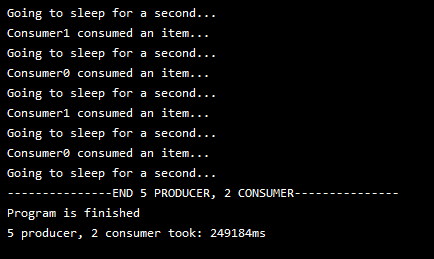
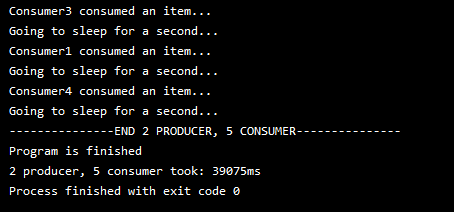
Bryce Callender

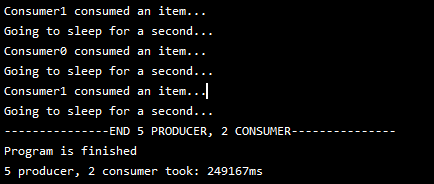
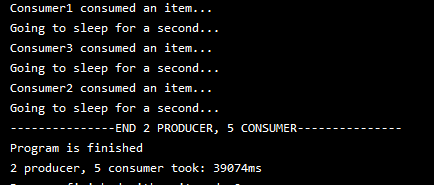
Producer/Consumer (everything but the actors that is included below): <https://github.com/BryceCallender/CS-3700/tree/master/HW5/src>

Actors: <https://github.com/BryceCallender/CS-3700/tree/master/ActorsHw5/src/main/java>

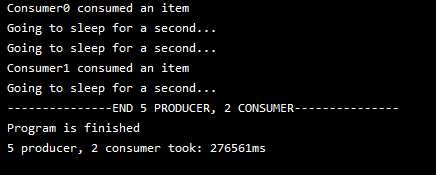
Locks:

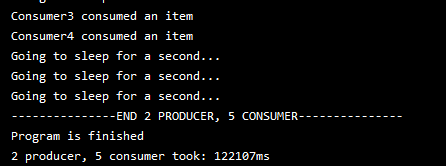
1. 
2. 

Isolated Sections:

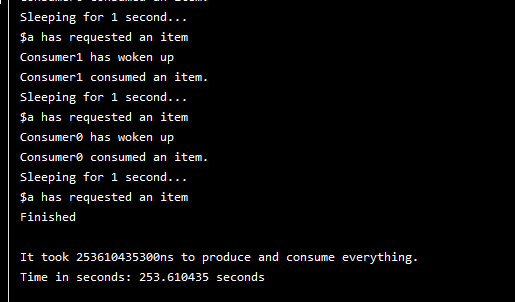
1. 
2. 

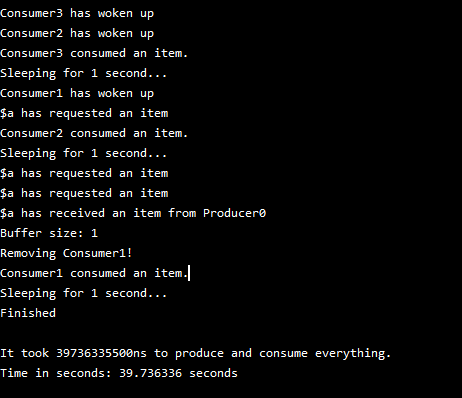
Atomics:

A) 

B) 

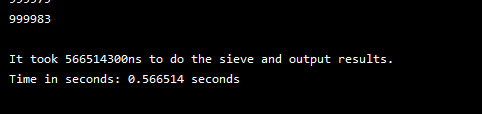
Actors:

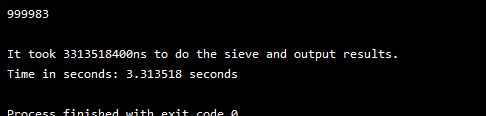
A) 

B) 

1. As we can see here the implementations of locks and isolated sections are almost identical in nature except the two outliers are Atomics and Actors. Isolated sections and locks should be close however since locks are the most low-level, they should be the faster of the two since isolated sections abstracts that. The time for Atomics theoretically should have been the fastest, however my implementation made them take turns and didn’t quite take advantage of atomics as it should so there was a bottleneck with the consumers having to sleep if they accessed something they shouldn’t have. Actors was tough to program and required the least amount of thought. It was slow, however that was due to the fact of all the messages piling into the buffer actor.

2. Sieve

Single Thread: 

Using Actors: 

c) I got no speedup for my implementation because of the nature of having to create threads and the messaging system for the actor. The single thread was able to just execute the numbers really fast while the other was having to create and send/receive messages. Single threaded algorithm was also very optimized so it could be hard to beat out as well.