**Exercise 3: Sorting Customer Orders**

**Step 1: Understand Sorting Algorithms**

**Bubble Sort**

* **Description:** A simple comparison-based algorithm where each pair of adjacent elements is compared, and they are 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)

**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) (when the array is already sorted)
  + Average-case: O(n^2)
  + Worst-case: O(n^2)
* **Space Complexity:** O(1)

**Quick Sort**

* **Description:** A divide-and-conquer algorithm. It selects a pivot element and partitions the array into two sub-arrays according to whether they 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 the recursive stack)

**Merge Sort**

* **Description:** Another divide-and-conquer algorithm. It divides the array into halves, 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) (due to the auxiliary arrays used for merging)

**Performance Comparison**

* **Bubble Sort:**
  + **Best-case:** O(n) - When the array is already sorted.
  + **Average-case:** O(n^2) - Due to the nested loops.
  + **Worst-case:** O(n^2) - When the array is sorted in reverse order.
  + **Space Complexity:** O(1) - Only requires a constant amount of additional space.
* **Quick Sort:**
  + **Best-case:** O(n log n) - When the pivot divides the array into two equal halves.
  + **Average-case:** O(n log n) - Expected runtime over a large number of inputs.
  + **Worst-case:** O(n^2) - When the pivot is the smallest or largest element repeatedly (can be mitigated by randomizing the pivot).
  + **Space Complexity:** O(log n) - Due to the recursive stack.

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

* **Efficiency:** Quick Sort is significantly faster for large datasets due to its O(n log n) average time complexity compared to Bubble Sort’s O(n^2) average time complexity.
* **Scalability:** Quick Sort performs well even as the dataset size increases, whereas Bubble Sort becomes impractically slow.
* **Practical Use:** Quick Sort is widely used in practice due to its efficiency and has optimizations that make its worst-case scenario rare.