Theory Questions and Short Answers on Parallel Computations with OpenMP

1. What is the purpose of this program?

To compute the minimum, maximum, sum, and average of an array using parallel processing with OpenMP.

2. Which OpenMP directive is used to parallelize loops in this code?

#pragma omp parallel for.

3. Why might the parallelMin() and parallelMax() functions produce incorrect results?

Because multiple threads may update the shared variable (min_val or max_val) at the same time, causing race conditions.

4. What is a race condition?

A race condition occurs when multiple threads access and modify a shared variable concurrently, leading to unpredictable results.

5. How can race conditions be prevented in OpenMP?

By using reduction clauses or #pragma omp critical to manage shared variable updates safely.

6. What does the reduction(+:sum) clause do?

It tells OpenMP to perform the sum operation in parallel and combine the results at the end safely.

7. What data structure is used to store input values?

std::vector<int> from the C++ Standard Library.

8. How is the average calculated?

By dividing the sum of elements by the number of elements.

9. Is it safe to parallelize a loop that modifies a shared variable without synchronization?

No, it can lead to incorrect results due to race conditions.

10. Can OpenMP improve the performance of this program significantly?

It can for large datasets, but correctness and thread safety must be ensured first.

11. Why should vector<int> vec be passed by reference instead of value?

Passing by reference avoids unnecessary copying and improves performance.

12. What header file must be included to use OpenMP?

#include <omp.h>

13. What is the default number of threads used by OpenMP?

It depends on the system, but it usually matches the number of available CPU cores.

14. Can we manually set the number of threads in OpenMP?

Yes, using omp_set_num_threads() or the OMP_NUM_THREADS environment variable.

15. Why is parallelizing simple operations on small datasets sometimes inefficient?

Because the overhead of managing threads may outweigh the performance gains.