Q1) Write an MPI program to multiply two 3x3 matrices (AxB). Use four processors, one processor acts as a master/server. All worker processors will have a copy of Matrix A. The master processor will send one column from matrix B to each processor. Each processor calculates one column of the result matrix and sends it back to the master. The master prints out the result. Use collective communications.

Q2) Write an MPI program to multiply 100x100 matrices (AxB). Use four processors, one acts as a master. The master processor will manage the process of multiplication. It will be waiting for workers to send a ready message. Once the master receives a ready message, it will send one row and one column to be multiplied. Once the worker finishes, it sends the result back to master and asks for another row and column. This continues until the result matrix is calculated. The master then prints the time taken to finish this process. Use collective communications.

Q3) Update questions 2 above such that a copy of matrices A and B are saved in all processors and the master only sends the indices of the row and column to be multiplied. The master reports the time when the process is done.

Q4) Generate an array (5x5) of integers using random numbers. Use 10 processors. Split the processors into two groups, even and odd processors. Distribute the array to both groups. Each processor in the odd group should find the minimum value for one row, the even group the maximum value of one column. Use MPI_Reduce operation to find the minimum and maximum value on one processor and prints them out.

Q5) In sorting problem, sorting algorithm puts the elements of a list in a certain order either ascending or
descending. In this project, you are asked to parallelize some of comparison-based sorting algorithms. Some comparison-based sorting algorithms perform exchange of adjacent elements repeatedly to produce the sorted list. On the other hand, some other comparison-based algorithms divide the list into two sub-lists and sort these sub-lists recursively. This technique is called divide and conquer. Comparison-based exchange algorithms are usually slower than divide and conquer algorithms. Examples of exchange algorithms are bubble sort, insertion sort, and selection sort. Popular examples of divide and conquer algorithms are quick sort and merge sort.

Parallelizing sorting algorithms is not a trivial process due to data dependency. In this project we will try to parallelize two algorithms (one from each approach) and compare between the performance of the sequential version and the parallel version of these algorithms. We will try to parallelize selection sort from exchange category and quick sort from divide and conquer category. The parallelizing of the algorithms can be done in three main steps. Distribution step, where a master/server process generates a large random array and distributes it evenly to slave/client processes. Sorting step, where each slave/client processor sorts its sub-list. Finally, Merge process, where processors exchange their sub-list in a merge-fashion and send them back to the master/server processor.

Here are the main steps in pseudo code for your parallelizing project.

$ParallelSorting(n)$

1. $master\ process\ generates\ a\ list\ of\ n\ random\ numbers\ between\ 1\ and\ 1000$
2. $master\ process\ records\ current\ time\ t1$
3. $master\ process\ scatters/distributes\ the\ list\ to\ all\ slaves\ including\ itself$
4. $slave\ processes\ sort\ the\ sub-list\ using\ the\ sorting\ algorithm$
5. $if\ processor\ id\ is\ even$
6. $send\ even\ sub-list\ to\ processor\ id+1$

7. *receive odd sub − list from processor id* + 1
8. *merge two sub − lists*
9. *replace even sub − list with the first half of the merged list*
10. *else*
11. *receive even sub − ⊠ist from processor id* − 1
12. *send odd sub − list to processor id* − 1
13. *merge two sub − lists*
14. *replace odd sub − list with the second half of the merged list*
15. *slave process send their sub − lists to master process*
16. *master process records current time t*2
17. *master process writes out the sor⊠ed list and reports t*2 − *t*1

SequentialSorting(*n*)

1. *generate a list of n random numbers between* 1 *and* 1000
2. *record current time t*1
3. *sort the list using sorting algorithm*
4. *record current time t*2
5. *write out the sorted list and report the time t*2 − *t*1