Problem 1

The MPI count-sort program reads a binary file of 4-byte integers and redistributes the data evenly across all processes. Each process builds a local histogram of values in the range 0–999 and then participates in an MPI_Reduce to sum these histograms on the root rank. Once the global counts are available, the root process reconstructs the sorted sequence by writing each integer the number of times indicated in the histogram. Finally, the root reopens the same file in update mode and overwrites it with the sorted data. File I/O is handled with fread and fwrite, and the file name is broadcast from rank 0 using MPI_Bcast. This approach guarantees that every integer is accounted for and ends up in ascending order without any extra output to the console. Memory is allocated just once on each rank, and all buffers are freed before calling MPI_Finalize.

Problem 2

The pthread pi-approximation program uses the midpoint rule to estimate the area under $1/(1 + x^2)$ from 0 to 1 and multiplies the result by 4. The total number of terms and the number of threads are both taken from the command line. The work is split evenly so each thread computes a partial sum over its range of midpoints. After all threads join, their partial sums are added together and scaled by 4/n to produce the final approximation of pi. Timing with the time command shows that increasing threads up to the number of CPU cores shortens the real execution time, while going beyond the core count offers little extra benefit. The code sticks to simple loops and standard pthread calls, matching exactly what the assignment asks for. Memory usage is minimal and the result is printed with fifteen decimal places to show convergence as the number of terms grows.