Theory Questions and Short Answers on Parallel Sorting with OpenMP

1. What is the main objective of this program?

To compare the performance of Bubble Sort and Merge Sort using OpenMP parallelization.

2. Which two sorting algorithms are used in this code?

Bubble Sort and Merge Sort.

3. What is OpenMP?

OpenMP is an API that enables parallel programming in shared-memory environments using compiler directives.

4. How is time measured in this program?

Using clock() function from the <ctime> library.

5. How does Bubble Sort work?

It repeatedly swaps adjacent elements if they are in the wrong order until the array is sorted.

6. What is the time complexity of Bubble Sort?

$O(n^2)$ in the worst and average case.

7. How is OpenMP applied in Bubble Sort?

By parallelizing the for-loop using #pragma omp parallel for.

8. Why is Bubble Sort parallelization unsafe in this code?

Because multiple threads may swap adjacent elements simultaneously, leading to data races.

9. What is Merge Sort?

A divide-and-conquer sorting algorithm that splits the array, sorts both halves, and merges them.

10. What is the time complexity of Merge Sort?

O(n log n) in all cases.

11. How is OpenMP used in Merge Sort?

By using #pragma omp parallel sections to sort two halves of the array concurrently.

12. Which sorting algorithm is more efficient for large data?

Merge Sort, due to its lower time complexity and better scalability with parallelism.

13. What does the merge() function do?

It combines two sorted subarrays into a single sorted array.

14. What are #pragma omp section and #pragma omp parallel sections used for?

To divide work among multiple threads for concurrent execution of code blocks.

15. Why is Merge Sort more suitable for parallelism than Bubble Sort?

Because its recursive structure allows independent sorting of subarrays without shared data conflicts.