**Exercise 3: Sorting Customer Orders**

**Understanding the Problem:**  
Sorting helps in prioritizing high-value orders. For example, shipping expensive orders first. Comparing sorting algorithms helps choose the best based on time complexity.

**Setup and Implementation:**  
An Order class is used with:

* orderId (int)
* customerName (String)
* totalPrice (double)

Two sorting algorithms are implemented:

* Bubble Sort (simple, but slow)
* Quick Sort (efficient on average)

**Java Code:**

import java.util.\*;

class Order {

int orderId;

String customerName;

double totalPrice;

Order(int id, String name, double price) {

this.orderId = id;

this.customerName = name;

this.totalPrice = price;

}

}

public class SortCustomerOrders {

static void bubbleSort(Order[] arr) {

int n = arr.length;

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

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

if (arr[j].totalPrice > arr[j + 1].totalPrice) {

Order temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

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

double pivot = arr[high].totalPrice;

int i = low - 1;

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

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

i++;

Order temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

Order temp = arr[i + 1];

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

arr[high] = temp;

return i + 1;

}

static void display(Order[] arr) {

for (Order o : arr) {

System.out.println(o.orderId + " " + o.customerName + " " + o.totalPrice);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of orders: ");

int n = sc.nextInt();

Order[] orders1 = new Order[n];

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

System.out.println("Enter details for order " + (i + 1));

System.out.print("Order ID: ");

int id = sc.nextInt();

System.out.print("Customer Name: ");

String name = sc.next();

System.out.print("Total Price: ");

double price = sc.nextDouble();

orders1[i] = new Order(id, name, price);

}

Order[] orders2 = Arrays.copyOf(orders1, n);

bubbleSort(orders1);

quickSort(orders2, 0, n - 1);

System.out.println("Orders sorted using Bubble Sort:");

display(orders1);

System.out.println("Orders sorted using Quick Sort:");

display(orders2);

}

}

**Time Complexity Analysis:**

* Bubble Sort: O(n²)
* Quick Sort: O(n log n) average, O(n²) worst-case

**Optimization Discussion:**  
Quick Sort is usually better for performance. Bubble Sort is mainly used for teaching or very small lists. In production systems, built-in Java sorting (Arrays.sort()) or Merge Sort may be preferable for consistency and efficiency.