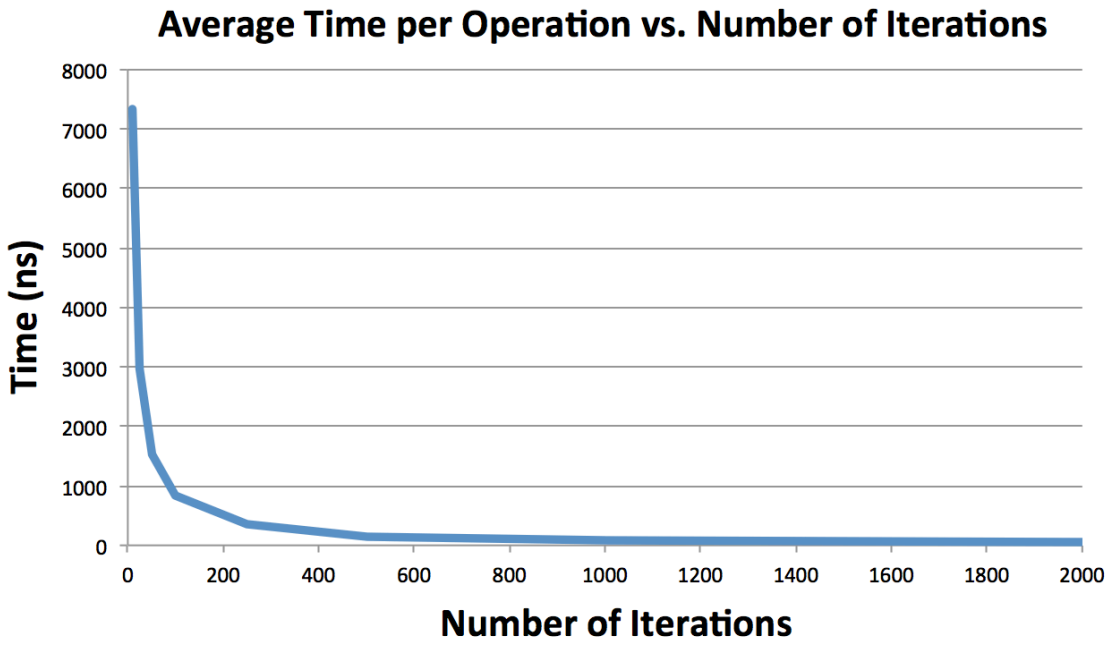
**Lab 4**
**Graphing Results**
**CS 111, Winter 2016**

**Eric Chan**
**504447283**
**echan996@gmail.com**

**Elise Yuen**
**604418732**
**ejyuen@att.net**

**Graph for 1.2**

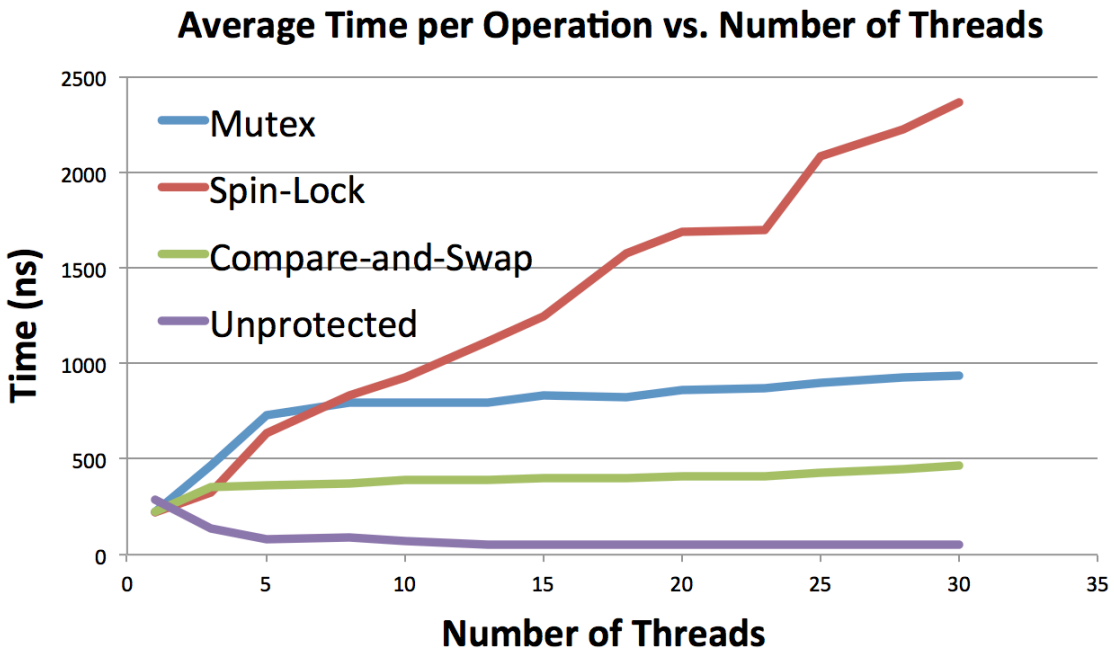
Graph the average cost per operation (non-yield) as a function of the number of iterations, with a single thread.




These results show that the overhead of creating a single thread is spread out amongst a larger number of iterations, and thus, more operations. As we increase the number of iterations, this adds very little time to the overall run time, so the main component of the time is spent setting up the single thread. We can increase the number of iterations and the cost will decrease exponentially. However, after a large number of iterations, we begin to approach a limit that represents the "true" cost of each operation.

**Graph for 1.3**

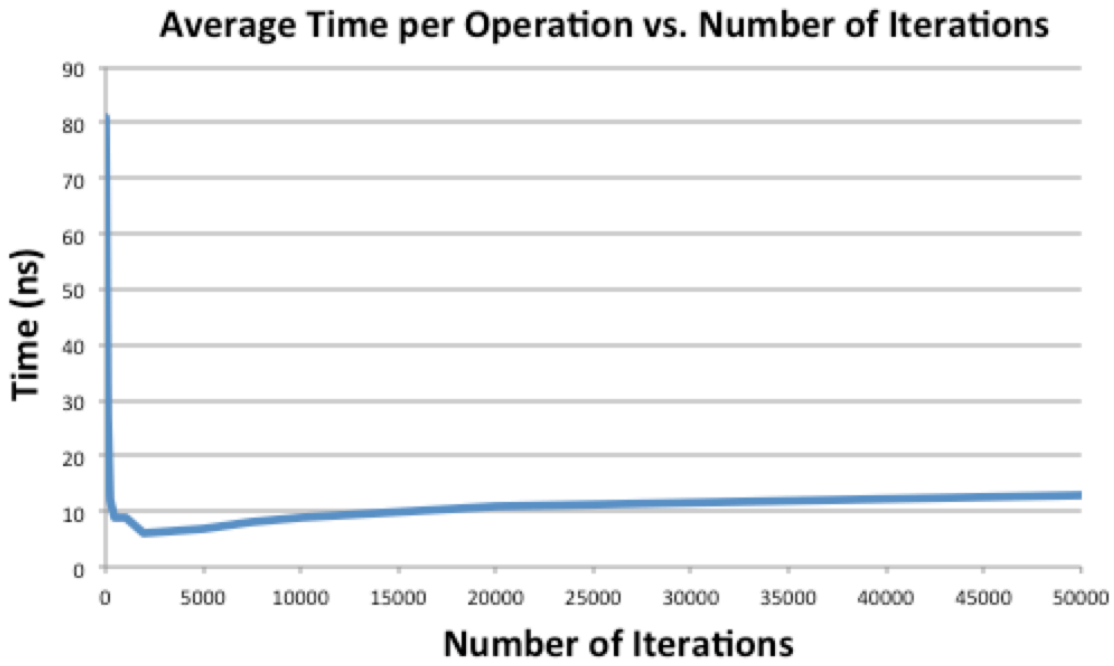
Graph the performance of all four versions (unprotected, mutex, spin-lock, compare-and-swap) vs. the number of threads. We used 50,000 iterations.




In all of the situations except for the unprotected case, we see an increasing trend. The unprotected decreasing trend is expected because this is similar to what we see in the graph for 1.2 since that is technically run on the unprotected case. For the other cases, we see an increasing function because they are all relying on some sort of shared variable, in particular, a lock. A lot of time is spent waiting to use this lock, so we see greater operation times as we add more threads because more threads are demanding this lock, so the queue is longer.

**Graph for 2.1**

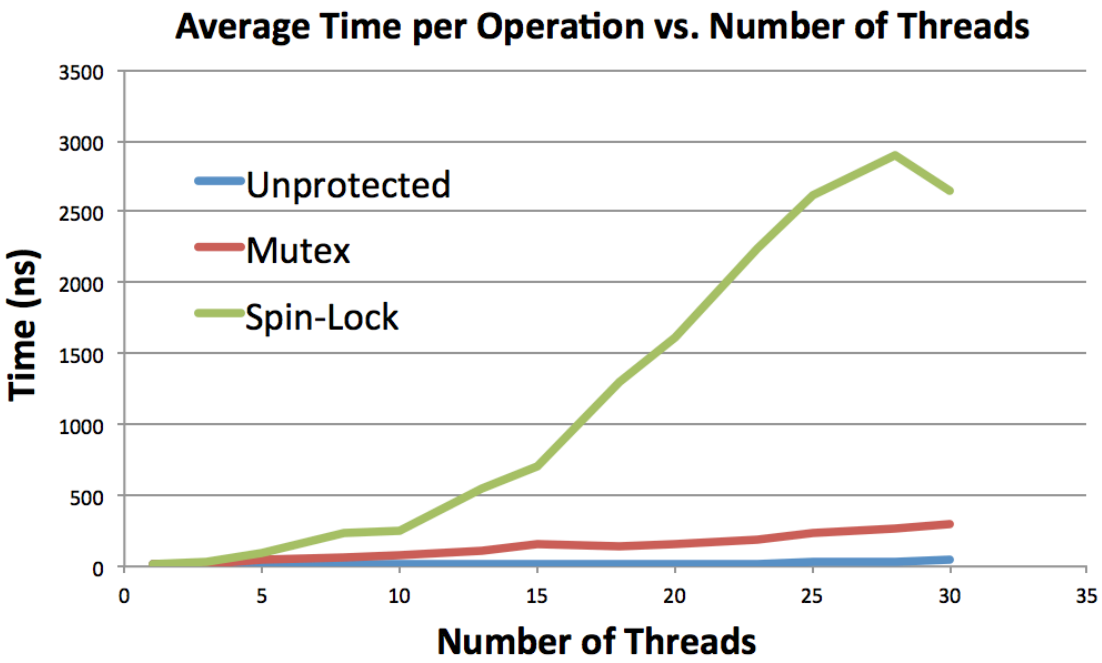
Graph the time per operation vs. the number of iterations (for a `--threads=1`).




These results are discussed in the answers regarding 2.2 in the answers.txt file. However, as a general overview, we can see a trend where the time rapidly decreases, but eventually increases once again. The decrease is due to similar reasons from the previous two graphs. On the other hand, the increase occurs because we have more iterations placing more elements into the linked list. This makes each list longer to traverse, so the time slowly, but surely, begins to increase, reflecting the time it takes to get through more data across various pointers.

**Graph for 2.2**

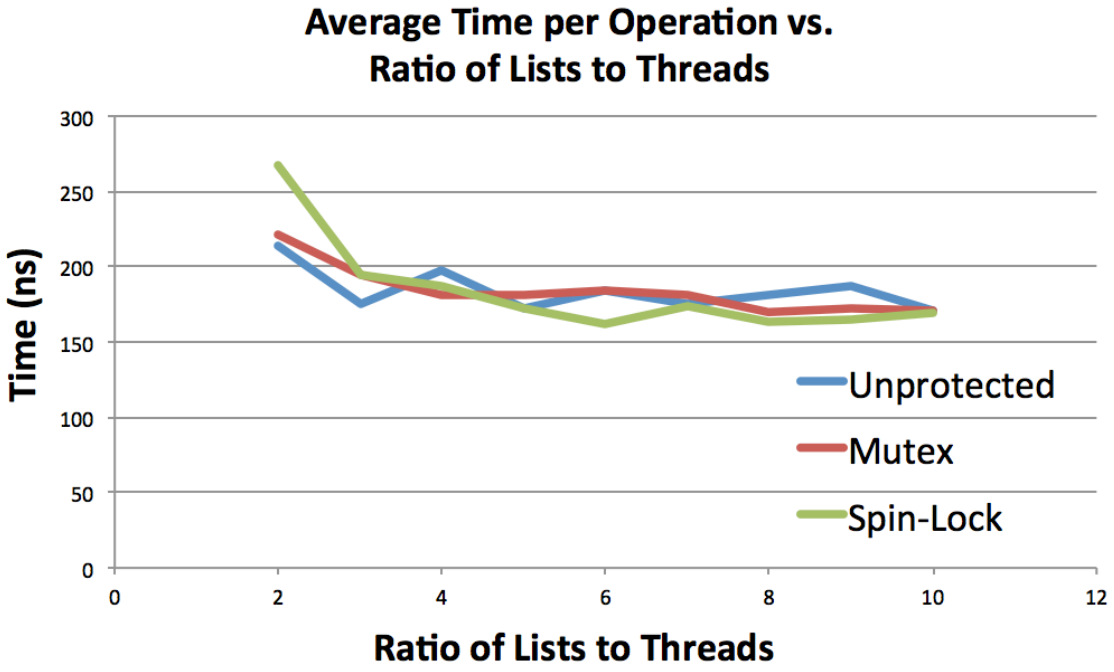
Graph the (corrected) per operation times (for each of the three synchronization options: unprotected, mutex, spin) vs. the number of threads.




In all of the situations, we see an increasing trend. This is because all the threads are relying on sharing a lock. A lot of time is spent waiting to use this lock, so we see greater operation times as we add more threads because more threads are demanding this lock, so the queue is longer.

**Graph for 2.3**

Graph the per operation times (for each of the three synchronization options) vs the ratio of threads to lists.




Unlike the previous graph, we see a decreasing function here. This is because as we split the threads into a greater number of lists, there are less threads waiting to share a single lock and there are more locks for the threads to use. This means that the threads are spending less time (like in previous cases) to attain the lock and can spend more time doing useful work, resulting in lower operation time.