**DATA STRUCTURES AND ALGORITHMS:**

**Exercise 3: Sorting Customer Orders:**

**1. Understand Sorting Algorithms**

There are many sorting algorithms used in programming. Here are a few with simple explanations:

* **Bubble Sort:**  
  Repeatedly compares adjacent elements and swaps them if they’re in the wrong order. It’s simple but slow for large data.
* **Insertion Sort:**  
  Takes one element at a time and inserts it into the correct position in the sorted part. Good for small arrays.
* **Quick Sort:**  
  Divides the array using a pivot and recursively sorts the left and right sides. It is much faster in most cases.
* **Merge Sort:**  
  Divides the array into halves, sorts them, and merges them back. Very efficient and works well even on large datasets.

**2. Setup: Create Order Class**

Each order has:

* orderId
* customerName
* totalPrice

**3. Implementation (Bubble Sort & Quick Sort)**

**OrderSorting.java**

class Order {

int orderId;

String customerName;

double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

void printOrder() {

System.out.println("Order ID: " + orderId);

System.out.println("Customer Name: " + customerName);

System.out.println("Total Price: " + totalPrice);

System.out.println();

}

}

public class OrderSorting {

// Bubble Sort: Sort orders by totalPrice (ascending)

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].totalPrice > orders[j + 1].totalPrice) {

// Swap

Order temp = orders[j];

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

orders[j + 1] = temp;

}

}

}

}

// Quick Sort:

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);

}

}

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++;

// Swap

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) {

// Sample orders

Order[] orders = {

new Order(1, "Anushka", 4500),

new Order(2, "Rahul", 2200),

new Order(3, "Sneha", 7000),

new Order(4, "Amit", 1500)

};

System.out.println("Original Orders:");

for (Order o : orders) {

o.printOrder();

}

// Bubble Sort

System.out.println("Orders after Bubble Sort (by total price):");

bubbleSort(orders);

for (Order o : orders) {

o.printOrder();

}

// Reset the array for fresh Quick Sort

Order[] orders2 = {

new Order(1, "Anushka", 4500),

new Order(2, "Rahul", 2200),

new Order(3, "Sneha", 7000),

new Order(4, "Amit", 1500)

};

// Quick Sort

System.out.println("Orders after Quick Sort (by total price):");

quickSort(orders2, 0, orders2.length - 1);

for (Order o : orders2) {

o.printOrder();

}

}}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

Quick sort

A screenshot of a computer

AI-generated content may be incorrect.

**Comparison of Time Complexity**

| **Sorting Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Bubble Sort** | O(n) | O(n²) | O(n²) |
| **Quick Sort** | O(n log n) | O(n log n) | O(n²) |

*Worst case in Quick Sort happens when the pivot always divides poorly (like smallest or largest element every time). But in most real cases, it's rare.*

**Why Quick Sort is Preferred Over Bubble Sort:**

1. **Performance**
2. **Efficiency**
3. **Scalability**
4. **Real-World Usage**

Bubble Sort is easy to understand and good for small practice examples. But for real-world systems like an e-commerce platform handling customer orders, **Quick Sort is the better choice** because of its speed and efficiency.