**MODULE-2**

**DATA STRUCTURES AND ALGORITHM**

**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.

**IMPLEMENTATION:**

**1.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 + ")";

}

}

**2. SearchMethods.java**

public class SearchMethods {

//linear search(unsorted array)

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

for (Product p : products) {

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

return p;

}

}

return null;

}

// Binary Search(sorted array)

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

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

while (low <= high) {

int mid = (low + high) / 2;

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

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}}

**3.ProductMain.java**

import java.util.\*;

public class ProductMain {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shoes", "Fashion"),

new Product(103, "Phone", "Electronics"),

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

new Product(105, "Backpack", "Fashion")

};

// Linear Search Example

System.***out***.println("Linear Search:");

Product result1 = SearchMethods.*linearSearch*(products, "Phone");

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

// Sort array for binary search by productName

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

// Binary Search Example

System.***out***.println("\nBinary Search:");

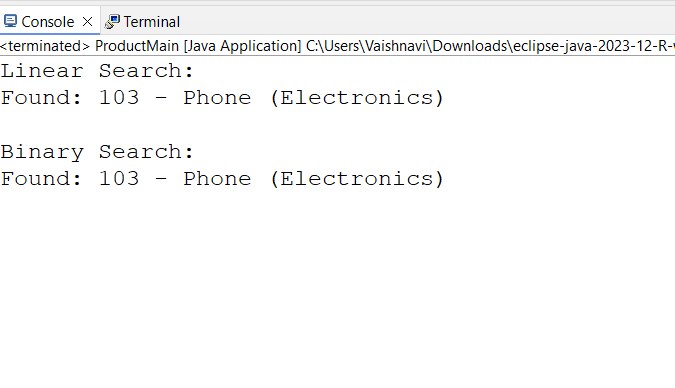
Product result2 = SearchMethods.*binarySearch*(products, "Phone");

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

}

}

**OUTPUT:**

**ANALYSIS:**

1.Compare the time complexity of linear and binary search algorithms.

**Time Complexity:**

| **Algorithm** |  | **Time Complexity** |
| --- | --- | --- |
| Linear Search |  | O(n) |
| Binary Search |  | O(log n) |

2. Discuss which algorithm is more suitable for your platform and why.

Use **Binary Search** after sorting product names (or use efficient data structures like HashMap, TreeMap, or databases).

For small or unsorted lists, **Linear Search** is okay.

**EXERCISE 7:**

**FINANCIAL FORECASTING**

**SCENARIO:**

You are developing a financial forecasting tool that predicts future values based on past data.

**IMPLEMENTATION:**

**1.FinancialForecast.java**

public class FinancialForecast {

// Recursive method to calculate future value

public static double futureValue(double initialAmount, double rate, int years) {

if (years == 0) {

return initialAmount;

} else {

return *futureValue*(initialAmount, rate, years - 1) \* (1 + rate);

}

}

public static void main(String[] args) {

double initialAmount = 10000;

double annualRate = 0.05; // 5% growth

int years = 5;

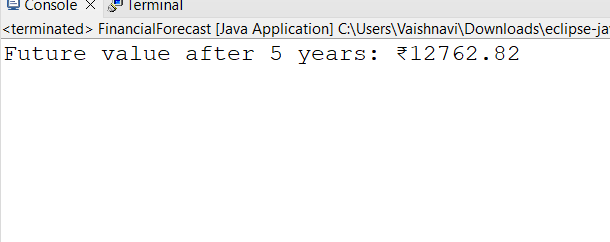
double predictedValue = *futureValue*(initialAmount, annualRate, years);

System.*out*.printf("Future value after %d years: ₹%.2f\n", years, predictedValue);

}

}

**OUTPUT:**

****

**ANALYSIS:**

1.Discuss the time complexity of your recursive algorithm

Recursive Depth = n (years)

So time complexity is: O(n)

2. Explain how to optimize the recursive solution to avoid excessive computation.

->Recursive methods can be inefficient for large values due to stack memory use.

**Optimized Approach:** Use Iteration (for better performance)