**Data Structures & Algorithms**

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

**Code:**

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

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;

}

public static void sortProducts(Product[] products) {

Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

}

// Binary search

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 compare = products[mid].productName.compareToIgnoreCase(targetName);

if (compare == 0) {

return products[mid];

} else if (compare < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

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

new Product(103, "Shoes", "Fashion"),

new Product(104, "Book", "Education"),

new Product(105, "Watch", "Accessories")

};

sortProducts(products);

Scanner scanner = new Scanner(System.in);

System.out.print("Enter product name to search: ");

String searchName = scanner.nextLine();

Product result = binarySearch(products, searchName);

if (result != null) {

System.out.println("Product found: " + result);

} else {

System.out.println("Product not found.");

}

scanner.close();

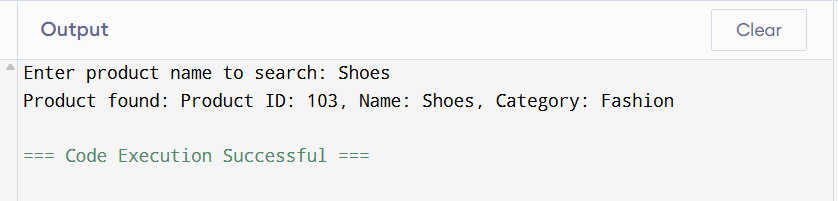
}

}

**Analysis:**

* Binary Search is preferred if products are sorted and need high speed.(O(logn))
* Linear Search is better if the product list is small or unsorted.(O(n)).

**Output:**



**Exercise 3: Sorting Customer Orders**

**Code:**

import java.util.\*;

class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

@Override

public String toString() {

return "OrderID: " + orderId + ", Customer: " + customerName + ", Total: $" + totalPrice;

}

public static void quickSort(Order[] orders, int low, int high) {

if (low < high) {

int pi = partition(orders, low, high);

quickSort(orders, low, pi - 1);

quickSort(orders, pi + 1, high);

}

}

private static int partition(Order[] orders, int low, int high) {

double pivot = orders[high].totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].totalPrice < pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Order[] orders = {

new Order(1001, "Ajay", 450.00),

new Order(1002, "Arun", 125.50),

new Order(1003, "Kavya", 785.25),

new Order(1004, "Riya", 310.40),

new Order(1005, "Neeta", 200.00)

};

System.out.println("Before Sorting:");

for (Order o : orders) {

System.out.println(o);

}

quickSort(orders, 0, orders.length - 1);

System.out.println("\nAfter Sorting :");

for (Order o : orders) {

System.out.println(o);

}

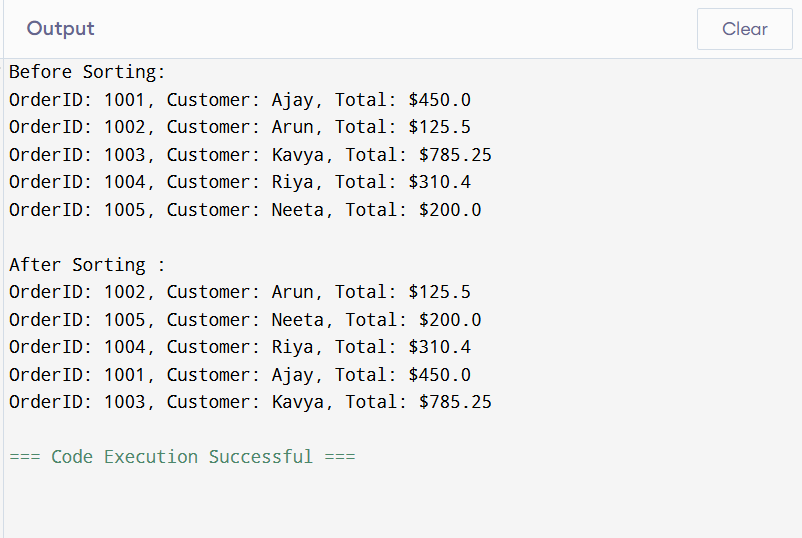
}

}

**Analysis:**

* Quick sort is better as it performs fewer swaps and comparisons.(O(nlogn))
* Bubble sort is only suitable for smaller inputs.

**Output:**

****

**Exercise 6: Library Management System**

**Code:**

import java.util.Scanner;

class Book {

int bookId;

String title;

String author;

public Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

@Override

public String toString() {

return "BookID: " + bookId + ", Title: \"" + title + "\", Author: " + author;

}

public static Book linearSearch(Book[] books, String searchTitle) {

for (Book book : books) {

if (book.title.equalsIgnoreCase(searchTitle)) {

return book;

}

}

return null;

}

public static void main(String[] args) {

Book[] books = {

new Book(101, "The Hobbit", "J.R.R. Tolkien"),

new Book(102, "1984", "George Orwell"),

new Book(103, "To Kill a Mockingbird", "Harper Lee"),

new Book(104, "Pride and Prejudice", "Jane Austen"),

new Book(105, "Moby-Dick", "Herman Melville")

};

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the title of the book to search: ");

String inputTitle = scanner.nextLine();

// linear search

Book foundBook = linearSearch(books, inputTitle);

if (foundBook != null) {

System.out.println("Book Found: " + foundBook);

} else {

System.out.println("Book not found: \"" + inputTitle + "\"");

}

scanner.close();

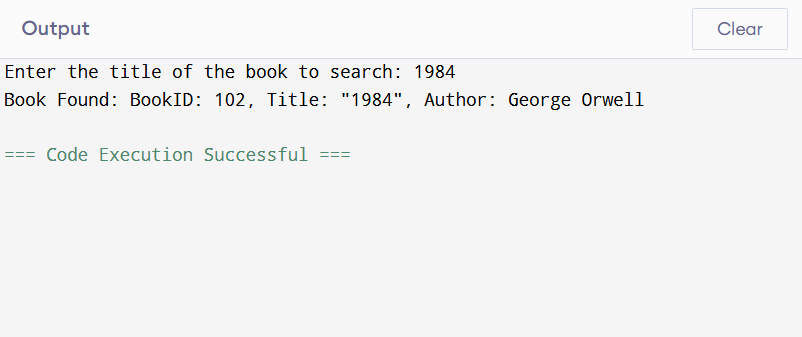
}

}

**Analysis:**

* Use Linear Search when book data is unsorted or very small.
* Use Binary Search when book data is sorted and performance matters.

**Output:**

****

**Exercise 7: Financial Forecasting**

**Code:**

public class Main2 {

int calculate(int amount, int rate, int years) {

if (years == 0) {

return amount;

}

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

}

void run() {

int Amount = 5000;

int Rate = 2;

int Years = 5;

int result = calculate(Amount, Rate, Years);

System.out.println("No of years:" + Years);

System.out.println("Results: Rs."+result);

}

public static void main(String[] args) {

new Main2().run();

}

}

**Output:**