage grows as the input size increases

2. **Setup:**

   Product.java

```
class Product {

    int productId;

    String productName;

    String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

}
```

3. **Implementation:**

```
import java.util.*;
public class Main {
    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 main(String[] args) {
        Product[] products = {
            new Product(3, "Phone", "Electronics"),
            new Product(1, "Book", "Education"),
            new Product(2, "Shirt", "Clothing")
        };

        Product p = linearSearch(products, 2);
        if (p != null)
            System.out.println("Linear Search Found: " +
p.productName);
        else
            System.out.println("Linear Search: Not Found");

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

        Product pp = binarySearch(products, 3);
        if (pp != null)
            System.out.println("Binary Search Found: " +
pp.productName);
        else
            System.out.println("Binary Search: Not Found");
    }
}
class Product {
    int productId;
    String productName;
    String category;

    Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }
}
```

4. `Output :`

```
Linear Search Found: Shirt
Binary Search Found: Phone
```

5. **Analysis :**

Linear Search :	**Binary Search :**
Time Complexity : Worst case : O(N) Best Case : O(1) Average Case : O(N) Space Complexity : Worst case : O(1) Best Case : O(1) Average Case : O(1)	Time Complexity : Worst case : O(log(N)) Best Case : O(1) Average Case : O(log(N)) Space Complexity : Worst case : O(1) Best Case : O(1) Average Case : O(1)

2. Financial Forecasting

Scenario:

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

1. **Understand Recursive Algorithms:**

   The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Recursion helps in logic building. Recursive thinking helps in solving complex problems by breaking them into smaller subproblems.

2. **Setup:**

```
public static double help(double presentValue, double rate, int years) {
  if (years == 0) {
    return presentValue;
  }
  return (1 + rate) * help(presentValue, rate, years - 1);
}
```

3. **Implementation:**

```
public class Main {
  public static double help(double presentValue, double rate, int years) {
    if (years == 0) {
      return presentValue;
    }
    return (1 + rate) * help(presentValue, rate, years - 1);
  }
  public static void main(String[] args) {
    double p = 1000;
    double r = 0.055;
    int n = 20;
    double ans = help(p, r, n);
    System.out.println("Future Value after " + n + " years = " + ans);
  }
}
```

4. **Output :**

```
Future Value after 20 years = 2917.7574906040877
```

5. **Analysis:**

Time Complexity : O(N) since it’s a linear recursion