**WEEK – 1 HANDS ON**

**SKILL – Algorithms, Data Structures.**

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:

**//class Product**

public class Product {

      private int productId;

        private String productName;

        private String category;

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

            this.productId = productId;

            this.productName = productName;

            this.category = category;

        }

        public int getProductId() {

            return productId;

        }

        public String getProductName() {

            return productName;

        }

        public String getCategory() {

            return category;

        }

        @Override

        public String toString() {

            return productId + " - " + productName + " (" + category + ")";

        }

}

**//class SearchUtil**

import java.util.Arrays;

import java.util.Comparator;

public class SearchUtil {

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

        for (Product product : products) {

            if (product.getProductId() == targetId) {

                return product;

            }

        }

        return null;

    }

    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;

            int midId = products[mid].getProductId();

            if (midId == targetId) {

                return products[mid];

            } else if (midId < targetId) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public static void sortProducts(Product[] products) {

        Arrays.sort(products, Comparator.comparingInt(Product::getProductId));

    }

}

**//class Main**

public class Main {

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(101, "Shoes", "Fashion"),

            new Product(105, "Watch", "Accessories"),

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

        };

        Product foundLinear = SearchUtil.linearSearch(products, 104);

        System.out.println("Linear search found: " + foundLinear);

        SearchUtil.sortProducts(products);

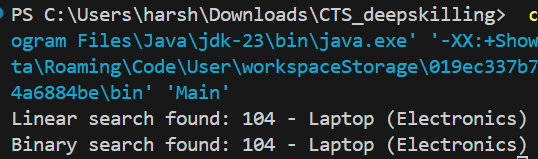
        Product foundBinary = SearchUtil.binarySearch(products, 104);

        System.out.println("Binary search found: " + foundBinary);

    }

}

**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:**

public class FinancialForecasting {

**//Recursion**

    public static double predictFutureValue(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        }

        return predictFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

    }

**// without Recursion**

    public static double predictFutureValueIterative(double presentValue, double rate, int years) {

        double value = presentValue;

        for (int i = 0; i < years; i++) {

            value \*= (1 + rate);

        }

        return value;

    }

    public static void main(String[] args) {

        double presentValue = 1000.0;

        double rate = 0.05;

        int years = 10;

        System.out.println("Using Recurison:");

        double predicted = predictFutureValue(presentValue, rate, years);

        System.out.printf("Predicted future value after %d years: %.2f\n", years, predicted);

        System.out.println("Using Iteration:");

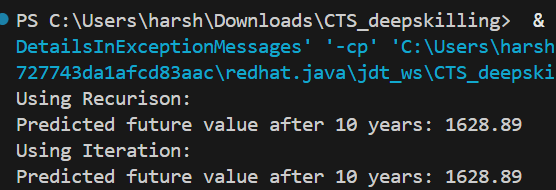
        double predicted1 = predictFutureValueIterative(presentValue, rate, years);

        System.out.printf("Predicted future value after %d years: %.2f\n", years, predicted1);

    }

}

**OUTPUT:**

****