**Understand Sorting Algorithms**

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

**Bubble Sort**

**Description**: Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted. Time Complexity: Best Case: O(n) Average Case: O(n^2) Worst Case: O(n^2)

**Insertion Sort**

**Description**: Insertion Sort builds the sorted array one element at a time by repeatedly picking the next element and inserting it into its correct position. Time Complexity: Best Case: O(n) Average Case: O(n^2) Worst Case: O(n^2)

**Quick Sort**

**Description**: Quick Sort is 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 sub-arrays. Time Complexity: Best Case: O(n log n) Average Case: O(n log n) Worst Case: O(n^2)

**Merge Sort**

**Description**: Merge Sort is a divide-and-conquer algorithm that splits the array into halves, recursively sorts them, 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)

**Analysis**

**Compare the performance (time complexity) of Bubble Sort and Quick Sort?**

**Bubble Sort:**

Best Case: O(n) 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 rare with good pivot selection)

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

1. Efficiency: Quick Sort's average and best-case time complexity of O(n log n) makes it much faster for large datasets compared to Bubble Sort’s O(n^2).
2. Scalability: Quick Sort can handle large arrays more efficiently and is well-suited for in-place sorting.
3. Practical Performance: In practice, Quick Sort is faster than Bubble Sort for most inputs due to lower constants and better cache performance.