**SORTING CUSTOMER ORDERS**

**SORTING ALGORITHMS:**

-Bubble Sort: Repeatedly swaps adjacent elements if they are in the wrong order. Time Complexity: O(n^2)

- Insertion Sort: Builds the sorted array one item at a time. Time Complexity: O(n^2)

- Quick Sort: Picks a pivot and partitions the array around the pivot. Time Complexity: O(n log n) on average.

- Merge Sort: Divides the array and merges sorted halves. Time Complexity: O(n log n)

**PROGRAM:**

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 "Order(" + orderId + ", " + customerName + ", " + totalPrice + ")";

}

}

public class Main {

public static List<Order> bubbleSort(List<Order> orders) {

int n = orders.size();

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

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

if (orders.get(j).totalPrice > orders.get(j + 1).totalPrice) {

Order temp = orders.get(j);

orders.set(j, orders.get(j + 1));

orders.set(j + 1, temp);

}

}

}

return orders;

}

public static List<Order> quickSort(List<Order> orders) {

if (orders.size() <= 1) {

return orders;

}

double pivot = orders.get(orders.size() / 2).totalPrice;

List<Order> left = new ArrayList<>();

List<Order> middle = new ArrayList<>();

List<Order> right = new ArrayList<>();

for (Order o : orders) {

if (o.totalPrice < pivot) {

left.add(o);

} else if (o.totalPrice == pivot) {

middle.add(o);

} else {

right.add(o);

}

}

List<Order> sorted = new ArrayList<>();

sorted.addAll(quickSort(left));

sorted.addAll(middle);

sorted.addAll(quickSort(right));

return sorted;

}

public static void main(String[] args) {

List<Order> orders = Arrays.asList(

new Order(101, "AAA", 250.75),

new Order(102, "BBB", 99.99),

new Order(103, "CCC", 320.50),

new Order(104, "DDD", 150.00),

new Order(105, "EEE", 299.99)

);

List<Order> bubbleSortedOrders = new ArrayList<>(orders);

bubbleSort(bubbleSortedOrders);

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

for (Order order : bubbleSortedOrders) {

System.out.println(order);

}

List<Order> quickSortedOrders = quickSort(new ArrayList<>(orders));

System.out.println("\nQuick Sort Result:");

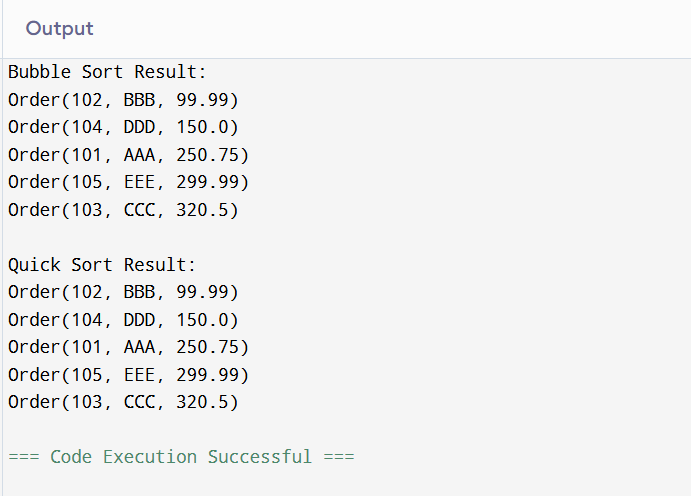
for (Order order : quickSortedOrders) {

System.out.println(order);

}

}

}



**PERFORMANCE ANALYSIS:**

- Bubble Sort has a time complexity of O(n^2), making it inefficient for large datasets.

- Quick Sort has an average time complexity of O(n log n), making it much faster for large lists.

**WHY QUICK SORT IS PREFERRED:**

- Quick Sort is significantly faster than Bubble Sort for large datasets.

- Bubble Sort performs too many unnecessary comparisons and swaps.