**Exercise 3: Sorting Customer Orders**

**1. Understand Sorting Algorithms**

**Different sorting algorithms:**

* **Bubble Sort:** Simple but inefficient, O(n^2) time complexity.
* **Insertion Sort:** Efficient for small or nearly sorted datasets, O(n^2) time complexity.
* **Quick Sort:** Efficient, average O(n log n) time complexity, but worst-case O(n^2).
* **Merge Sort:** Stable, always O(n log n) time complexity.

**2. Setup**

**Create an Order class:**

public class Order {

private int orderId;

private String customerName;

private double totalPrice;

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

this.orderId = orderId; this.totalPrice = totalPrice;

this.customerName = customerName;

}

public int getOrderId() {

return orderId;

}

public void setOrderId(int orderId) {

this.orderId = orderId;

}

public String getCustomerName() {

return customerName;

}

public void setCustomerName(String customerName) {

this.customerName = customerName;

}

public double getTotalPrice() {

return totalPrice;

}

public void setTotalPrice(double totalPrice) {

this.totalPrice = totalPrice;

}

}

**3. Implementation**

**Implement Bubble Sort and Quick Sort:**

import java.util.Arrays;

public class SortingAlgorithms {

**// Bubble Sort**

public static void bubbleSort(Order[] orders) {

int n = orders.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {

// Swap orders[j] and orders[j+1]

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

**// Quick Sort**

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].getTotalPrice();

int i = (low - 1);

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

if (orders[j].getTotalPrice() <= pivot) {

i++;

**// Swap**

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

**// Swap**

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(1, "Alice", 250.50),

new Order(2, "Bob", 150.75),

new Order(3, "Charlie", 300.40),

new Order(4, "Daisy", 100.20)

};

**// Bubble Sort**

bubbleSort(orders);

System.out.println("Bubble Sort:");

for (Order order : orders) {

System.out.println(order.getCustomerName() + ": " + order.getTotalPrice());

}

**// Quick Sort**

orders = new Order[]{

new Order(1, "Alice", 250.50),

new Order(2, "Bob", 150.75),

new Order(3, "Charlie", 300.40),

new Order(4, "Daisy", 100.20)

};

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

System.out.println("Quick Sort:");

for (Order order : orders) {

System.out.println(order.getCustomerName() + ": " + order.getTotalPrice());

}

}

}

**4. Analysis**

**Performance comparison:**

* **Bubble Sort:** O(n^2) - inefficient for large datasets.
* **Quick Sort:** Average O(n log n), worst-case O(n^2) - generally more efficient for large datasets.

**Preference:**

Quick Sort is generally preferred over Bubble Sort due to its average O(n log n) time complexity, making it more suitable for large datasets.