Exercise 3: Sorting Customer Orders

1. Understand Sorting Algorithms

Common Sorting Algorithms:

🔹 Bubble Sort:

Repeatedly compares adjacent elements and swaps if they are in the wrong order.

Time Complexity:

Best: O(n) (already sorted)

Average/Worst: O(n²)

Use case: Educational purposes only; inefficient for large data.

🔹 Insertion Sort:

Builds the sorted array one element at a time.

Time Complexity:

Best: O(n),

Average/Worst: O(n²)

Use case: Efficient for small or nearly sorted datasets.

🔹 Quick Sort (Recommended):

Uses divide-and-conquer: selects a pivot and partitions the array around it.

Time Complexity:

Best/Average: O(n log n)

Worst: O(n²) (rare, when poorly chosen pivot)

🔹 Merge Sort:

Divide-and-conquer sorting by splitting and merging arrays.

Time Complexity: O(n log n) in all cases.

Use case: Stable sort; used in systems where worst-case performance is critical.

2. Setup

Create Order class:

public 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 + ", " + customerName + ", ₹" + totalPrice + "]";

}

}

3. Implementation

Bubble Sort & Quick Sort in OrderSorter.java:

public class OrderSorter {

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

Order temp = orders[j];

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

orders[j + 1] = temp;

}

}

}

}

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;

}

}

Main.java – Demonstration:

public class Main {

public static void main(String[] args) {

Order[] orders = {

new Order(501, "Alice", 2599.0),

new Order(502, "Bob", 899.0),

new Order(503, "Charlie", 3999.0),

new Order(504, "David", 1499.0),

new Order(505, "Eve", 299.0)

};

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

for (Order o : orders) System.out.println(o);

OrderSorter.bubbleSort(orders);

System.out.println("\nSorted by Bubble Sort:");

for (Order o : orders) System.out.println(o);

orders = new Order[] {

new Order(501, "Alice", 2599.0),

new Order(502, "Bob", 899.0),

new Order(503, "Charlie", 3999.0),

new Order(504, "David", 1499.0),

new Order(505, "Eve", 299.0)

};

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

System.out.println("\nSorted by Quick Sort:");

for (Order o : orders) System.out.println(o);

}

}

A screenshot of a computer

AI-generated content may be incorrect.

4. Analysis

Time Complexity Comparison:

Algorithm Best Case

Bubble Sort O(n)

Quick Sort O(n log n)