**1. Understand Sorting Algorithms**

**Q: Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

**Bubble Sort:**

* **Description:** Bubble Sort continuously traverses the list, compares adjacent elements, and swaps them if they are in the incorrect order. This process is repeated until the entire list is sorted.
* **Time Complexity:**
  + **Best Case:** O(n) - When the list is already sorted.
  + **Average Case:** O(n^2) - Typical performance with random data.
  + **Worst Case:** O(n^2) - When the list is sorted in reverse order.

**Insertion Sort:**

* **Description:** Insertion Sort builds the final sorted array one element at a time by repeatedly taking the next unsorted element and inserting it into its correct position within the already sorted portion.
* **Time Complexity:**
  + **Best Case:** O(n) - When the list is already sorted.
  + **Average Case:** O(n^2) - Average performance with random data.
  + **Worst Case:** O(n^2) - When the list is sorted in reverse order.

**Quick Sort:**

* **Description:** Quick Sort is a divide-and-conquer algorithm that selects a pivot element, partitions the array into elements less than and greater than the pivot, and recursively sorts the sub-arrays.
* **Time Complexity:**
  + **Best Case:** O(n log n) - When the pivot selection is optimal.
  + **Average Case:** O(n log n) - Typical performance with random data.
  + **Worst Case:** O(n^2) - When the pivot selection is poor, though this is rare with good pivot strategies.

**Merge Sort:**

* **Description:** Merge Sort is a divide-and-conquer algorithm that divides the array into two halves, recursively sorts each half, and then merges the sorted halves into a single sorted array.
* **Time Complexity:**
  + **Best Case:** O(n log n) - Consistent performance.
  + **Average Case:** O(n log n) - Consistent performance.
  + **Worst Case:** O(n log n) - Consistent performance.

**4. Analysis**

**Q: Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

**Bubble Sort:**

* **Best Case:** O(n) - When the list is already sorted.
* **Average Case:** O(n^2) - Average performance with random data.
* **Worst Case:** O(n^2) - When the list is in reverse order.

**Quick Sort:**

* **Best Case:** O(n log n) - When the pivot selection divides the array evenly.
* **Average Case:** O(n log n) - Typical performance with random data.
* **Worst Case:** O(n^2) - When the pivot selection is consistently poor, though this is less common with effective pivot strategies.

**Q: Discuss why Quick Sort is generally preferred over Bubble Sort.**

**1. Efficiency:**

* **Quick Sort** has an average and best-case time complexity of O(n log n), making it significantly faster for large datasets compared to **Bubble Sort**'s O(n^2) time complexity.

**2. Scalability:**

* **Quick Sort** efficiently handles large arrays and is well-suited for in-place sorting, whereas **Bubble Sort** struggles with large datasets due to its quadratic time complexity.

**3. Practical Performance:**

* In practice, **Quick Sort** is generally faster than **Bubble Sort** for most inputs due to its lower constant factors and better utilization of CPU cache, resulting in improved performance overall.