# DATA STRUCTURES AND ALGORITHMS

## 1. E-commerce Platform Search Function

### Code:

### **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;

}

}

### **LinearSearch.java**

public class LinearSearch {

public static int search(Product[] products, String productName) {

for (int i = 0; i < products.length; i++) {

if (products[i].productName.equalsIgnoreCase(productName)) {

return i;

}

}

return -1;

}

}

### **BinarySearch.java**

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

public static int search(Product[] products, String productName) {

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

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

if (result == 0)

return mid;

if (result > 0)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

}

### **Main.java**

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

new Product(103, "Watch", "Accessories"),

new Product(104, "Phone", "Electronics"),

new Product(105, "Shirt", "Clothing")

};

String target = "Phone";

int linearIndex = LinearSearch.search(products, target);

System.out.println("Linear Search Index: " + linearIndex);

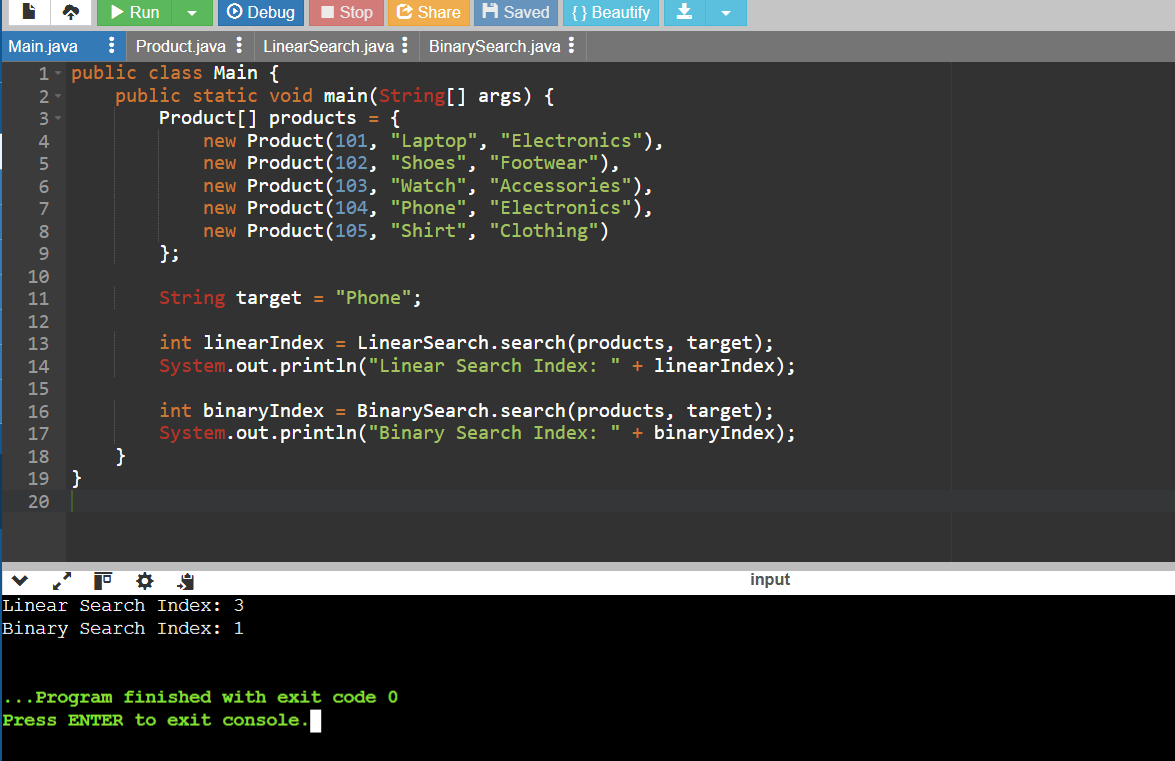
int binaryIndex = BinarySearch.search(products, target);

System.out.println("Binary Search Index: " + binaryIndex);

}

}

### Output:



**1. Big O Notation and Search Performance:**

Big O notation is a way to evaluate the efficiency of an algorithm by describing its performance in relation to the input size. It gives an idea of how an algorithm scales.

* **Best Case:** The most favorable condition (e.g., finding the item immediately).
* **Average Case:** A general expectation of performance across many inputs.
* **Worst Case:** The scenario where the algorithm takes the most time (e.g., item not found).

**2. Time Complexities Comparison:**

* **Linear Search:**
  + Best Case: O(1) – first element matches
  + Average Case: O(n)
  + Worst Case: O(n) – element at the end or not found
* **Binary Search (sorted data required):**
  + Best Case: O(1)
  + Average Case: O(log n)
  + Worst Case: O(log n)

**3. Choosing the Right Search:**

Binary search is faster than linear search for large and sorted datasets. It's ideal when your product list doesn't change often. On the other hand, linear search is better for small or unsorted collections where sorting would be inefficient.

## 2. Financial Forecasting

### Code:

### **FinancialForecast.java**

public class FinancialForecast {

public static double predictFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return predictFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

}

### **Main.java**

public class Main {

public static void main(String[] args) {

double currentValue = 10000;

double growthRate = 0.05;

int years = 5;

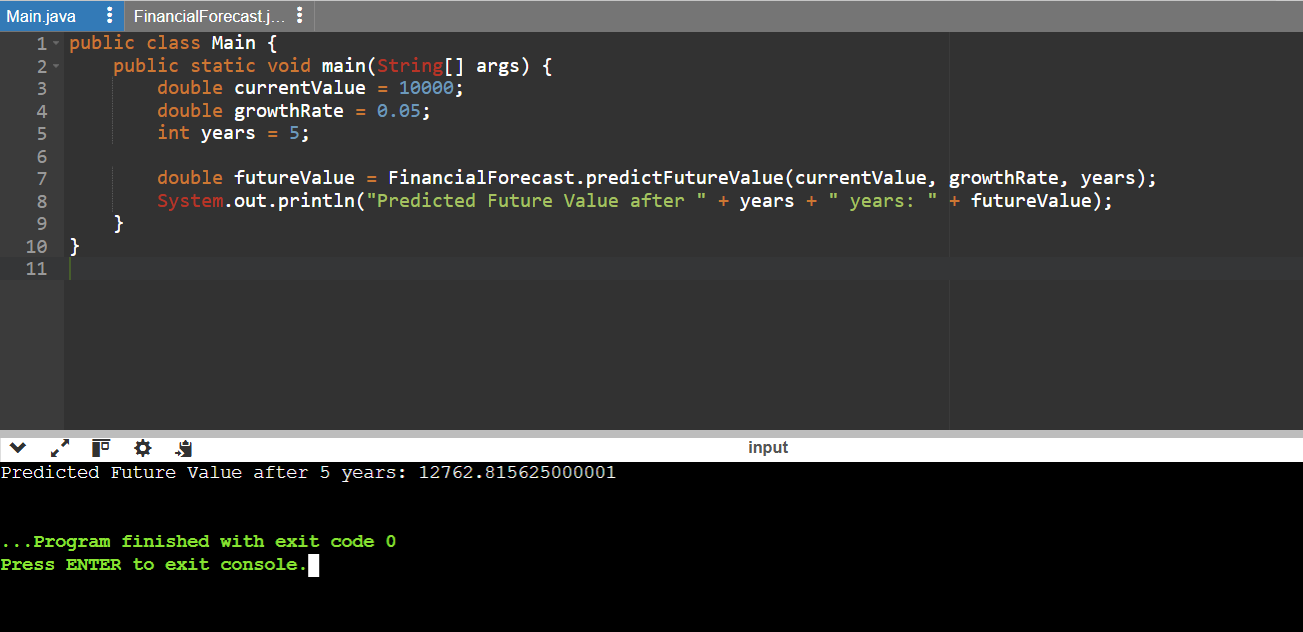
double futureValue = FinancialForecast.predictFutureValue(currentValue, growthRate, years);

System.out.println("Predicted Future Value after " + years + " years: " + futureValue);

}

}

### Output:



**What is Recursion?**

Recursion happens when a function repeatedly calls itself with a modified argument. It’s especially useful for solving problems that have a repetitive or divide-and-conquer nature.

**Efficiency of the Recursive Approach:**

The algorithm’s time complexity is **O(n)** since the function executes once for every year. The deeper the recursion, the more function calls are made.

**Reducing Computation:**

To make the solution more efficient:

* You can cache results using memoization to prevent redoing calculations.
* A better option for this case is a loop-based method, which uses less memory and is quicker for large inputs.

Example of loop method:

double value = initial;

for (int i = 0; i < years; i++) {

value \*= (1 + rate);

}

return value;