### Step 1: Understand Sorting Algorithms

Bubble Sort:

- Description: A simple comparison-based algorithm where each pair of adjacent elements is compared and swapped if they are in the wrong order. This process is repeated until the array is sorted.

- Time Complexity:

- Best Case: O(n) (when the array is already sorted)

- Average Case: O(n^2)

- Worst Case: O(n^2)

- Space Complexity: O(1) (in-place sort)

Insertion Sort:

- Description: Builds the final sorted array one item at a time. It picks an element and places it at its correct position by comparing it with the already sorted elements.

- Time Complexity:

- Best Case: O(n) (when the array is already sorted)

- Average Case: O(n^2)

- Worst Case: O(n^2)

- Space Complexity: O(1) (in-place sort)

Quick Sort:

- Description: A divide-and-conquer algorithm that selects a 'pivot' element and partitions the array into two sub-arrays according to whether the elements are less than or greater than the pivot. It then recursively sorts the sub-arrays.

- Time Complexity:

- Best Case: O(n log n)

- Average Case: O(n log n)

- Worst Case: O(n^2) (when the smallest or largest element is always chosen as the pivot)

- Space Complexity: O(log n) (due to recursion stack)

Merge Sort:

- Description: A divide-and-conquer algorithm that divides the array into 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)

- Space Complexity: O(n) (auxiliary space for merging)

### Step 2: Setup

Order Class:

Create an `Order` class with attributes such as `orderId`, `customerName`, and `totalPrice`.

### Step 3: Implementation

Bubble Sort Implementation:

- Bubble Sort compares adjacent elements and swaps them if they are in the wrong order, repeating this process until the entire list is sorted.

Quick Sort Implementation:

- Quick Sort selects a pivot, partitions the array into elements less than and greater than the pivot, and recursively sorts the sub-arrays.

### Step 4: Analysis

Time Complexity Comparison:

- Bubble Sort:

- Best Case: O(n)

- Average Case: O(n^2)

- Worst Case: O(n^2)

- Quick Sort:

- Best Case: O(n log n)

- Average Case: O(n log n)

- Worst Case: O(n^2) (with poor pivot selection)

Why Quick Sort is Generally Preferred Over Bubble Sort:

1. Efficiency: Quick Sort is significantly more efficient on average, with a time complexity of O(n log n) compared to Bubble Sort's O(n^2).

2. Scalability: Quick Sort scales better with larger datasets, making it suitable for sorting large arrays quickly.

3. Practical Performance: Even though Quick Sort can have a worst-case time complexity of O(n^2), with good pivot selection strategies (like using the median or random pivot), it performs much better in practice.

### Example Analysis Using Code

1. Bubble Sort Test:

- The code creates a copy of the orders array and sorts it using Bubble Sort.

- The process involves nested loops, resulting in O(n^2) complexity.

- Suitable for small datasets or educational purposes but not recommended for large datasets due to inefficiency.

2. Quick Sort Test:

- The code creates a copy of the orders array and sorts it using Quick Sort.

- The process involves partitioning and recursive sorting, resulting in O(n log n) complexity on average.

- Suitable for large datasets due to its efficiency and scalability.

Conclusion:

- Quick Sort is generally preferred for sorting large datasets due to its efficient average-case time complexity of O(n log n).

- Bubble Sort is simple but inefficient for large datasets due to its O(n^2) time complexity. It's more suited for educational purposes or small datasets.