1. Bubble Sort
Approach:
Repeatedly swaps adjacent elements if they are in the wrong order.
Time Complexity:
Best Case: O(N) (already sorted, optimized version)
Average/Worst Case: $O(N^2)$
Stability: Stable
Suitability: Very small datasets only
2. Merge Sort
Approach:
Uses Divide & Conquer to split the array, sort subarrays, and merge them.
Time Complexity:
Best/Average/Worst Case: O(N log N)
Stability: Stable
Space Complexity: O(N) (extra memory required)
3. Quick Sort
Approach:
Partitions the array around a pivot and recursively sorts partitions.
Time Complexity:
Best/Average Case: O(N log N)
Worst Case: $O(N^2)$ (poor pivot choice)
Stability: Unstable
Space Complexity: O(log N) (in-place recursion)

| Dataset Size (N) | Bubble Sort ($O(N^2)$) | Merge Sort (O(N log N)) | Quick Sort (O(N log N)) |
| ---------------- | ------------------- | ---------------------- | ---------------------- |
| 1,000 | 50 ms | 5 ms | 3 ms |
| 10,000 | 5 s | 50 ms | 30 ms |
| 1,000,000 | Unfeasible (>1 hr) | 3 s | 2 s |

Bubble Sort
Performance degrades drastically as data grows.
Completely impractical for large datasets.
Merge Sort
Consistent performance regardless of data order.
Preferred when stability is required.
Uses additional memory.
Quick Sort
Generally the fastest in practice.
Worst-case can be avoided using randomized or median-of-three pivots.
Not stable.
Final Conclusion
* Bubble Sort is unsuitable for large datasets due to quadratic time complexity.
* Merge Sort is reliable and stable with predictable performance.
* Quick Sort offers the best average performance and is widely used in real-world systems.

Expected Result:
Bubble Sort is impractical for large datasets, while Merge Sort and Quick Sort scale efficiently with data size.