**Step 1: Understand Sorting Algorithms**

**1. Bubble Sort:**

* **Description:** Compares adjacent elements and swaps them if they are in the wrong order. This process is repeated until the entire array is sorted.
* **Time Complexity:**
  + **Best Case:** O(n) – When the array is already sorted (optimized version).
  + **Average Case:** O(n^2) – Requires multiple passes to sort.
  + **Worst Case:** O(n^2) – When the array is in reverse order.
* **Pros:** Simple to implement.
* **Cons:** Inefficient for large datasets.

**2. Insertion Sort:**

* **Description:** Builds the sorted array one item at a time, by inserting each new item into its correct position among the previously sorted items.
* **Time Complexity:**
  + **Best Case:** O(n) – When the array is already sorted.
  + **Average Case:** O(n^2) – When the array is randomly ordered.
  + **Worst Case:** O(n^2) – When the array is in reverse order.
* **Pros:** Efficient for small datasets or nearly sorted data.
* **Cons:** Inefficient for large datasets.

**3. Quick Sort:**

* **Description:** Divides the array into smaller sub-arrays based on a pivot element, sorting the sub-arrays independently. The process is recursive.
* **Time Complexity:**
  + **Best Case:** O(n log n) – When the pivot divides the array evenly.
  + **Average Case:** O(n log n) – Average case performance.
  + **Worst Case:** O(n^2) – When the pivot is the smallest or largest element (rare with good pivot selection).
* **Pros:** Very efficient for large datasets; generally faster than other O(n log n) algorithms.
* **Cons:** Performance can degrade if not implemented with a good pivot selection strategy.

**4. Merge Sort:**

* **Description:** 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)
* **Pros:** Consistently efficient; stable sort.
* **Cons:** Requires additional space proportional to the array size.

**Step 4: Analysis**

**Time Complexity Comparison:**

1. **Bubble Sort:**
   * **Best Case:** O(n) – When the array is already sorted (with optimization).
   * **Average Case:** O(n^2) – Requires multiple passes to sort.
   * **Worst Case:** O(n^2) – When the array is in reverse order.
2. **Quick Sort:**
   * **Best Case:** O(n log n) – When the pivot divides the array evenly.
   * **Average Case:** O(n log n) – Average case performance.
   * **Worst Case:** O(n^2) – When the pivot is the smallest or largest element (rare with good pivot selection).

**Why Quick Sort is Preferred:**

* **Efficiency:** Quick Sort has a better average-case time complexity (O(n log n)) compared to Bubble Sort's O(n^2). This makes it much more efficient for large datasets.
* **Performance:** In practice, Quick Sort often performs faster than Bubble Sort because it makes fewer comparisons and swaps.