**ECommerceSearch**

**Setup:**

* + Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.

**Implementation:**

* + Implement linear search and binary search algorithms.

Store products in an array for linear search and a sorted array for binary search.  
  
Code:  
  
package Week\_1\_Data\_Structures\_and\_Algorithms;

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearch {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shirt", "Clothing"),

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

new Product(104, "Shoes", "Footwear"),

new Product(105, "Book", "Stationery")

};

Product target = products[0];

int linearIndex = Search.*linearSearch*(products, target.getProductId());

System.*out*.println("liner\_Index : " + linearIndex);

System.*out*.println("found : "+ products[linearIndex].toString());

Arrays.*sort*(products, Comparator.*comparingInt*(Product::getProductId));

int binaryIndex = Search.*binarySearch*(products, target.getProductId());

System.*out*.println("Binary\_Index: " + binaryIndex);

System.*out*.println("found : "+ products[binaryIndex].toString());

Product tar = products[4];

int Index = Search.*linearSearch*(products, tar.getProductId());

System.*out*.println("liner\_Index : " + Index);

System.*out*.println("found : "+ products[Index].toString());

Arrays.*sort*(products, Comparator.*comparingInt*(Product::getProductId));

int bin\_Index = Search.*binarySearch*(products, tar.getProductId());

System.*out*.println("Binary\_Index: " + bin\_Index);

System.*out*.println("found : "+ products[bin\_Index].toString());

}

}

class Product {

private final int productId;

private final String productName;

private final String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() {

return productId;

}

public String toString() {

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

}

}

class Search {

public static int linearSearch(Product[] products, int productId) {

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

if (products[i].getProductId() == productId) return i;

}

return -1;

}

public static int binarySearch(Product[] products, int productId) {

int left = 0, right = products.length - 1;

while (left <= right) {

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

if (products[mid].getProductId() == productId) return mid;

if (products[mid].getProductId() < productId) left = mid + 1;

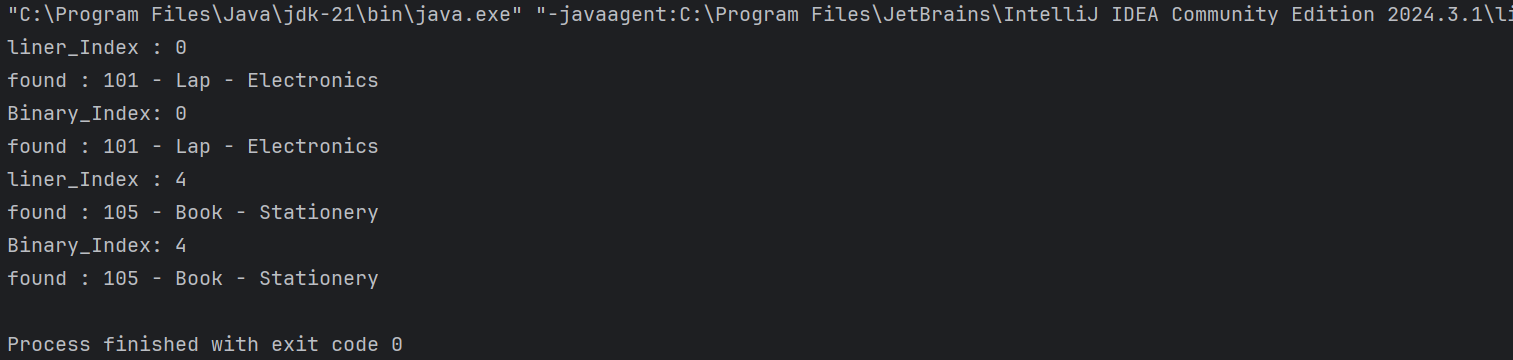
else right = mid - 1;

}

return -1;

}

}

Output:  


**FinancialForecasting**

**Setup:**

Create a method to calculate the future value using a recursive approach.

**Implementation:**

Implement a recursive algorithm to predict future values based on past growth rates  
  
  
  
   
 Code:  
  
 **package Week\_1\_Data\_Structures\_and\_Algorithms;**

public class FinancialForecasting {

public static void main(String[] args) {

double initialValue = 3000;

double growthRate = 0.008;

int years = 4;

double futureValue = Forecast.*recursiveForecast*(initialValue, growthRate, years);

System.*out*.println(" value in future " + futureValue);

}

}

class Forecast {

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

if (years == 0) return amount;

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

}

}

Output:  
  
 