**Understanding Sorting Algorithms**

**Bubble Sort**:

Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

Time Complexity: O(n²) in the worst and average cases, O(n) in the best case (when the array is already sorted).

**Insertion Sort**:

Builds the final sorted array one item at a time, with the assumption that the first element is already sorted.

**Time Complexity**: O(n²) in the worst and average cases, O(n) in the best case.

**Quick Sort**:

Uses the divide-and-conquer approach to partition the array into sub-arrays, which are then sorted independently.

**Time Complexity**: O(n log n) on average, O(n²) in the worst case.

**Merge Sort**:

Divides the array into halves, sorts each half, and then merges the sorted halves.

Time Complexity: O(n log n) in all cases.

**Implementation**

**Bubble Sort**:

Bubble Sort will repeatedly compare and swap adjacent elements if they are in the wrong order.

**Quick Sort**:

Quick Sort will partition the array around a pivot element and recursively sort the sub-arrays.

**Analysis**

**Performance Comparison**:

* Bubble Sort: O(n²) - Inefficient for large datasets due to its quadratic time complexity.
* **Quick Sort**: O(n log n) on average - More efficient for large datasets due to its logarithmic time complexity.

**Why Quick Sort is Preferred**:

* Quick Sort is generally preferred over Bubble Sort because it is significantly faster for large datasets. While Bubble Sort is simple and easy to implement, its performance degrades quickly as the size of the dataset increases. Quick Sort, on the other hand, efficiently handles large datasets with its divide-and-conquer approach, making it a better choice for sorting operations on an e-commerce platform.