**Exercise 2: 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.category = category;  
 this.productName = productName;  
 this.productId = productId;  
 }  
 @Override  
 public String toString() {  
 return "Product{" +  
 "productId=" + productId +  
 ", productName='" + productName + '\'' +  
 ", category='" + category + '\'' +  
 '}';  
 }  
}

Main.java

import java.util.\*;  
public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(1, "Mobile", "Electronics"),  
 new Product(2, "Kurta", "Fashion"),  
 new Product(3, "Keyboard", "Electronics"),  
 new Product(4, "Notebook", "Stationery")  
 };  
 Product foundLinear = *linearSearch*(products, "Mobile");  
 System.*out*.println("Linear Search Result: " + foundLinear);  
 Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName));  
 Product foundBinary = *binarySearch*(products, "Mobile");  
 System.*out*.println("Binary Search Result: " + foundBinary);  
 }  
 public static Product linearSearch(Product[] products, String targetName) {  
 for (Product product : products) {  
 if (product.productName.equals(targetName)) {  
 return product;  
 }  
 }  
 return null;  
 }  
 public static Product binarySearch(Product[] products, String targetName) {  
 int low = 0;  
 int high = products.length - 1;  
  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int comparison = products[mid].productName.compareTo(targetName);  
  
 if (comparison == 0) {  
 return products[mid];  
 } else if (comparison < 0) {  
 low = mid + 1;  
 } else {  
 high = mid - 1;  
 }  
 }  
 return null;  
 }

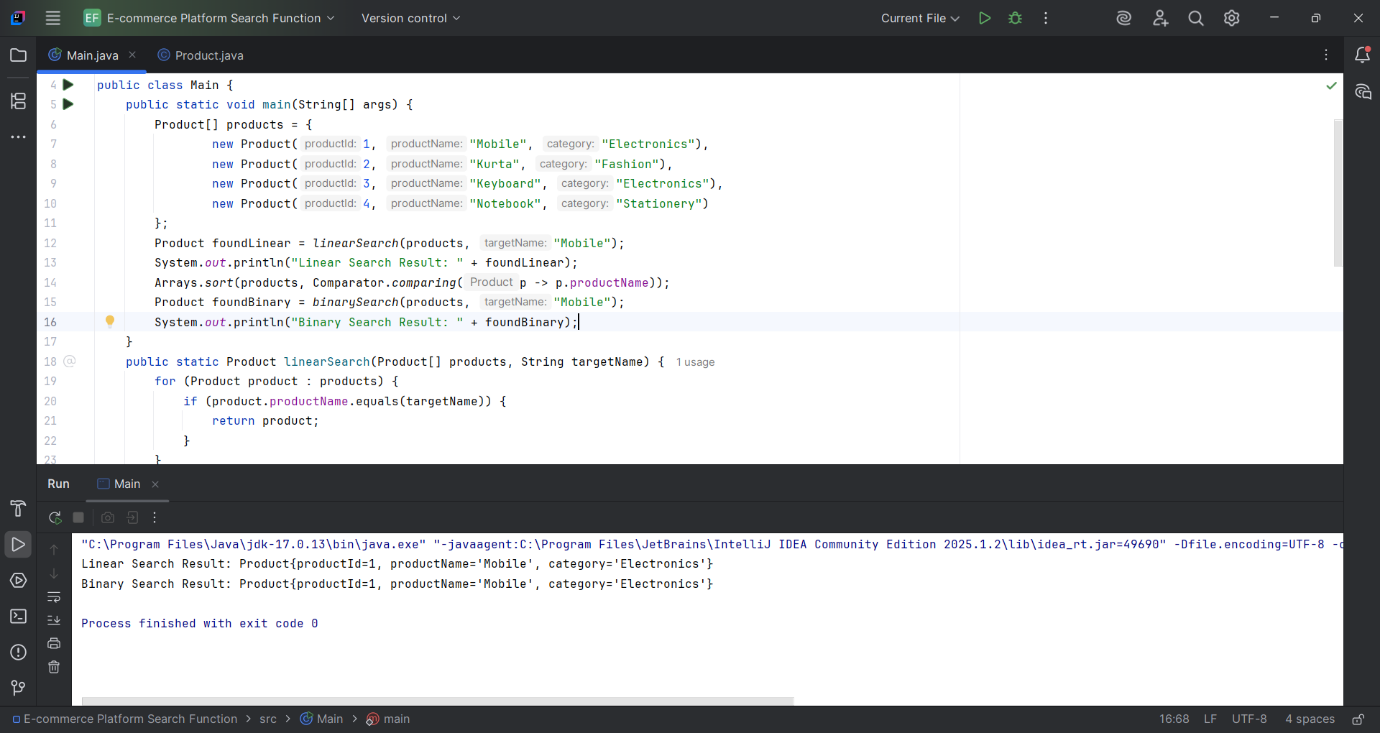
**Time Complexity**

|  |  |  |
| --- | --- | --- |
| **Name** | **Big O Notation** | **Example** |
| Constant | O(1) | Single operation |
| Logarithmic | O(log n) | Binary search |
| Linear | O(n) | Linear search |
| Quadratic | O(n²) | Nested loops |
| Exponential | O(2ⁿ) | Recursive branching |

**Analysis**

|  |  |  |
| --- | --- | --- |
| Algorithm | Time Complexity | Needs Sorted Input |
| Linear Search | O(n) | No |
| Binary Search | O(log n) | Yes |

**Output**



**Exercise 7: Financial Forecasting**

**Code**

*Main.java*public class Main {  
 public static double futureValue(double currentValue, double growthRate, int years) {  
 if (years == 0) {  
 return currentValue;  
 }  
 return *futureValue*(currentValue, growthRate, years - 1) \* (1 + growthRate);  
 }  
 public static void main(String[] args) {  
 double currentValue = 10000;  
 double growthRate = 0.10;  
 int years = 5;  
  
 double futureVal = *futureValue*(currentValue, growthRate, years);  
 System.*out*.printf("Value after %d years: %.2f\n", years, futureVal);  
 }  
}

**Time Complexity**

* Each call reduces years by 1.
* So, for n years it becomes O(n) time.

**Optimization**

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

double result = currentValue;

for (int i = 1; i <= years; i++) {

result \*= (1 + growthRate);

}

return result;

}

**Output**

