**1. Understanding Sorting Algorithms**

**Common Sorting Algorithms**

1. **Bubble Sort**: A simple comparison-based 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.
   * **Time Complexity**: O(n²) in the worst and average cases.
   * **Best Case**: O(n) when the list is already sorted.
2. **Insertion Sort**: Builds the final sorted array one item at a time, by repeatedly picking the next item and inserting it into the correct position among the already sorted elements.
   * **Time Complexity**: O(n²) in the worst and average cases.
   * **Best Case**: O(n) when the list is already sorted.
3. **Quick Sort**: A divide-and-conquer algorithm that 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**: O(n log n) on average.
   * **Worst Case**: O(n²), but this is rare with good pivot selection.
   * **Best Case**: O(n log n).
4. **Merge Sort**: Another divide-and-conquer algorithm that divides the array into two halves, recursively sorts each half, and then merges the two sorted halves back together.
   * **Time Complexity**: O(n log n) in the best, worst, and average cases.
   * **Space Complexity**: O(n) due to the additional space required for the merging process.

**4. Analysis:**

**Performance Comparison**

* **Bubble Sort**:
  + **Time Complexity**: O(n²) in the average and worst cases, due to the nested loops.
  + **Space Complexity**: O(1), as it sorts the array in place.
* **Quick Sort**:
  + **Time Complexity**: O(n log n) on average, with a worst case of O(n²) (rare with good pivot selection).
  + **Space Complexity**: O(log n) due to recursive calls.

**Why Quick Sort is Preferred**

1. **Efficiency**: Quick Sort is generally faster than Bubble Sort for large datasets, as its average-case time complexity is O(n log n) compared to Bubble Sort's O(n²).
2. **Scalability**: Quick Sort scales better with increasing data sizes, making it more suitable for large e-commerce platforms with numerous orders.
3. **Flexibility**: Quick Sort can be optimized with techniques like choosing a good pivot (e.g., median-of-three) and using insertion sort for small subarrays, which can further improve its performance.

Overall, Quick Sort's superior time complexity and adaptability make it a more practical choice for sorting tasks in real-world applications.