**1. Understand Sorting Algorithms**

**Bubble Sort**

**Bubble Sort** is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The process repeats until the list is sorted.

* **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) (in-place sorting)

**Insertion Sort**

**Insertion Sort** builds the final sorted array one item at a time. It takes each element and inserts it into its correct position in a sorted portion of the array.

* **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) (in-place sorting)

**Quick Sort**

**Quick Sort** is a divide-and-conquer algorithm. It selects a 'pivot' element and partitions the array into two sub-arrays, according to whether elements are less than or greater than the pivot. The sub-arrays are then sorted recursively.

* **Time Complexity**:
  + **Best-case**: O(n log n)
  + **Average-case**: O(n log n)
  + **Worst-case**: O(n^2) (when the pivot is the smallest or largest element)
* **Space Complexity**: O(log n) (due to recursion stack)

**Merge Sort**

**Merge Sort** is a divide-and-conquer algorithm that divides the array into halves, sorts each half, and then merges the sorted halves.

* **Time Complexity**:
  + **Best-case**: O(n log n)
  + **Average-case**: O(n log n)
  + **Worst-case**: O(n log n)
* **Space Complexity**: O(n) (additional space for merging)

**4. Analysis**

**Time Complexity Comparison**

* **Bubble Sort**:
  + **Best-case**: O(n) if the list is already sorted.
  + **Average-case**: O(n^2).
  + **Worst-case**: O(n^2).
* **Quick Sort**:
  + **Best-case**: O(n log n).
  + **Average-case**: O(n log n).
  + **Worst-case**: O(n^2) (can be mitigated with better pivot selection strategies).

**Why Quick Sort is Preferred Over Bubble Sort**

* **Efficiency**: Quick Sort has a better average and best-case time complexity (O(n log n)) compared to Bubble Sort’s O(n^2).
* **Scalability**: Quick Sort is more suitable for larger datasets due to its logarithmic factor in time complexity.
* **Practical Use**: Quick Sort is often preferred in practice due to its in-place sorting capability and efficient average-case performance. Although it has a worst-case time complexity of O(n^2), this can be mitigated with techniques such as random pivot selection or median-of-three pivot selection.

In conclusion, while Bubble Sort is easy to implement and understand, Quick Sort is generally preferred for performance reasons, especially with larger datasets.