**Understand Sorting Algorithms**

**Bubble Sort:**

* **Description:** A simple comparison-based algorithm where each pair of adjacent elements is compared, and the elements are swapped if they are in the wrong order. This process is repeated until the list is sorted.
* **Time Complexity:** O(n^2) in the worst and average case, O(n) in the best case when the list is already sorted.
* **Use Case:** Useful for small lists or when the list is nearly sorted.

**Insertion Sort:**

* **Description:** Builds the final sorted array one item at a time. It picks the next element and inserts it into the correct position within the sorted part of the array.
* **Time Complexity:** O(n^2) in the worst and average case, O(n) in the best case when the list is already sorted.
* **Use Case:** Efficient for small lists or partially sorted lists.

**Quick Sort:**

* **Description:** A divide-and-conquer algorithm that picks a pivot element, partitions the array into elements less than and greater than the pivot, and recursively sorts the subarrays.
* **Time Complexity:** O(n log n) on average, O(n^2) in the worst case.
* **Use Case:** Widely used for its average-case efficiency and suitability for large datasets.

**Merge Sort:**

* **Description:** A divide-and-conquer algorithm that divides the array into halves, recursively sorts each half, and then merges the sorted halves.
* **Time Complexity:** O(n log n) in all cases.
* **Use Case:** Preferred for large datasets and linked lists due to its stable sorting nature.

**Analysis**

**Time Complexity Comparison:**

* **Bubble Sort:**
  + **Best Case:** O(n) when 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), but this is rare and can be mitigated with good pivot selection strategies