**Exercise 3: Sorting Customer Orders**

**Understanding Sorting Algorithms:**

**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)
  + Average Case: O(n^2)
  + Worst Case: O(n^2)

**Insertion Sort**

* **Description**: Builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
* **Time Complexity**:
  + Best Case: O(n) (already sorted)
  + Average Case: O(n^2)
  + Worst Case: O(n^2)

**Quick Sort**

* **Description**: A divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.
* **Time Complexity**:
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n^2) (rare, usually mitigated with good pivot selection)

**Merge Sort**

* **Description**: Also a divide-and-conquer algorithm. It divides the unsorted list into n sublists until each contains one element, then merges those sublists to produce new sorted sublists until there is only one sublist remaining.
* **Time Complexity**:
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n log n)

**Analysis:**

* **Bubble Sort**: O(n^2) - less suitable for large datasets.
* **Quick Sort**: Preferred for large datasets due to better average-case performance (O(n log n)).

**Preference Discussion**: Quick Sort is generally favoured for its efficiency over Bubble Sort in handling larger datasets. Quick Sort can be implemented in-place with O(log n) additional space, whereas Bubble Sort is also in-place but much slower.