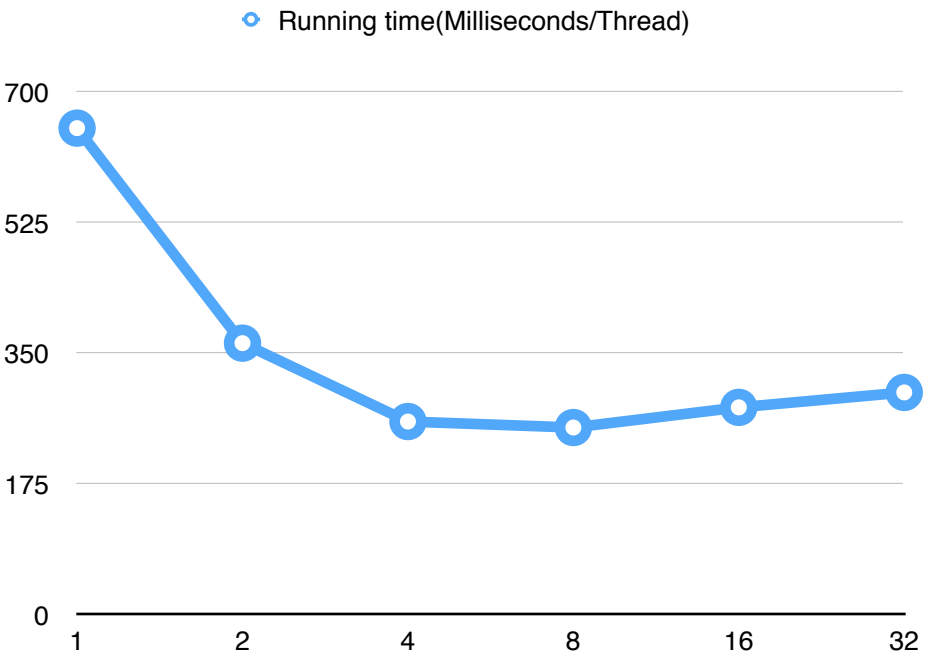
Comp 409 - Assignment 1

Main class is Graham.java

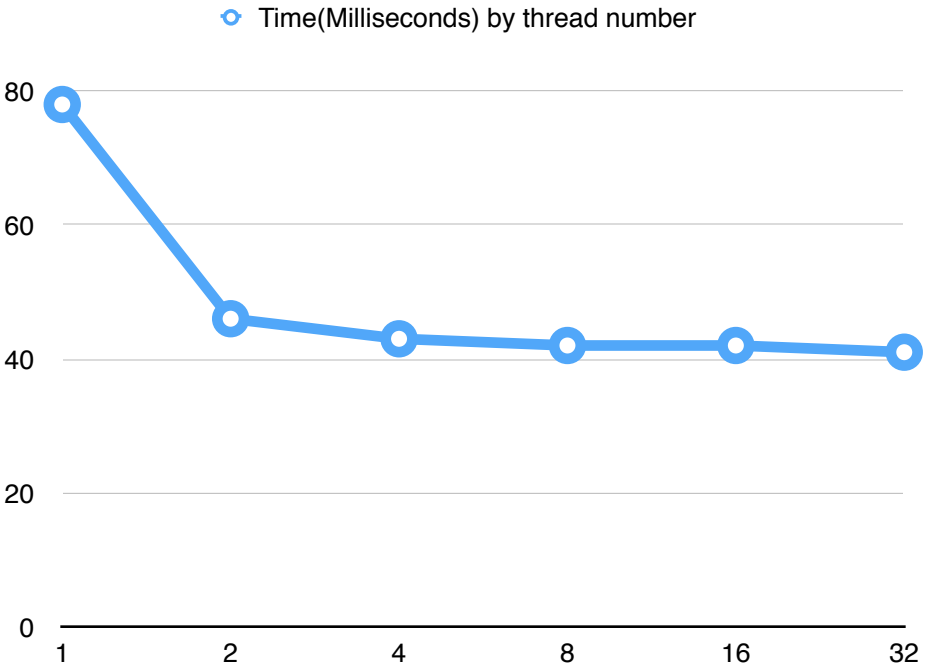
Question 1.a



Threads	Time(Milliseconds)
1	652
2	364
4	259
8	251
16	278
32	298

We can see that the running time decrease when using 2, 4, 8 but increase slightly after as the cost of handling multiple thread overcome the speed gain. As my computer has 2 cores and 4 virtual this correspond to the values.

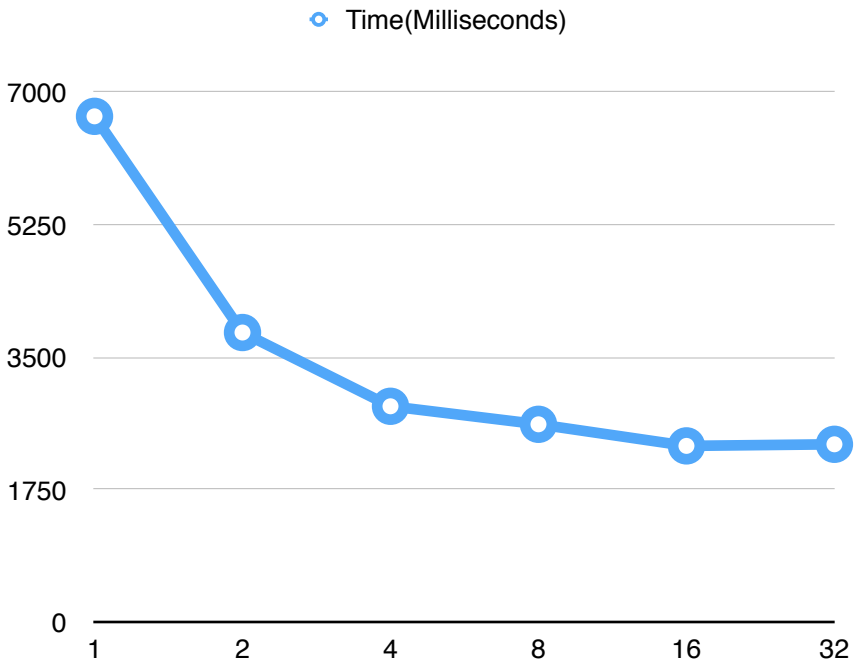
Question 1.b



Threads	Time(Milliseconds)
1	78
2	46
4	43
8	42
16	42
32	41

Similarly we notice a significant improvement with 2 thread but afterward the improvement is very small.

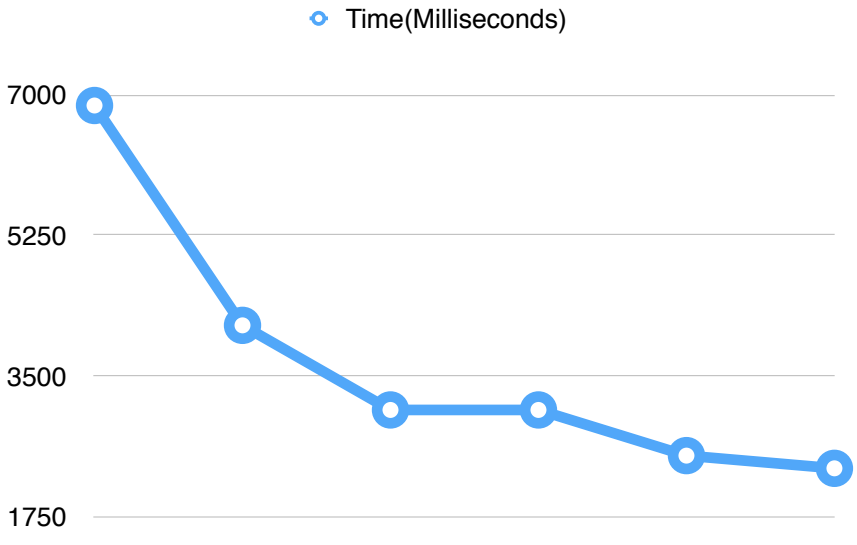
Question 1.c



Threads	Time(Milliseconds)
1	6683
2	3828
4	2854
8	2616
16	2331
32	2352

Here again similar improvements

Question 1.d



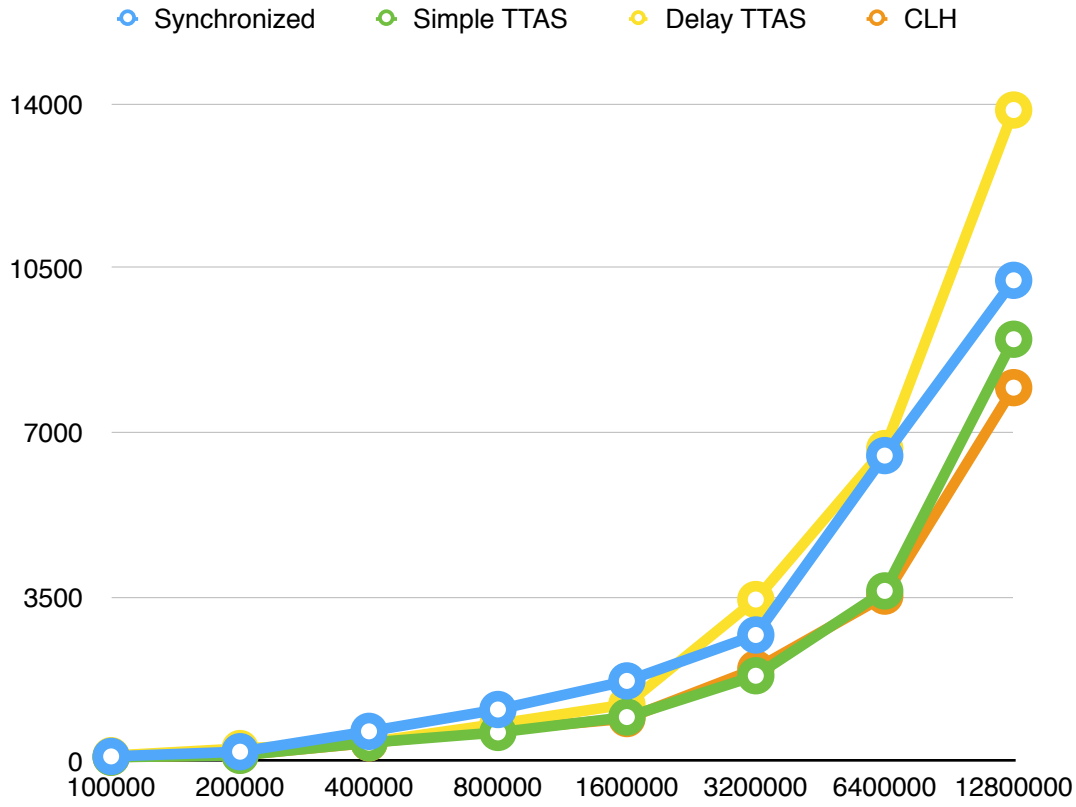
Threads	Time(Milliseconds)
1	6862
2	4128
4	3076
8	3076
16	2508
32	2350

Question 2

In this question I used 2 thread as I have a 2-core processor.

Time

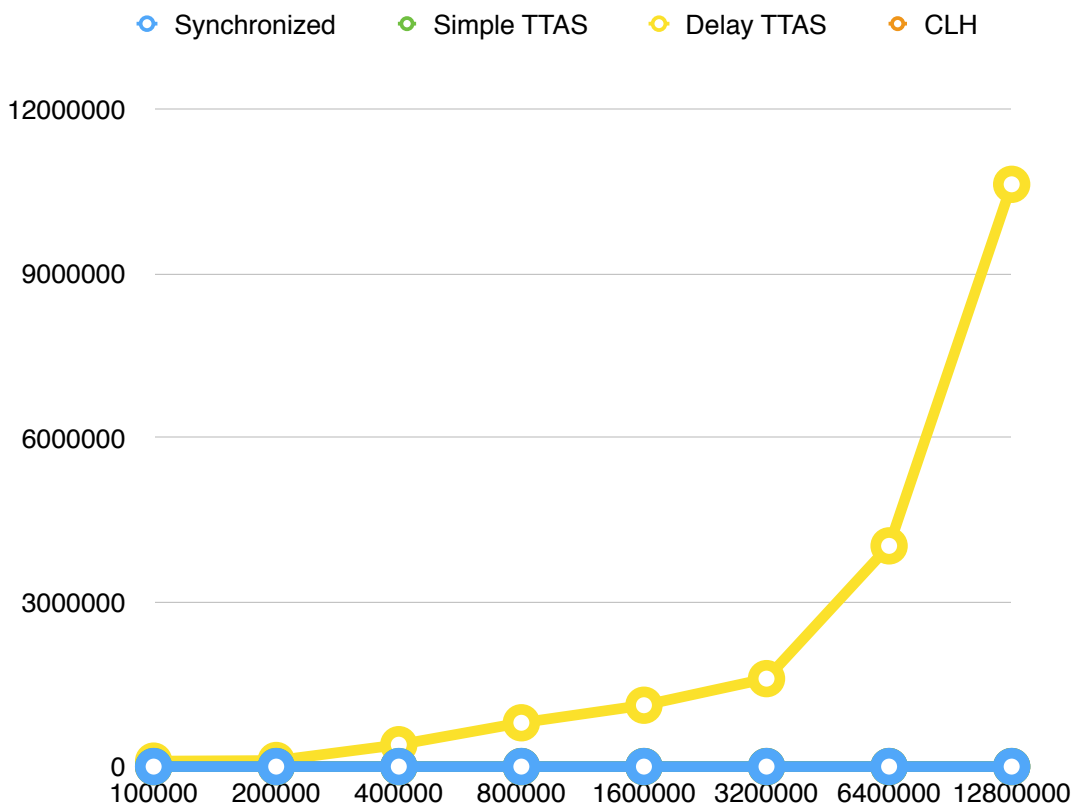
n	Synchronized	Simple TTAS	Delay TTAS	CLH
100000	98	81	114	84
200000	198	121	265	127
400000	632	391	400	386
800000	1097	609	790	658
1600000	1706	936	1604	904
3200000	2684	1816	3439	1964
6400000	6505	3623	6656	3512
12800000	10238	8983	13865	7952



Here we can see that all method have a similar timing. The exponential back off is taking the lead by the end which seems normal as there can be some times when a thread is sleeping while the lock is open so we are wasting time.

Delay

n	Synchronized	Simple TTAS	Delay TTAS	CLH
100000	72	80	99853	31
200000	98	76	110959	26
400000	111	58	399843	58
800000	201	62	799960	71
1600000	109	87	1120350	40
3200000	199	101	1605834	88
6400000	349	104	4029542	96
12800000	113	100	10629922	129



Now for the delay all method but the exponential back off have a relatively small back off. This seems totally normal as when the thread is asked to sleep other thread can alternate the request of lock which can be a very high number and when it wake up it's not guaranteed to be unlocked. We also notice that the CLH is the one with the smallest values which is predictable as we are queueing the execution of a thread so there is has less chance of missing its turn.