**Sorting Customer Orders:**

1. **Bubble Sort:**

**Algorithm:**

* Repeatedly steps through the list
* Compares adjacent elements and swaps them if they're in the wrong order
* Continues until no swaps are needed

**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)

**Usage:**Simple to understand and implement but inefficient for large datasets

1. **Insertion Sort:**

**Algorithm:**

* Builds the final sorted array one item at a time
* Takes each element and inserts it into its correct position among the previously sorted elements

**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)

**Usage:** Simple implementation but Inefficient for large datasets

**3.Quick Sort:**

**Algorithm:**

* Selects a 'pivot' element from the array
* Partitions the other elements into two sub-arrays, according to whether they are less than or greater than the pivot
* Recursively sorts the sub-arrays

**Time Complexity:**

* Best case: O(n log n)
* Average case: O(n log n)
* Worst case: O(n^2) (rare, occurs with bad pivot choices)

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

**Usage:** Generally very efficient but Worst-case performance is poor.

1. **Merge Sort:**

**Algorithm:**

* Divides the unsorted list into n sublists, each containing one element
* Repeatedly merges 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)

**Space Complexity:** O(n)

**Usage:** Consistent performance regardless of input data but Requires extra space proportional to the input size

**Time Complexity Analysis:**

1. **Bubble Sort:**
   * **Best Case:** O(n) (when the list is already sorted)
   * **Average Case:** O(n^2)
   * **Worst Case:** O(n^2)
2. **Quick Sort:**
   * **Best Case:** O(n log n) (when the pivot divides the array into two equal halves)
   * **Average Case:** O(n log n)
   * **Worst Case:** O(n^2) (when the pivot selection is poor and results in unbalanced partitions)

**Quick Sort is preferred because:**

* Much faster for large datasets due to its average-case time complexity of O(n log n), compared to Bubble Sort's O(n^2).
* More scalable as data size increases
* Better cache efficiency.
* Quick Sort's divide-and-conquer approach leads to better performance.