**WEEK – 1 : DATA STRUCTURES AND ALGORITHM**

**EXERCISE 1: E-COMMERCE PLATFORM SEARCH FUNCTION**

**AppMenu.java:**

package com.example.ecommerce;

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 "Product{" +

"ID=" + productId +

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

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

'}';

}

}

**SearchEngine.java:**

package com.example.ecommerce;

import java.util.Arrays;

import java.util.Comparator;

public class SearchEngine {

// Linear Search by Product Name

public static Product linearSearchByName(Product[] products, String name) {

for (Product product : products) {

if (product.getProductName().equalsIgnoreCase(name)) {

return product;

}

}

return null;

}

// Linear Search by Product ID

public static Product linearSearchById(Product[] products, int id) {

for (Product product : products) {

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

return product;

}

}

return null;

}

// Linear Search by Category (returns multiple products)

public static Product[] linearSearchByCategory(Product[] products, String category) {

return Arrays.stream(products)

.filter(p -> p.getCategory().equalsIgnoreCase(category))

.toArray(Product[]::new);

}

// Binary Search by Product Name (array must be sorted by name)

public static Product binarySearchByName(Product[] products, String name) {

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

String midName = products[mid].getProductName();

int comparison = name.compareToIgnoreCase(midName);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

right = mid - 1;

} else {

left = mid + 1;

}

}

return null;

}

// Sort by product name (for binary search)

public static void sortByName(Product[] products) {

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

}

public static void displayAllProducts(Product[] products) {

for (Product product : products) {

System.out.println(product);

}

}

}

**Product.java:**

package com.example.ecommerce;

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 "Product{" +

"ID=" + productId +

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

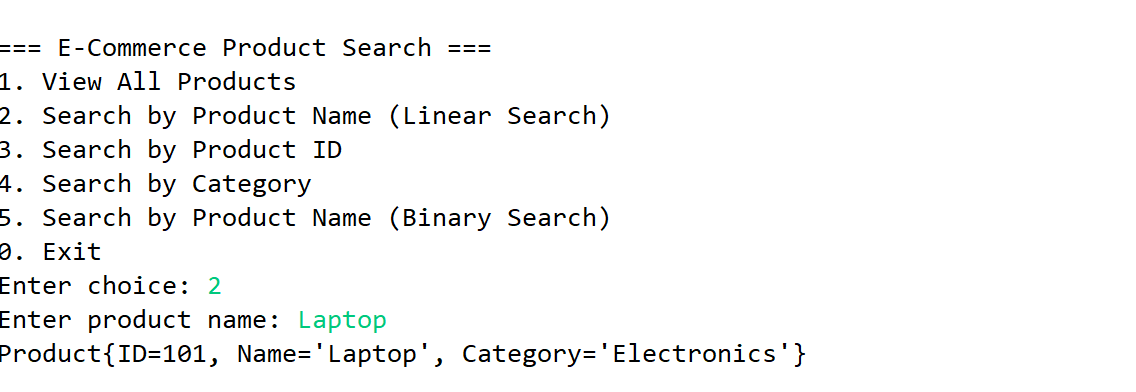
", Category='" + category + '\'' +

'}';

}

}

**OUTPUT:**



**Time Complexity Analysis:**

|  |
| --- |
| **Linear Search** |

|  |  |  |
| --- | --- | --- |
| **O(n)**   |  | | --- | | **Binary Search** |  |  | | --- | | **O(log n)** | |

**EXERCISE 2: FINANCIAL FORCASTING**

**FinancialForecast.java**:

package com.example.finance;

public class FinancialForecast {

// Recursive method to calculate future value

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

if (years == 0) {

return presentValue;

} else {

return (1 + rate) \* forecastFutureValue(presentValue, rate, years - 1);

}

}

public static void main(String[] args) {

double presentValue = 10000; // ₹10,000

double growthRate = 0.08; // 8% annual growth

int years = 5;

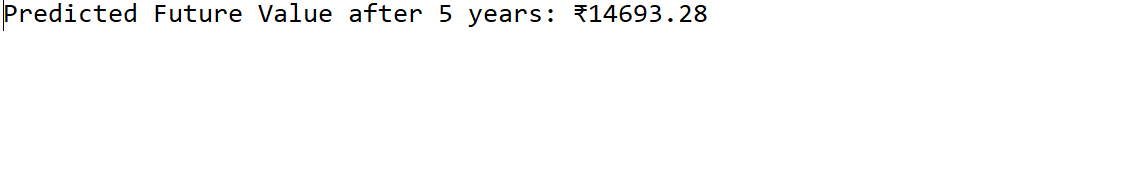
double futureValue = forecastFutureValue(presentValue, growthRate, years);

System.out.printf("Predicted Future Value after %d years: ₹%.2f%n", years, futureValue);

}

}

**OUTPUT:**



**ANALYSIS:**

**Time Complexity**

**T(n) = T(n-1) + O(1) → O(n) (Linear Time)**

**One recursive call per year**