**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.

**Understand Asymptotic Notation**

* **Big O Notation**: Describes the **upper bound** of an algorithm’s performance as input size grows. Helps compare efficiency.
* **Search Time Complexities**:
  + **Linear Search**:
    - Best: O(1) (first item)
    - Average/Worst: O(n)
  + **Binary Search**:
    - Best: O(1) (middle)
    - Average/Worst: O(log n)
    - Requires a **sorted array**

**Implementation:**

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;

    }

    @Override

    public String toString() {

        return "Product{" +

                "productId=" + productId +

                ", productName='" + productName + '\'' +

                ", category='" + category + '\'' +

                '}';

    }

}

public class Main {

**// Linear Search**

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

        for (Product p : products) {

            if (p.productId == targetId) {

                return p;

            }

        }

        return null;

    }

**// Binary Search (array must be sorted)**

    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;

    }

**// Main method**

    public static void main(String[] args) {

        Product[] products = {

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

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

            new Product(103, "Shoes", "Footwear"),

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

        };

**// Sorting array for binary search**

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

**// Linear Search**

        Product resultLinear = linearSearch(products, 103);

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

        System.out.println(resultLinear != null ? resultLinear : "Product not found");

**// Binary Search**

        Product resultBinary = binarySearch(products, 103);

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

        System.out.println(resultBinary != null ? resultBinary : "Product not found");

    }

}

**OUTPUT**

A screenshot of a computer program

AI-generated content may be incorrect.

**Analysis**

| **Algorithm** | **Time Complexity** | **Needs Sorted Data** |
| --- | --- | --- |
| Linear Search | O(n) | No |
| Binary Search | O(log n) | Yes |

**Exercise 7: Financial Forecasting**

**Scenario:**

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

**Understand Recursive Algorithms**

* **Recursion** is a programming technique where a method calls itself to solve smaller subproblems.
* It is helpful in problems like **financial forecasting**, where future values depend on past values in a repetitive pattern.

**Setup – Method Signature**

We'll write a method to calculate the **future value** based on:

* Initial value
* Growth rate
* Number of years (n)

**Java Implementation**

public class FinancialForecast {

**// Recursive method to calculate future value**

public static double futureValue(double initialValue, double growthRate, int years) {

if (years == 0) {

return initialValue; // Base case: no growth in year 0

}

return futureValue(initialValue, growthRate, years - 1) \* (1 + growthRate);

}

public static void main(String[] args) {

double initialInvestment = 1000.0;

double annualGrowthRate = 0.10; // 10% growth

int years = 5;

double result = futureValue(initialInvestment, annualGrowthRate, years);

System.out.printf("Future value after %d years = ₹%.2f%n", years, result);

}

}

**OUTPUT**

A screenshot of a computer program

AI-generated content may be incorrect.

**Analysis**

**⏱ Time Complexity**

* **T(n) = T(n-1) + O(1)** → **O(n)**  
  One recursive call per year.