**Sorting Algorithms Overview:**

* **Bubble Sort:** A simple comparison-based algorithm. It 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 For:** Small datasets or nearly sorted data.
  + **Drawbacks:** Inefficient on large datasets due to its quadratic time complexity.
* **Insertion Sort:** Another simple sorting algorithm. It builds the sorted array one element at a time by repeatedly picking the next element and inserting it into the correct position.
  + **Time Complexity:** O(n²) in the worst case, O(n) in the best case (already sorted array).
  + **Best For:** Small datasets or datasets that are nearly sorted.
  + **Drawbacks:** Inefficient on large datasets, but more efficient than Bubble Sort on small datasets.
* **Quick Sort:** A divide-and-conquer algorithm. It works by selecting a 'pivot' element and partitioning 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, O(n²) in the worst case.
  + **Best For:** General-purpose sorting, efficient for large datasets.
  + **Drawbacks:** Worst-case performance can be quadratic, but this is rare with good pivot selection strategies.
* **Merge Sort:** Also a divide-and-conquer algorithm. It divides the array into halves, recursively sorts each half, and then merges the sorted halves.
  + **Time Complexity:** O(n log n) in all cases.
  + **Best For:** Sorting large datasets, especially when stability (preserving the relative order of equal elements) is required.
  + **Drawbacks:** Requires additional memory for the temporary arrays during the merge process.

**Performance Comparison:**

* **Bubble Sort:**
  + Time Complexity: O(n²) in all cases (best, average, and worst).
  + Not efficient for large datasets due to quadratic time complexity.
  + Easy to understand and implement.
* **Quick Sort:**
  + Time Complexity: O(n log n) on average, but can degrade to O(n²) in the worst case (e.g., when the pivot is the smallest or largest element).
  + Typically much faster than Bubble Sort due to better average-case performance.
  + Efficient for large datasets.

**Why Quick Sort is Preferred:**

Quick Sort is generally preferred over Bubble Sort because it has a significantly better average-case time complexity, making it suitable for large datasets. While the worst-case scenario for Quick Sort can be as bad as Bubble Sort, this can usually be mitigated with good pivot selection strategies (e.g., choosing the median, random pivot selection). Additionally, Quick Sort's in-place sorting mechanism (minimal extra memory usage) makes it more memory-efficient compared to other sorting algorithms like Merge Sort.