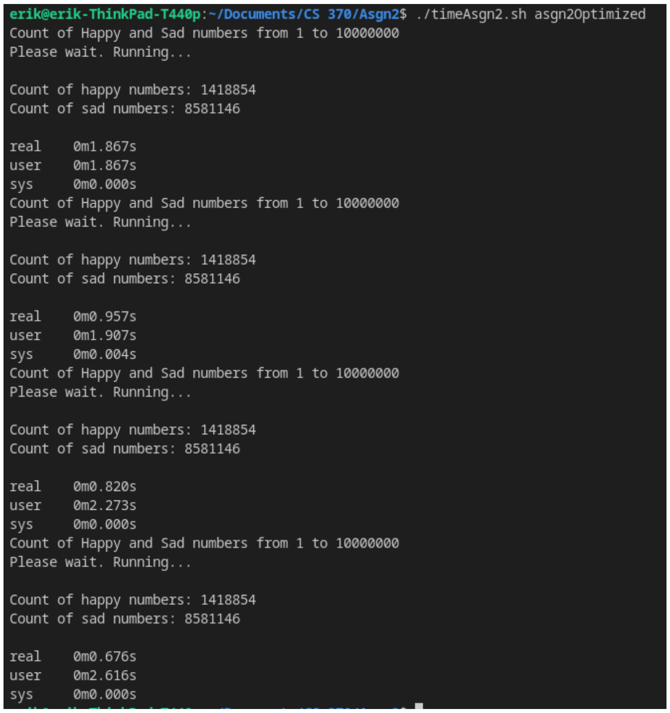
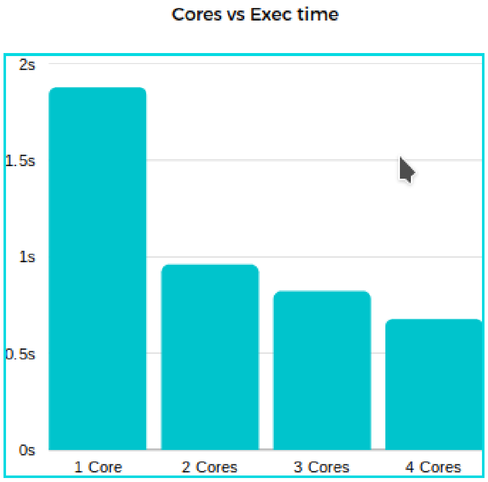
Execution times (real time) for processing 1 - 10,000,000 on
1 core: 1.867s
2 cores: 0.957s
3 cores: 0.820s
4 cores: 0.676s

Speedup for:
2 cores: 1.95x
3 cores: 2.27x
4 cores: 2.76x



```
erik@erik-ThinkPad-T440p:~/Documents/CS 370/Asgn2$ ./timeAsgn2.sh asgn2Optimized
Count of Happy and Sad numbers from 1 to 10000000
Please wait. Running...

Count of happy numbers: 1418854
Count of sad numbers: 8581146

real    0m1.867s
user    0m1.867s
sys     0m0.000s
Count of Happy and Sad numbers from 1 to 10000000
Please wait. Running...

Count of happy numbers: 1418854
Count of sad numbers: 8581146

real    0m0.957s
user    0m1.907s
sys     0m0.004s
Count of Happy and Sad numbers from 1 to 10000000
Please wait. Running...

Count of happy numbers: 1418854
Count of sad numbers: 8581146

real    0m0.820s
user    0m2.273s
sys     0m0.000s
Count of Happy and Sad numbers from 1 to 10000000
Please wait. Running...

Count of happy numbers: 1418854
Count of sad numbers: 8581146

real    0m0.676s
user    0m2.616s
sys     0m0.000s
```



The speedup provided from using more threads provides diminishing returns, as is expected. Using more threads consistently makes the program faster, but I doubt that 5+ would provide any noticeable results (on my laptop, perhaps on a better machine it would). When removing the locks my program actually gets the correct result. I changed it to call the lock less and instead have each thread store local counters, and add the proper amount to the global counter after they finish executing. Because the threads finish at different times it is almost, but not actually, guaranteed that it will correctly find the happy number total.