**EXERCISE 3 : Sorting Customer Orders**

**Understanding Sorting Algorithms**

**1. 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 is repeated until the list is sorted.
* **Time Complexity:**
  + Best Case: O(n) when the array is already sorted.
  + Average Case: O(n²)
  + Worst Case: O(n²)
* **Space Complexity:** O(1) (in-place sorting).

**2. Insertion Sort:**

* **Description:** Insertion Sort builds the final sorted array one item at a time. It picks the next element and inserts it into its correct position in the already sorted part of the array.
* **Time Complexity:**
  + Best Case: O(n) when the array is already sorted.
  + Average Case: O(n²)
  + Worst Case: O(n²)
* **Space Complexity:** O(1) (in-place sorting).

**3. Quick Sort:**

* **Description:** Quick Sort is a divide-and-conquer algorithm. It selects a 'pivot' element from the array and partitions the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.
* **Time Complexity:**
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n²) (when the smallest or largest element is always chosen as the pivot).
* **Space Complexity:** O(log n) for recursive stack space.

**4. Merge Sort:**

* **Description:** Merge Sort is a divide-and-conquer algorithm. It divides the array into halves, recursively sorts each half, and then merges the two 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 need for an auxiliary array for merging).

**Analysis**

**Performance Comparison:**

* **Bubble Sort:**
  + **Time Complexity:**
    - Best Case: O(n)
    - Average Case: O(n²)
    - Worst Case: O(n²)
  + **Space Complexity:** O(1)
* **Quick Sort:**
  + **Time Complexity:**
    - Best Case: O(n log n)
    - Average Case: O(n log n)
    - Worst Case: O(n²)
  + **Space Complexity:** O(log n) for the recursive stack space.

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

1. **Efficiency:**
   * Quick Sort generally performs better than Bubble Sort, especially for large datasets. Its average and best-case time complexity of O(n log n) is significantly better than Bubble Sort's O(n²).
2. **Scalability:**
   * Quick Sort is more scalable for large arrays, making it suitable for real-world applications like sorting customer orders in an e-commerce platform.
3. **Practical Performance:**
   * Despite its worst-case time complexity being O(n²), Quick Sort is often faster in practice because of good cache performance and its divide-and-conquer approach, which works well with large datasets.