**Exercise 3: Sorting Customer Orders**

**Bubble Sort**

Bubble Sort is a simple sorting algorithm that 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.

Best-case Time Complexity: O(n) (when the array is already sorted)

Average-case Time Complexity: O(n^2)

Worst-case Time Complexity: O(n^2)

Space Complexity: O(1)

**Insertion Sort**

Insertion Sort 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.

Best-case Time Complexity: O(n) (when the array is already sorted)

Average-case Time Complexity: O(n^2)

Worst-case Time Complexity: O(n^2)

Space Complexity: O(1)

**Quick Sort**

Quick Sort is an efficient, divide-and-conquer sorting 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.

Best-case Time Complexity: O(n log n)

Average-case Time Complexity: O(n log n)

Worst-case Time Complexity: O(n^2) (when the pivot is the smallest or largest element every time)

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

**Merge Sort**

Merge Sort is an efficient, stable, divide-and-conquer sorting algorithm. It divides the input array into two halves, sorts them, and then merges them back together

Best-case Time Complexity: O(n log n)

Average-case Time Complexity: O(n log n)

Worst-case Time Complexity: O(n log n)

Space Complexity: O(n) (due to the need for an auxiliary array)

**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)

Why Quick Sort is Generally Preferred Over Bubble Sort

Efficiency: Quick Sort is significantly more efficient for large datasets due to its average-case time complexity of O(n log n), compared to Bubble Sort’s O(n^2).

Divide and Conquer: Quick Sort’s divide-and-conquer approach allows it to sort large arrays faster by breaking them down into smaller, more manageable sub-arrays.

Memory Usage: While both Quick Sort and Bubble Sort have O(1) auxiliary space complexity in their non-recursive implementations, the recursive nature of Quick Sort may lead to better use of cache memory, enhancing performance on large datasets.