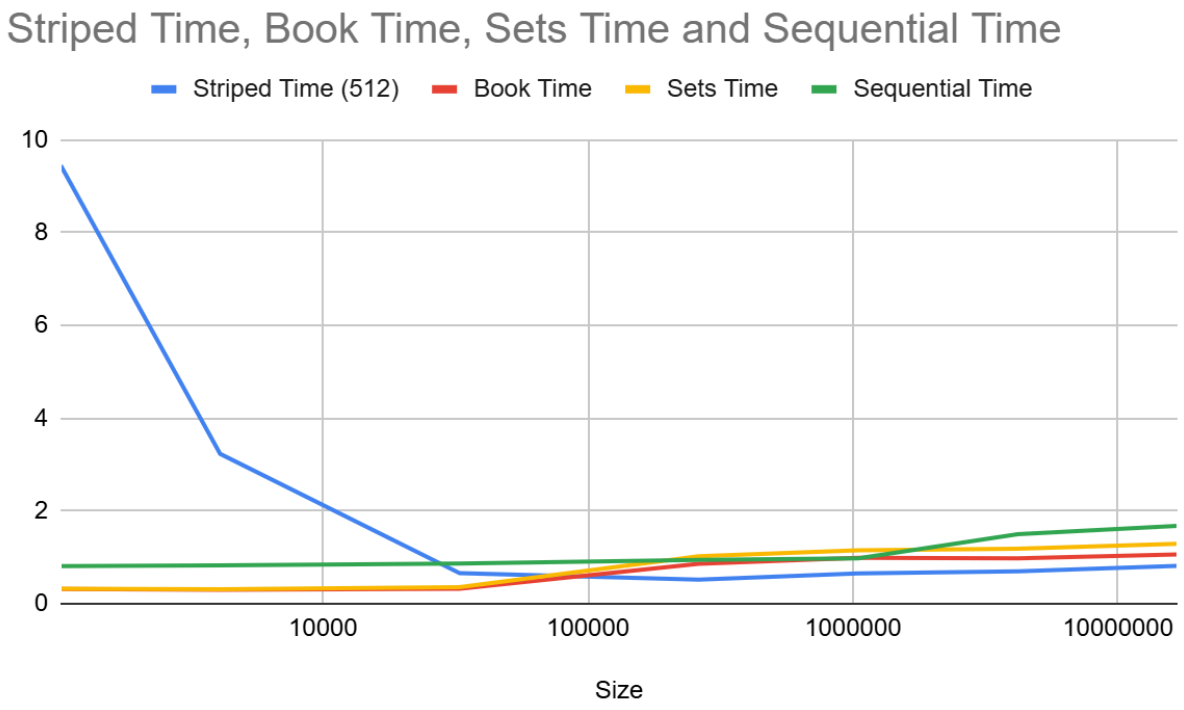
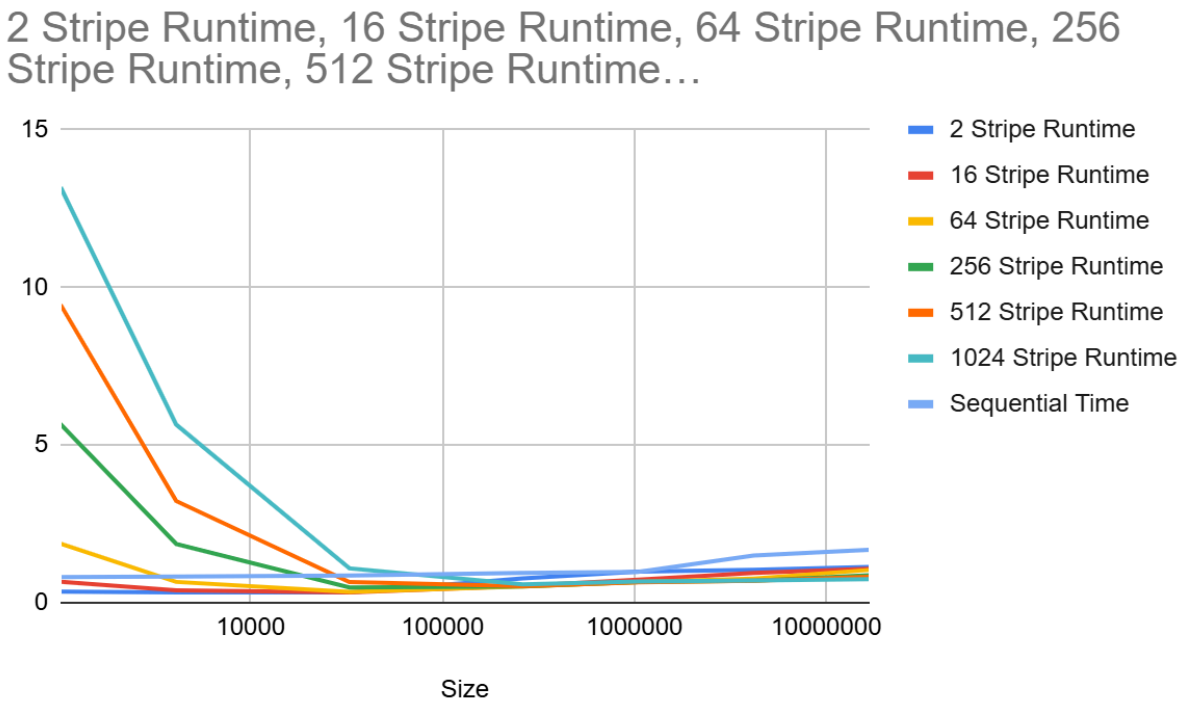
HW3 Concurrent

I ended up implementing a few different versions of concurrent Cuckoo Hashing, including one with pretty much exactly what the book had, one with that but using unordered sets instead of lists for buckets, one with additional lock striping instead of fine grain locking, and one with just pure coarse grain locking (coarse grain locking has been left out of the charts because it took so insanely long to run I didn't want to wait for my program to finish every time).

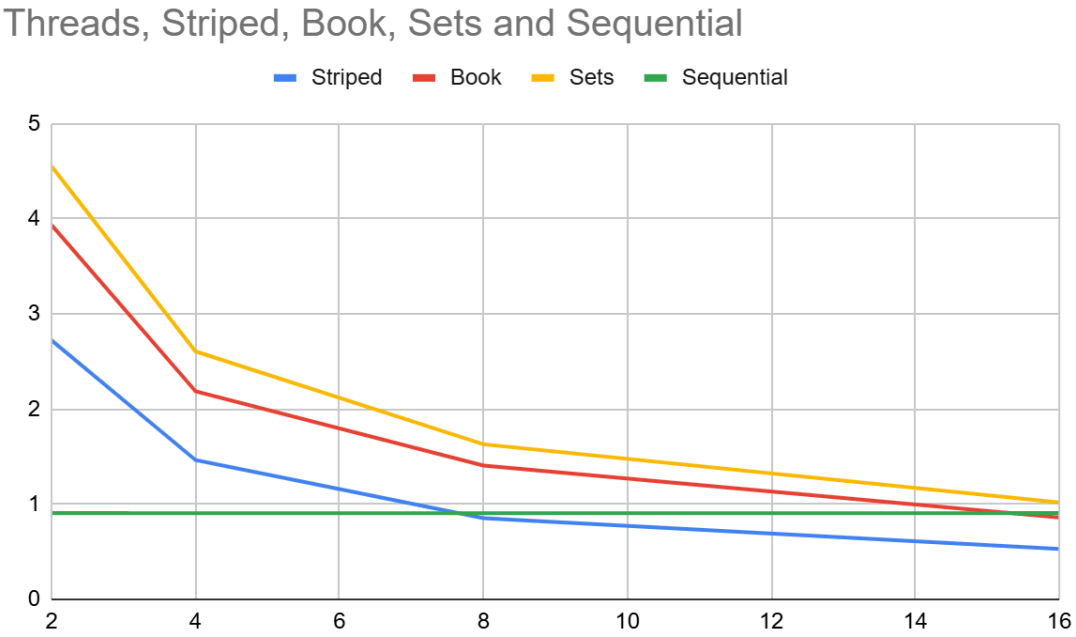
As you can see in the following chart, with a large enough size, all three ended up outperforming the sequential version, with striped locking (512 locks total) performing the best. However, those that did not use lock striping performed better for smaller sizes since there was less contention for the locks.



This chart further demonstrates that same idea, that a larger size is needed for lock striping to become efficient. At the lower sizes, the hash programs with fewer locks performed the best, but as the sizes increased the ones with more locks performed better.



We can also see how the number of threads impacts this performance. The concurrent versions really only work better with a high enough number of threads (at least 8, really 16 for any sizeable benefit).



Finally, we can see how changing the original population of the hashsets affects the runtime. We can see that it really does not affect them greatly, which I suspect has to do with how we are generating the random numbers to be inserted and removed. Since they are all within 0-size, when we add in size numbers that really just guarantees that adds will fail and removes will succeed at the start, not that there will be more collisions.