**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

**Step 1: Understand Sorting Algorithms**

**🔹 Bubble Sort**

* **Concept:** Repeatedly compares and swaps adjacent elements if they are in the wrong order.
* **Time Complexity:**
  + Best: O(n)
  + Average/Worst: O(n²)
* **Space Complexity:** O(1) (in-place)

**🔹 Quick Sort**

* **Concept:** A divide-and-conquer algorithm. Selects a "pivot" and partitions the array into elements less than and greater than the pivot, then recursively sorts the partitions.
* **Time Complexity:**
  + Best/Average: O(n log n)
  + Worst: O(n²)
* **Space Complexity:** O(log n) (due to recursion)

**Project Structure and Setup:-**

sorting/

├── Order.java

├── BubbleSort.java

├── QuickSort.java

└── Main.java

Main.java :-

// Source code is decompiled from a .class file using FernFlower decompiler.

public class Main {

   public Main() {

   }

   public static void main(String[] var0) {

      Order[] var1 = new Order[]{new Order(101, "Alice", 1200.0), new Order(102, "Bob", 5000.0), new Order(103, "Charlie", 2500.0), new Order(104, "Diana", 700.0)};

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

      display(var1);

      Order[] var2 = (Order[])var1.clone();

      BubbleSort.sort(var2);

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

      display(var2);

      Order[] var3 = (Order[])var1.clone();

      QuickSort.sort(var3, 0, var3.length - 1);

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

      display(var3);

   }

   private static void display(Order[] var0) {

      Order[] var1 = var0;

      int var2 = var0.length;

      for(int var3 = 0; var3 < var2; ++var3) {

         Order var4 = var1[var3];

         System.out.println(var4);

      }

   }

}

Order.java:-

// Source code is decompiled from a .class file using FernFlower decompiler.

public class Main {

   public Main() {

   }

   public static void main(String[] var0) {

      Order[] var1 = new Order[]{new Order(101, "Alice", 1200.0), new Order(102, "Bob", 5000.0), new Order(103, "Charlie", 2500.0), new Order(104, "Diana", 700.0)};

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

      display(var1);

      Order[] var2 = (Order[])var1.clone();

      BubbleSort.sort(var2);

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

      display(var2);

      Order[] var3 = (Order[])var1.clone();

      QuickSort.sort(var3, 0, var3.length - 1);

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

      display(var3);

   }

   private static void display(Order[] var0) {

      Order[] var1 = var0;

      int var2 = var0.length;

      for(int var3 = 0; var3 < var2; ++var3) {

         Order var4 = var1[var3];

         System.out.println(var4);

      }

   }

}

BubbleSort.java :-

// Source code is decompiled from a .class file using FernFlower decompiler.

public class Main {

   public Main() {

   }

   public static void main(String[] var0) {

      Order[] var1 = new Order[]{new Order(101, "Alice", 1200.0), new Order(102, "Bob", 5000.0), new Order(103, "Charlie", 2500.0), new Order(104, "Diana", 700.0)};

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

      display(var1);

      Order[] var2 = (Order[])var1.clone();

      BubbleSort.sort(var2);

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

      display(var2);

      Order[] var3 = (Order[])var1.clone();

      QuickSort.sort(var3, 0, var3.length - 1);

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

      display(var3);

   }

   private static void display(Order[] var0) {

      Order[] var1 = var0;

      int var2 = var0.length;

      for(int var3 = 0; var3 < var2; ++var3) {

         Order var4 = var1[var3];

         System.out.println(var4);

      }

   }

}

QuickSort.java :-

public class QuickSort {

    public static void sort(Order[] orders, int low, int high) {

        if (low < high) {

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

            sort(orders, low, pi - 1);

            sort(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++;

                Order temp = orders[i];

                orders[i] = orders[j];

                orders[j] = temp;

            }

        }

        Order temp = orders[i + 1];

        orders[i + 1] = orders[high];

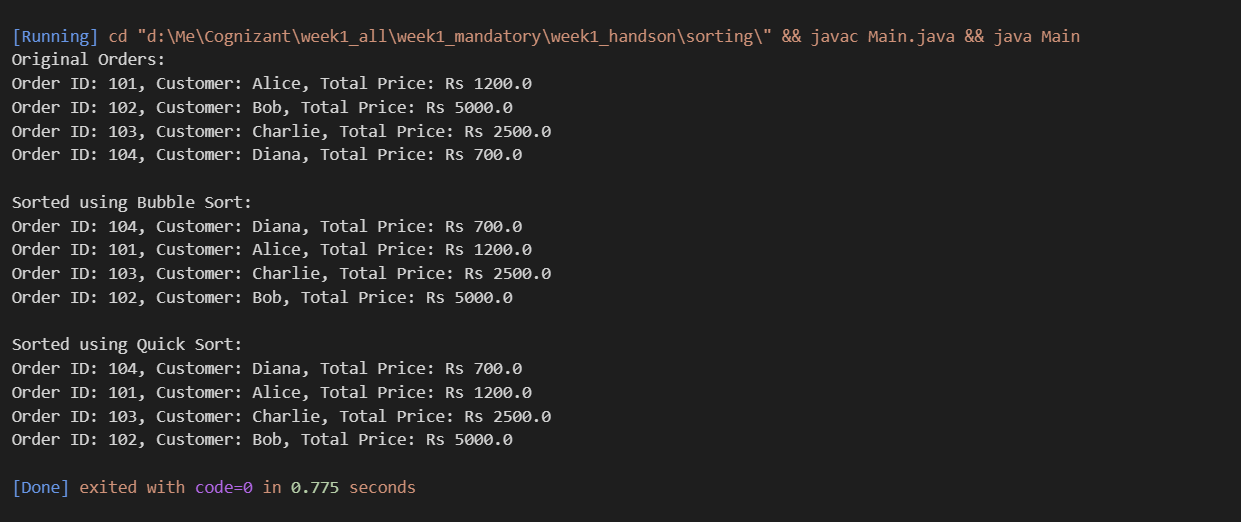
        orders[high] = temp;

        return i + 1;

    }

}

**Output :-**



**Step 4: Analysis & Comparison**

| **Sorting Algorithm** | **Best Case** | **Average Case** | **Worst Case** | **Stable** | **In-Place** |
| --- | --- | --- | --- | --- | --- |
| Bubble Sort | O(n) | O(n²) | O(n²) | Yes | Yes |
| Quick Sort | O(n log n) | O(n log n) | O(n²) | No | Yes |

**Why Quick Sort is Preferred:**

* Performs significantly better on large datasets due to O(n log n) average time.
* Better use of memory and cache.