**Digital Nurture 4.0 – Week 1**

**2.Data structures and Algorithms**

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

**Program:**

I created a class Product with fields such as productId, productName, and category, and implemented both Linear Search and Binary Search algorithms to search products by ID. The products are stored in an array for linear search and in a sorted array for binary search. The program demonstrates search performance comparison using asymptotic analysis.

**E-CommerceSearchExample.java**

import java.util.Arrays;

import java.util.Scanner;

class Product implements Comparable<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 void display() {

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

    }

    @Override

    public int compareTo(Product other) {

        return Integer.compare(this.productId, other.productId);

    }

}

public class ECommerceSearchExample {

   // Linear Search

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

        for (int i = 0; i < products.length; i++) {

            if (products[i].productId == targetId) {

                return i;

            }

        }

        return -1;

    }

    // Binary Search

    public static int 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 mid;

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

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return -1;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(105, "Shoes", "Footwear"),

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

            new Product(103, "Laptop", "Electronics"),

            new Product(102, "T-shirt", "Clothing"),

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

        };

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Product ID to search: ");

        int targetId = sc.nextInt();

        int indexLinear = linearSearch(products, targetId);

        System.out.println("\n--- Linear Search ---");

        if (indexLinear != -1) {

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

            products[indexLinear].display();

        } else {

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

        }

        Arrays.sort(products);

        int indexBinary = binarySearch(products, targetId);

        System.out.println("\n--- Binary Search ---");

        if (indexBinary != -1) {

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

            products[indexBinary].display();

        } else {

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

        }

    }

}

**Output:**

**A computer screen with white text

AI-generated content may be incorrect.**

**Analysis: Time Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Best Case | Average Case | Worst Case |
| Linear Search | O(1) | O(n/2) → O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

**Discussion:**

* Linear Search is simple and works on unsorted arrays, but is inefficient for large datasets.
* Binary Search is faster (logarithmic time) but requires the array to be sorted.
* For an E-commerce platform with a large product catalog, binary search is more efficient, provided data is kept sorted.

**Exercise 7:** Financial Forecasting

**Program:**

I created a recursive method to calculate future financial values based on initial amount and annual growth rate using recursion. This program demonstrates the concept of recursion and explores how it simplifies forecasting calculations over multiple years.

**FinancialForecasting.java**

public class FinancialForecasting {

    public static double forecast(double initialAmount, double growthRate, int years) {

        if (years == 0) {

            return initialAmount;

        }

        return forecast(initialAmount, growthRate, years - 1) \* (1 + growthRate);

    }

    public static void main(String[] args) {

        double initialAmount = 10000;

        double growthRate = 0.08;

        int years = 5;

        double futureValue = forecast(initialAmount, growthRate, years);

        System.out.println("Initial Amount: ₹" + initialAmount);

        System.out.println("Annual Growth Rate: " + (growthRate \* 100) + "%");

        System.out.println("Forecast for " + years + " years: ₹" + String.format("%.2f", futureValue));

    }

}

**Output:**

