Approach Summary

My first idea was to evenly distribute the amount of numbers that each thread would have to work on, and that is exactly what I did for the final submission. I calculated the range of numbers that would have to be checked and distributed them evenly for each thread that would be used. Then I started the multi-threaded work with a loop for each thread and set the ranges of numbers that each of them would work with. I used AtomicLong variables to store the end results, being the sum of all primes, the number of primes, and the 10 highest primes, which were then put into a “results.txt” file generated by the code. Additionally, my method of acquiring the 10 highest prime numbers was fairly simple. I created an AtomicLong array of size 10 and updated it with the highest numbers found in each range, and at the end only the final one would remain, giving us the desired result. I also collected the time it took for the program to run, from creating the thread pool to ending it.

**Reason for Approach**

After thinking about other solutions, I ended up realizing that the first one I came up with was the most efficient, given that it would always guarantee that each thread would have an equal amount of work to do.