**Solution 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.

**>> Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

**Sorting Algorithms Overview**:

1. **Bubble Sort**:
   * **Description**: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.
   * **Time Complexity**:
     + Best Case: O(n) (already sorted list)
     + Average Case: O(n^2)
     + Worst Case: O(n^2)
2. **Insertion Sort**:
   * **Description**: Builds the final sorted array one item at a time. It picks elements from the unsorted portion and inserts them into the correct position in the sorted portion.
   * **Time Complexity**:
     + Best Case: O(n) (already sorted list)
     + Average Case: O(n^2)
     + Worst Case: O(n^2)
3. **Quick Sort**:
   * **Description**: Divides the list into two smaller sub-lists: the low elements and the high elements. It then recursively sorts the sub-lists. Uses a pivot to partition the list.
   * **Time Complexity**:
     + Best Case: O(n log n)
     + Average Case: O(n log n)
     + Worst Case: O(n^2) (rare with good pivot selection)
4. **Merge Sort**:
   * **Description**: Divides the list into two halves, recursively sorts each half, and then merges the sorted halves.
   * **Time Complexity**:
     + Best Case: O(n log n)
     + Average Case: O(n log n)
     + Worst Case: O(n log n)

**Analysis**

**Time Complexity Comparison**

**Bubble Sort:**

* **Best Case:** O(n) (when the array is already sorted and the algorithm performs a single pass with no swaps)
* **Average Case:** O(n^2) (when elements are in random order and require multiple passes)
* **Worst Case:** O(n^2) (when the array is in reverse order and requires the maximum number of comparisons and swaps)

**Quick Sort:**

* **Best Case:** O(n log n) (when the pivot divides the array into nearly equal halves)
* **Average Case:** O(n log n) (typically observed in practice when pivots are well-chosen)
* **Worst Case:** O(n^2) (when the pivot choices consistently result in unbalanced partitions, such as always picking the smallest or largest element)

**>> Why Quick Sort is Generally Preferred Over Bubble Sort**

Use Cases:

* Quick Sort: Due to its efficiency, Quick Sort is often used in practice for general-purpose sorting tasks, especially when dealing with large datasets.
* Bubble Sort: Mostly used for educational purposes or small datasets where its simplicity outweighs its inefficiency.

Quick Sort is generally preferred over Bubble Sort because of its significantly better average and best-case time complexities. While Bubble Sort may be easier to understand and implement, its performance limitations make it impractical for larger datasets. Quick Sort's divide-and-conquer approach, along with its in-place sorting capability, makes it a more suitable choice for efficient sorting in most real-world applications.

**How to Run the code**

* Run Main.java file