WEEK 1-Exercise 2: 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;

}

public String toString() {

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

}

}

**SearchAlgorithms.java**

import java.util.\*;

public class SearchAlgorithms {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(targetName);

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

else if (cmp < 0) low = mid + 1;

else high = mid - 1;

}

return null;

}

}

**Main.java**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

new Product(103, "Book", "Stationery"),

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

new Product(105, "Phone", "Electronics")

};

String target = "Watch";

Product foundLinear = SearchAlgorithms.linearSearch(products, target);

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

// Sort for Binary Search

Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

Product foundBinary = SearchAlgorithms.binarySearch(products, target);

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

System.out.println("\nAnalysis:");

System.out.println("Linear Search Time Complexity: O(n)");

System.out.println("Binary Search Time Complexity: O(log n)");

System.out.println("Binary search is faster but needs sorted data.");

}

}

**Analysis**

Big O Analysis

Linear Search: O(n)

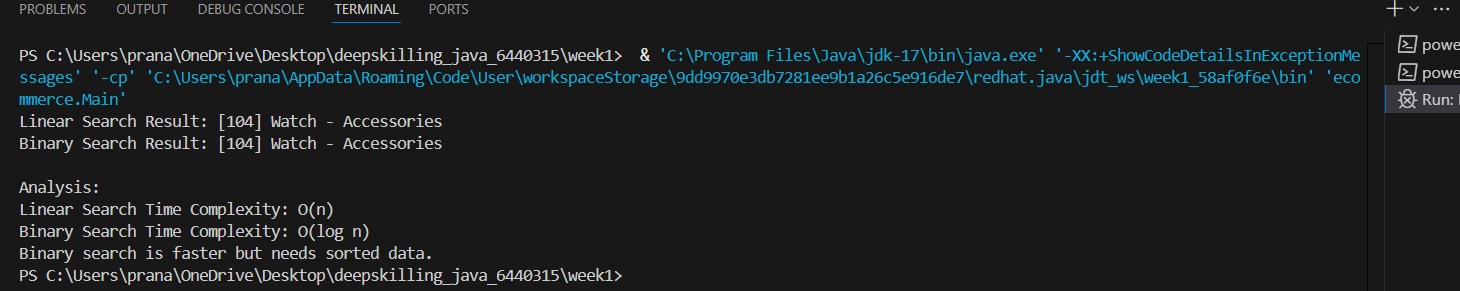
- Best Case: O(1)

- Average/Worst Case: O(n)

Binary Search: O(log n)

- Best Case: O(1)

- Average/Worst Case: O(log n)



**WEEK 1- Exercise 7: Financial Forecasting**

**FinancialForecast.java**

public class FinancialForecast {

public static double forecast(double amount, double rate, int years) {

if (years == 0) {

return amount;

}

return forecast(amount \* (1 + rate), rate, years - 1);

}

public static void main(String[] args) {

double initialAmount = 10000.0;

double growthRate = 0.05; // 5%

int years = 5;

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

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

}

}

**Time Complexity**

The recursive method makes **one recursive call per year**:

* So, **Time Complexity = O(n)** where n = years

**Optimization Tip**

* If the calculation became complex (e.g., with caching or multiple factors), use **memoization** or **dynamic programming** to avoid redundant calculations.
* For this simple exponential growth, recursion is clear and performant.

