**1. Understand Sorting Algorithms:**

* **Bubble Sort:**
  + Bubble Sort is a simple comparison-based sorting algorithm.
  + It works by repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order.
  + Time Complexity: O(n²) in the worst and average cases.
  + Space Complexity: O(1) as it is an in-place sorting algorithm.
  + It is inefficient for large datasets.
* **Insertion Sort:**
  + Insertion Sort builds the sorted array one item at a time.
  + It takes each element and places it in its correct position among the previously sorted elements.
  + Time Complexity: O(n²) in the worst case, but O(n) in the best case when the array is already sorted.
  + Space Complexity: O(1) as it is also an in-place sorting algorithm.
  + It is more efficient than Bubble Sort for small datasets or nearly sorted datasets.
* **Quick Sort:**
  + Quick Sort is a highly efficient divide-and-conquer sorting algorithm.
  + It works by selecting a 'pivot' element, partitioning the array around the pivot, and then recursively sorting the sub-arrays.
  + Time Complexity: O(n log n) on average, but O(n²) in the worst case (rare).
  + Space Complexity: O(log n) due to recursion.
  + It is generally faster and more efficient for large datasets.
* **Merge Sort:**
  + Merge Sort is a stable, comparison-based sorting 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, average, and worst).
  + Space Complexity: O(n) due to the auxiliary array used for merging.
  + It is more memory-intensive but guarantees consistent performance.
  + Inefficient for large datasets due to its quadratic time complexity.
  + Simple to implement but not suitable for large datasets.

**4. Analysis:**

* **Bubble Sort:**
  + Time Complexity: O(n²)
  + Inefficient for large datasets due to its quadratic time complexity.
  + Simple to implement but not suitable for large datasets.
* **Quick Sort:**
  + Time Complexity: O(n log n) on average.
  + More efficient and faster for large datasets compared to Bubble Sort.
  + Uses divide-and-conquer approach, which leads to better performance.
* **Why Quick Sort is Preferred:**
  + Quick Sort generally outperforms Bubble Sort due to its average-case time complexity of O(n log n), making it more suitable for larger datasets.
  + While the worst-case time complexity of Quick Sort is O(n²), this can be mitigated by using techniques like choosing a good pivot (e.g., median-of-three or random pivot).
  + Quick Sort is more efficient in terms of both time and space compared to Bubble Sort, making it a preferred choice in practical applications.

.