**Ques 1 : E – Commerce Platform Seach Function**

**Big O Notation :**

- It describes how the runtime of an algorithm grows with input size (n).

**Linear Search:**

- Best Case: O(1) → Match found at first index

- Average Case: O(n/2) → Match in the middle

- Worst Case: O(n) → Match at last or not found

**Binary Search:**

- Requires sorted array

- Best Case: O(1)

- Average & Worst Case: O(log n)

Binary search is much faster for large, sorted data sets.

**Code :**

import java.util.Arrays;  
import java.util.Comparator;  
  
public class EcommerceSearchFunction {  
  
 // Product class  
 static 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 + "]";  
 }  
 }

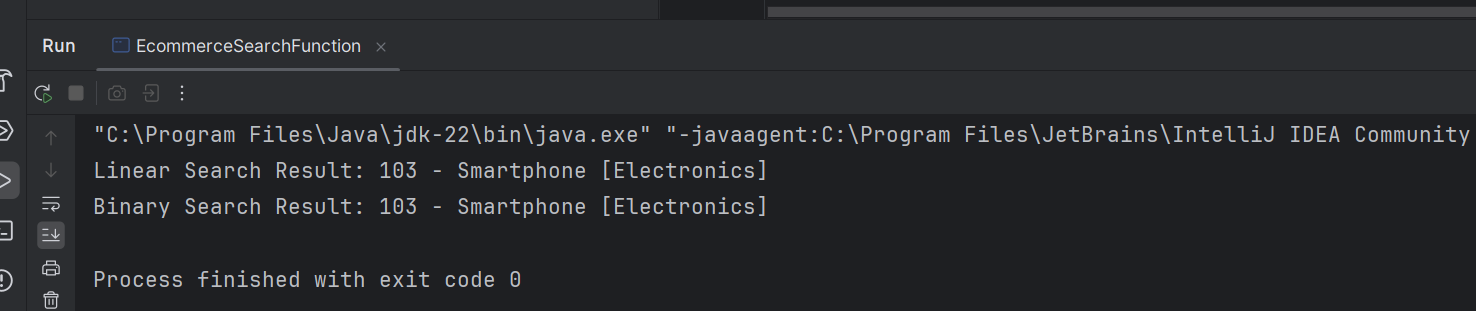
// Linear Search  
 public static Product linearSearch(Product[] products, String keyword) {  
 for (Product p : products) {  
 if (p.productName.equalsIgnoreCase(keyword)) {  
 return p;  
 }  
 }  
 return null;  
 }

// Binary Search (Assumes sorted by productName)

public static Product binarySearch(Product[] products, String keyword) {  
 int left = 0;  
 int right = products.length - 1;  
  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 int cmp = products[mid].productName.compareToIgnoreCase(keyword);  
  
 if (cmp == 0) {  
 return products[mid];  
 } else if (cmp < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
  
 return null;  
 }

// Main Method  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Laptop", "Electronics"),  
 new Product(102, "Shoes", "Footwear"),  
 new Product(103, "Smartphone", "Electronics"),  
 new Product(104, "Bag", "Accessories"),  
 new Product(105, "Watch", "Accessories")  
 };  
  
 String search = "Smartphone";  
  
 // Linear Search Test  
 Product result1 = *linearSearch*(products, search);  
 System.*out*.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));  
  
 // Sort products by name for binary search  
 Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName.toLowerCase()));  
  
 // Binary Search Test  
 Product result2 = *binarySearch*(products, search);  
 System.*out*.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));  
 }  
}

**Output :**



**Time Complexity Comparison**

* Linear Search: O(n) → Checks each element one by one.
* Binary Search: O(log n) → Cuts the search space in half each time.
* When to use Binary Search? When your product list is already sorted or can be sorted efficiently.

**Ques 2 : Financial Forecasting**

**Recursion :**

A technique where a method calls itself to solve a smaller version of the problem.

Example: To forecast the future value:

futureValue(n) = futureValue(n-1) \* (1 + growthRate)

base case: year 0 value = initial investment

**Code :**

public class FinancialForecast {

// Recursive method to calculate future value

public static double forecastValue(int years, double initialAmount, double growthRate) {

if (years == 0) {

return initialAmount; // base case

}

return forecastValue(years - 1, initialAmount, growthRate) \* (1 + growthRate);

}

public static void main(String[] args) {

double initialInvestment = 10000; // Rs 10,000

double annualGrowthRate = 0.08; // 8% growth

int years = 5;

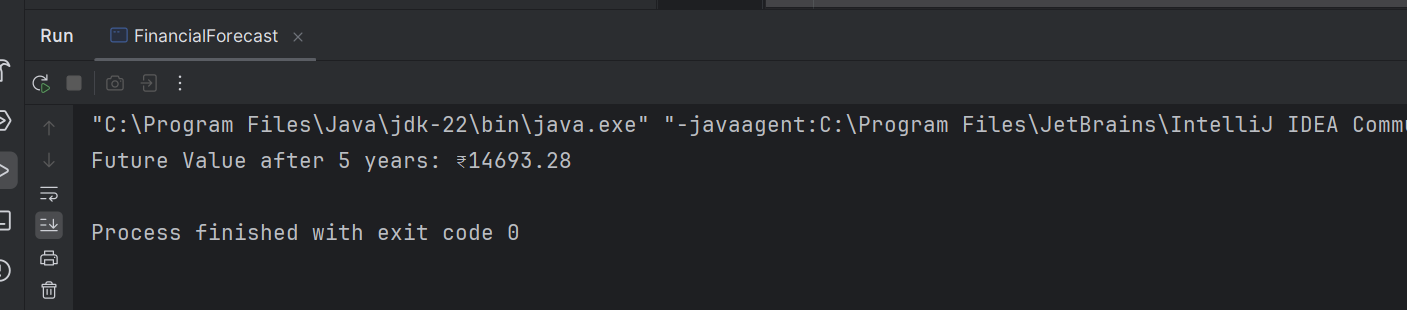
double futureValue = forecastValue(years, initialInvestment, annualGrowthRate);

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

}

}

**Output :**

****

**Time Complexity:**

- The recursive method calls itself once per year.

- So time complexity is O(n), where n = number of years.

**Optimization:**

- This problem doesn't need memoization, as there’s no repeated computation.

- For large 'n', use iteration to avoid stack overflow.

**Iterative alternative (for optimization):**

double value = initialAmount;

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

value \*= (1 + growthRate);

}