Data structures and Algorithms

Exercise 2: E-commerce Platform Search Function

Theory:

Big O Notation:

Big O notation describes the upper bound of an algorithm’s running time as the input size grows. It helps us understand how efficiently an algorithm scales.

Use Linear Search:

- For small datasets
- When data is unsorted

Use Binary Search:

- When data is sorted
- For frequent search operations

Binary search is more suitable for performance-sensitive e-commerce platforms where searching is done frequently and speed is important. Maintaining a sorted product list (or using a database index) is critical.

Code:

Product.java

```java
package ecommerce;
public class Product {
  int ProductId;
  String ProductName,Category;
    Product(int ProductId,String ProductName,String Category ){
        this.ProductId=ProductId;
        this.ProductName=ProductName;
        this.Category=Category;
    }

    public String toString(){
      return ProductId+"-"+ProductName+"("+Category+")";
    }
}
```

LinearSearch.java

```java
package ecommerce;

public class LinearSearch {
    public static Product linearSearch(Product[] products, String name) {
        for (Product p : products) {
            if (p.ProductName.equalsIgnoreCase(name)) {
                return p;
            }
        }
        return null;
    }
}
```

BinarySearch.java

```java
package ecommerce;

import java.util.Arrays;
import java.util.Comparator;
public class BinarySearch {
 public static Product binarySearch(Product[] products, String name) {
        int low = 0, high = products.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            int cmp = products[mid].ProductName.compareToIgnoreCase(name);
            if (cmp == 0) return products[mid];
            else if (cmp < 0) low = mid + 1;
            else high = mid - 1;
        }
        return null;
    }

    // Optional sorting helper
    public static void sortProductsByName(Product[] products) {
        Arrays.sort(products, Comparator.comparing(p ->
p.ProductName.toLowerCase()));
    }
}
```

SearchTest.java

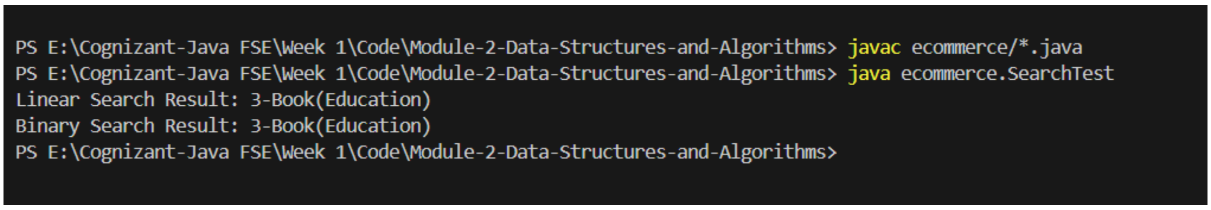
```java
package ecommerce;

public class SearchTest {
    public static void main(String[] args) {
        Product[] products = {
            new Product(1, "Laptop", "Electronics"),
            new Product(2, "Shampoo", "Personal Care"),
            new Product(3, "Book", "Education"),
            new Product(4, "Phone", "Electronics"),
            new Product(5, "Shoes", "Footwear")
        };

        Product found1 = LinearSearch.linearSearch(products, "Book");
        System.out.println("Linear Search Result: " + (found1 != null ? found1 : "Not Found"));

        BinarySearch.sortProductsByName(products);
        Product found2 = BinarySearch.binarySearch(products, "Book");
        System.out.println("Binary Search Result: " + (found2 != null ? found2 : "Not Found"));
    }
}
```

Output:



```
PS E:\Cognizant-Java FSE\Week 1\Code\Module-2-Data-Structures-and-Algorithms> javac ecommerce/*.java
PS E:\Cognizant-Java FSE\Week 1\Code\Module-2-Data-Structures-and-Algorithms> java ecommerce.SearchTest
Linear Search Result: 3-Book(Education)
Binary Search Result: 3-Book(Education)
PS E:\Cognizant-Java FSE\Week 1\Code\Module-2-Data-Structures-and-Algorithms>
```

Exercise 7: Financial Forecasting

Theory:

Recursion is a programming technique where a method calls itself to solve a smaller instance of the same problem.

- It simplifies problems that have a repetitive or nested structure.
- Ideal for computations like factorial, Fibonacci, or in our case, compound growth over time.

Code:

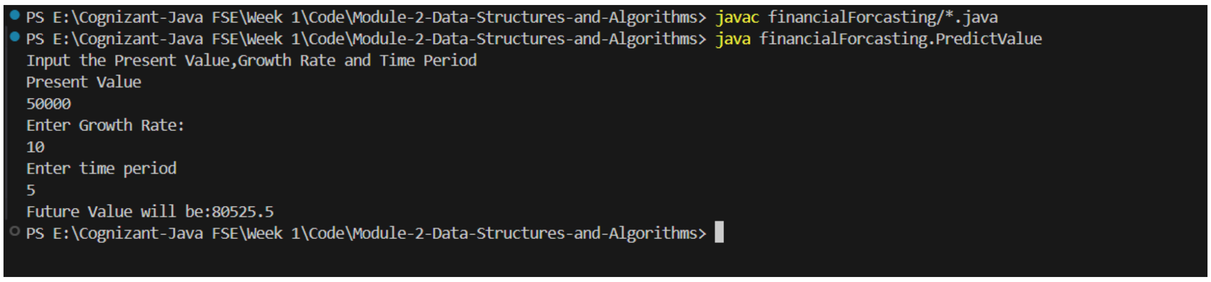
```java
package financialForcasting;

import java.util.*;

public class PredictValue {

    static float predict(int PresentValue,float growthRate,int time){
        if(time<=0)
        return PresentValue;
        return (1+growthRate/100)*predict(PresentValue, growthRate, time-1);
    }
    public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        System.out.println("Input the Present Value,Growth Rate and Time
Period");
        System.out.println("Present Value");
        int PresentValue=in.nextInt();
        System.out.println("Enter Growth Rate:");
        float growthRate=in.nextFloat();
        System.out.println("Enter time period");
        int time=in.nextInt();
        float futureRate=predict(PresentValue,growthRate,time);
        System.out.println("Future Value will be:"+futureRate);
        in.close();
    }
}
```

Output:



```
PS E:\Cognizant-Java FSE\Week 1\Code\Module-2-Data-Structures-and-Algorithms> javac financialForcasting/*.java
PS E:\Cognizant-Java FSE\Week 1\Code\Module-2-Data-Structures-and-Algorithms> java financialForcasting.PredictValue
Input the Present Value,Growth Rate and Time Period
Present Value
50000
Enter Growth Rate:
10
Enter time period
5
Future Value will be:80525.5
PS E:\Cognizant-Java FSE\Week 1\Code\Module-2-Data-Structures-and-Algorithms>
```