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

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

**Product.java:**

public 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 productId + " - " + productName + " [" + category + "]";

}

}

**SearchAlgorithms.java**

public class SearchAlgorithms {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

Product midProduct = products[mid];

if (midProduct.productName.equalsIgnoreCase(targetName)) {

return midProduct;

} else if (midProduct.productName.compareToIgnoreCase(targetName) < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

}

**Main.java**

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Chair", "Furniture"),

new Product(3, "Pen", "Stationery"),

new Product(4, "Mobile", "Electronics"),

new Product(5, "Desk", "Furniture")

};

Product foundLinear = SearchAlgorithms.linearSearch(products, "Pen");

System.out.println("Linear Search Result: " + (foundLinear != null ? foundLinear : "Not found"));

Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

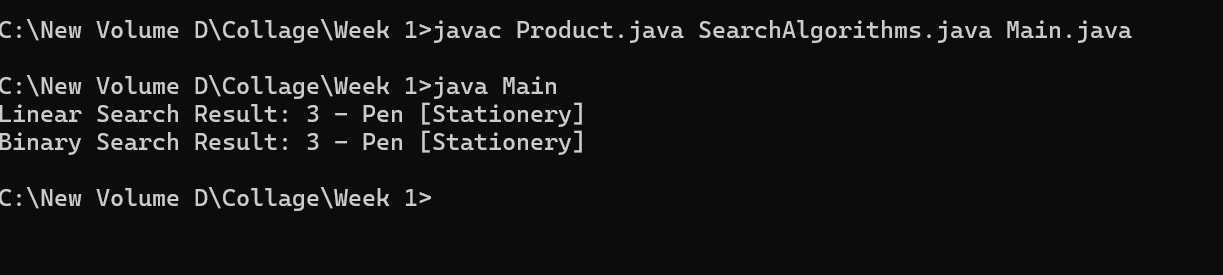
Product foundBinary = SearchAlgorithms.binarySearch(products, "Pen");

System.out.println("Binary Search Result: " + (foundBinary != null ? foundBinary : "Not found"));

}

}

**OUTPUT:**



**Time Complexity Analysis:**

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

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

**EXERCISE 2: FINANCIAL FORCASTING**

**FinancialForecast.java**:

public class FinancialForecast {

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

if (years == 0) {

return presentValue;

}

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

}

public static void main(String[] args) {

double initialInvestment = 10000; // Initial investment amount

double growthRate = 0.08; // Annual growth rate (8%)

int totalYears = 5; // Number of years

double result = calculateFutureValue(initialInvestment, growthRate, totalYears);

System.out.printf("Future Value after %d years: Rs.%.2f\n", totalYears, result);

}

}

**OUTPUT:**

A screen shot of a computer screen

AI-generated content may be incorrect.

**ANALYSIS:**

**Time Complexity**

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

**One recursive call per year**