**Exercise 2: E-commerce Platform Search Function**

Big O Notation & Search Cases :

Big O Notation describes the performance or complexity of an algorithm in terms of input size.  
It helps us understand:

* Best Case: Minimum operations.
* Average Case: Expected performance for random input.
* Worst Case: Maximum operations required

Product Class :

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;

}

@Override

public String toString() {

return "Product [ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";

}

}

Search Implementations :

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

public static Product linearSearchByName(Product[] products, String name) {

for (Product product : products) {

if (product.productName.equalsIgnoreCase(name)) {

return product;

}

}

return null;

}

public static Product binarySearchById(Product[] products, int targetId) {

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

while (left <= right) {

int mid = (left + right) / 2;

if (products[mid].productId == targetId) {

return products[mid];

} else if (products[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void sortProductsById(Product[] products) {

for (int i = 0; i < products.length - 1; i++) {

for (int j = i + 1; j < products.length; j++) {

if (products[i].productId > products[j].productId) {

Product temp = products[i];

products[i] = products[j];

products[j] = temp;

}

}

}

}

}

Main class :

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(102, "Laptop", "Electronics"),

new Product(101, "Shampoo", "Personal Care"),

new Product(104, "Desk", "Furniture"),

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

};

String searchName = "Phone";

Product foundByName = ProductSearch.linearSearchByName(products, searchName);

System.out.println("Linear Search Result (Name): " + (foundByName != null ? foundByName : "Not Found"));

ProductSearch.sortProductsById(products);

int searchId = 103;

Product foundById = ProductSearch.binarySearchById(products, searchId);

System.out.println("Binary Search Result (ID): " + (foundById != null ? foundById : "Not Found"));

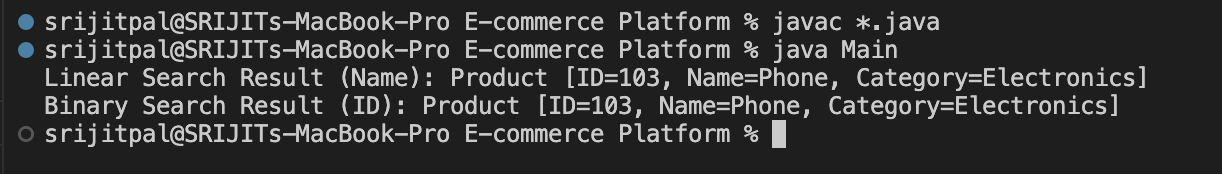
}

}

| Algorithm | Time Complexity | Suitable for |
| --- | --- | --- |
| Linear Search | O(n) | Unsorted data, small datasets |
| Binary Search | O(log n) | Sorted large datasets |

Analysis :

Binary Search is faster but needs pre-sorted data. In large-scale e-commerce platforms where performance is critical, binary search (along with optimized indexing or data structures like B-trees or hash maps) is preferred.



**Exercise 7: Financial Forecasting**

Recursive Algorithms :  
  
Recursion is a programming technique where a function calls itself to solve smaller subproblems.

Recursive Future Value Method :

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 class :

public class Main {

public static void main(String[] args) {

double currentValue = 1000;

double growthRate = 0.10; // 10%

int years = 5;

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

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

}

}

Analysis :

* Time Complexity : O(n) where n is the number of years.
* Problem : Recursive calls can cause stack overflow for large n.

Optimization :

It can be optimised using Memoization.

1. Create a storage (like HashMap<Integer, Double> in Java).
2. Before computing a result, check if it's already in the map.
3. If yes, return the stored result.
4. If not, compute it, store it in the map, and return it.

