**Exercise 3: Sorting Customer Orders**

**Explanation of different sorting algorithms:**

**Sorting algorithms** are techniques that arrange elements of a list or array in a specific order. Here are explanations for several common sorting algorithms:

* **Bubble Sort**:
  + **Description**: Bubble sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process repeats until the list is sorted.
  + **Time Complexity**:
    - **Best Case**: O(n) - when the list is already sorted.
    - **Average Case**: O(n^2) - average performance.
    - **Worst Case**: O(n^2) - when the list is sorted in reverse order.
* **Insertion Sort**:
  + **Description**: Insertion sort builds the final sorted array one item at a time. It iterates through the list, taking one element each time and inserting it into its correct position in the sorted part of the list.
  + **Time Complexity**:
    - **Best Case**: O(n) - when the list is already sorted.
    - **Average Case**: O(n^2) - average performance.
    - **Worst Case**: O(n^2) - when the list is sorted in reverse order.
* **Quick Sort**:
  + **Description**: Quick sort is a divide-and-conquer algorithm. It selects a pivot element and partitions the array around the pivot, such that elements less than the pivot are on its left and elements greater than the pivot are on its right. It then recursively sorts the subarrays.
  + **Time Complexity**:
    - **Best Case**: O(n log n) - occurs when the partitioning is balanced.
    - **Average Case**: O(n log n).
    - **Worst Case**: O(n^2) - can occur if the pivot selection leads to unbalanced partitions (e.g., always picking the smallest or largest element).
* **Merge Sort**:
  + **Description**: Merge sort is also a divide-and-conquer algorithm. It divides the array 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).

**Performance (Time Complexity) Comparison**

* **Bubble Sort**:
  + **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)

**Why Quick Sort is generally preferred over Bubble Sort:**

* **Efficiency**: Quick Sort has significantly better average and worst-case time complexity (O(n log n)) compared to Bubble Sort (O(n^2)). This means Quick Sort performs much faster, especially with larger datasets.
* **Divide-and-Conquer Approach**: Quick Sort's divide-and-conquer strategy makes it suitable for handling large datasets efficiently by reducing the problem size recursively.
* **Space Complexity**: Quick Sort generally has better space complexity than Bubble Sort, requiring only O(log n) additional space for recursion.
* **Practical Implementation**: Quick Sort is widely implemented in libraries and frameworks due to its efficient performance, making it a practical choice for real-world applications like sorting customer orders on an e-commerce platform.