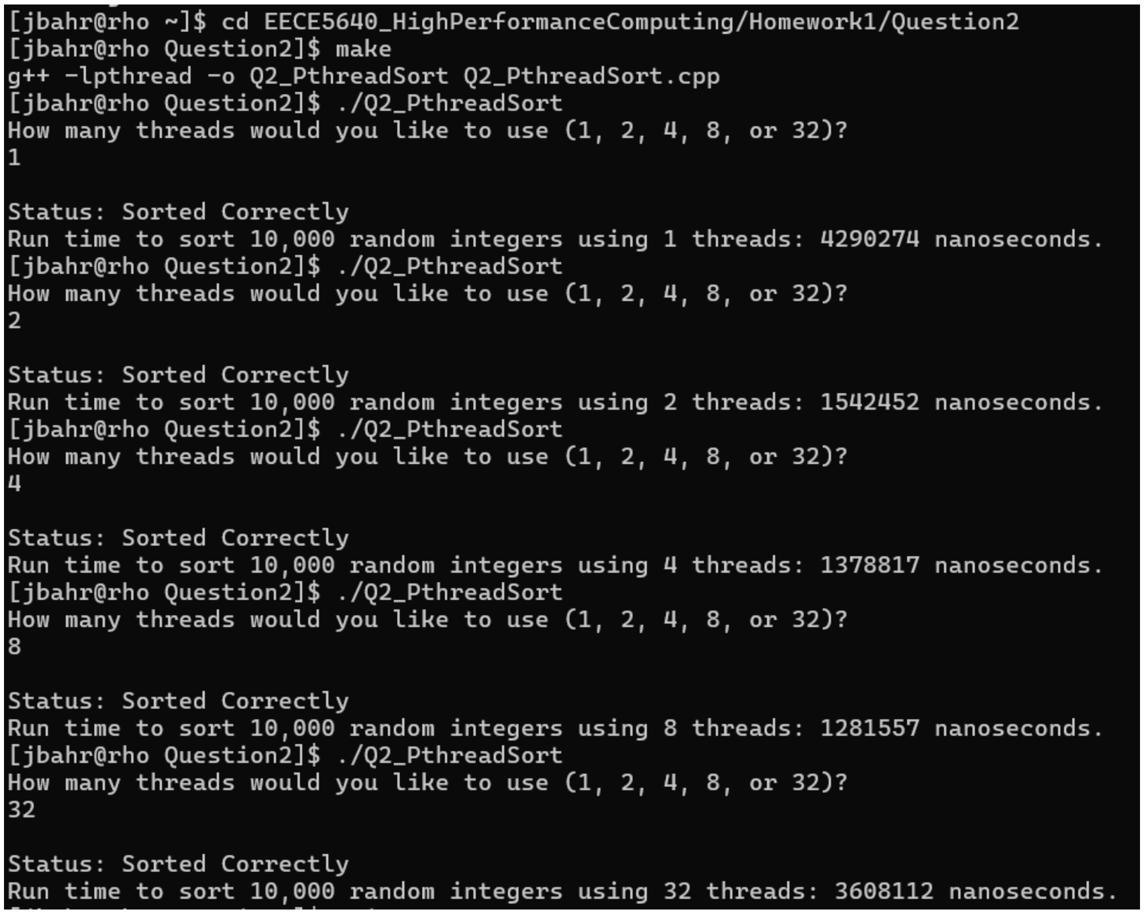
Justin Bahr
NUID: 001592707

High Performance Computing
Homework 1 - Question 2

a. I ran my parallel mergeSort program with 1, 2, 4, 8, and 32 threads on the COE Vector system. Below is a screenshot of the command line and a table with results:



```
[jbahr@rho ~]$ cd EECE5640_HighPerformanceComputing/Homework1/Question2
[jbahr@rho Question2]$ make
g++ -lpthread -o Q2_PthreadSort Q2_PthreadSort.cpp
[jbahr@rho Question2]$ ./Q2_PthreadSort
How many threads would you like to use (1, 2, 4, 8, or 32)?
1

Status: Sorted Correctly
Run time to sort 10,000 random integers using 1 threads: 4290274 nanoseconds.
[jbahr@rho Question2]$ ./Q2_PthreadSort
How many threads would you like to use (1, 2, 4, 8, or 32)?
2

Status: Sorted Correctly
Run time to sort 10,000 random integers using 2 threads: 1542452 nanoseconds.
[jbahr@rho Question2]$ ./Q2_PthreadSort
How many threads would you like to use (1, 2, 4, 8, or 32)?
4

Status: Sorted Correctly
Run time to sort 10,000 random integers using 4 threads: 1378817 nanoseconds.
[jbahr@rho Question2]$ ./Q2_PthreadSort
How many threads would you like to use (1, 2, 4, 8, or 32)?
8

Status: Sorted Correctly
Run time to sort 10,000 random integers using 8 threads: 1281557 nanoseconds.
[jbahr@rho Question2]$ ./Q2_PthreadSort
How many threads would you like to use (1, 2, 4, 8, or 32)?
32

Status: Sorted Correctly
Run time to sort 10,000 random integers using 32 threads: 3608112 nanoseconds.
```

Number of Threads	1	2	4	8	32
Runtime (ns)	4290274	1542452	1378817	1281557	3608112

Adding more cores continued to decrease the sort's runtime, but when I ran it with 32 threads, it slowed down significantly, meaning the extra workload of creating and joining many Pthreads eventually outweighs the speedup gained by adding more threads. Through testing, I noticed that the fastest runs were with either 4 or 8 threads, leading to a maximum speedup of about 3.35.

b. I chose to implement a parallelMergeSort() function with the desired number of threads as an input parameter. Treating the number of Pthreads as a variable was challenging when partitioning the array since I had to calculate and set the left and right bounds of thread arguments iteratively instead of simply typing in the bounds based on a set number of threads. Another challenge when using Pthreads is that they have very specific requirements for input parameters, which must be passed by reference. Due to this, I had to create a structure for thread arguments. Additionally, since the start routine's arguments must be of type void*, so I had to modify this from my original mergeSort() function. Finally, since I used an implementation of mergeSort that is not in-place, I did not need to use mutexes or locks, but for an in-place sorting algorithm, these would likely need to be added in order to prevent issues with data synchronization.

c. Even though this question involves a problem with fixed size, I can abstract that using this parallel function on a larger dataset would allow me to sort more integers with 2, 4, 8, or 32 threads in the same amount of time it would take to sort fewer integers using a single-threaded function. This concept of weak scaling is described by Gustafson's Law. Here, the serial workload remains the same, but the data set increases, allowing more (and a higher percentage) of tasks to be executed in parallel.

As I increased the number of threads, strong scaling was evident. I could clearly see that if multiple processors are working, splitting the entire sort into smaller sorts operating concurrently, it is possible to sort the same amount of numbers faster than a single-thread program could. Strong scaling can be described using Amdahl's Law. For example, as I switched from using one thread to eight threads, I observed a speedup of about 3.35.

$$S = \frac{1}{(1-P)+\frac{P}{n}}, \quad 3.35 = \frac{1}{(1-P)+\frac{P}{8}}, \quad P = 0.8$$

This means when I ran the program with 8 Pthreads, approximately 80% of the program was parallelizable and only 20% was serial. While adding more Pthreads increases the number of threads to work concurrently, it also adds more time to the serial workload to create and join the Pthreads.