***WEEK 2***

***MANDATORY HANDS ON***

***DATA STRUCTURES AND ALGORITHM***

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

**BinarySearch.java**

package com.ecommerce;

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

public static Product binarySearch(Product[] products, String targetName) {

// Binary Search requires sorted data

Arrays.sort(products, Comparator.comparing(Product::getProductName, String.CASE\_INSENSITIVE\_ORDER));

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int compare = targetName.compareToIgnoreCase(products[mid].getProductName());

if (compare == 0) return products[mid];

else if (compare < 0) right = mid - 1;

else left = mid + 1;

}

return null;

}

}

**LinearSearch.java**

package com.ecommerce;

public class LinearSearch {

public static Product linearSearch(Product[] products, String targetName) {

for (Product p : products) {

if (p.getProductName().equalsIgnoreCase(targetName)) {

return p;

}

}

return null;

}

}

**Main.java**

package com.ecommerce;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product("P001", "Shoes", "Footwear"),

new Product("P002", "Watch", "Accessories"),

new Product("P003", "T-shirt", "Clothing"),

new Product("P004", "Laptop", "Electronics"),

new Product("P005", "Bag", "Travel")

};

// Linear Search

Product result1 = LinearSearch.*linearSearch*(products, "Laptop");

System.***out***.println("Linear Search: " + (result1 != null ? result1 : "Product not found"));

// Binary Search

Product result2 = BinarySearch.*binarySearch*(products, "Laptop");

System.***out***.println("Binary Search: " + (result2 != null ? result2 : "Product not found"));

}

}

**Product.java**

package com.ecommerce;

public class Product {

String productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductName() {

return productName;

}

*@Override*

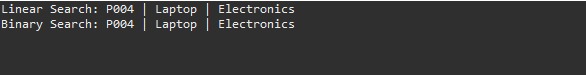
public String toString() {

return productId + " | " + productName + " | " + category;

}

}

**OUTPUT**

****

**Exercise 7: Financial Forecasting**

**FinancialForecast.java**

package com.forecast;

import java.util.Scanner;

public class FinancialForecast {

// Recursive method to calculate future value

public static double forecastRecursive(int year, double initialValue, double growthRate) {

// Base case: 0 years ahead returns current value

if (year == 0) {

return initialValue;

}

// Recursive step

return *forecastRecursive*(year - 1, initialValue, growthRate) \* (1 + growthRate);

}

// Optimized Iterative method to calculate future value

public static double forecastIterative(int year, double initialValue, double growthRate) {

double value = initialValue;

for (int i = 1; i <= year; i++) {

value \*= (1 + growthRate);

}

return value;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

// User input

System.***out***.print("Enter initial value (e.g., investment or revenue): ₹");

double initialValue = scanner.nextDouble();

System.***out***.print("Enter annual growth rate (in %): ");

double growthRatePercent = scanner.nextDouble();

double growthRate = growthRatePercent / 100; // Convert to decimal

System.***out***.print("Enter number of years to forecast: ");

int years = scanner.nextInt();

// Recursive forecast

double recursiveResult = *forecastRecursive*(years, initialValue, growthRate);

System.***out***.printf("\nFuture Value (Recursive) after %d years = ₹%.2f", years, recursiveResult);

// Iterative forecast

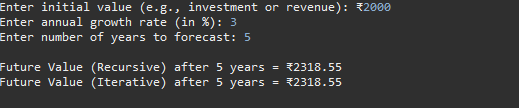
double iterativeResult = *forecastIterative*(years, initialValue, growthRate);

System.***out***.printf("\nFuture Value (Iterative) after %d years = ₹%.2f\n", years, iterativeResult);

}

}

**OUTPUT**

****