1. *How long did the project take? Which parts were most difficult? How could the project be better?*

The project took me about 20 hours. The most difficult part is to implement the commit of the TxInfo and the transactional hash set. I think the project could be better if we spend more

1. *What extra credit projects did you implement? What was interesting or difficult about them? Describe how you implemented them.*
2. *You implemented three versions of the hash set: coarse-grained locked, fine-grained locked, and transactional. Compare the perfomance of the three implementations as your vary the number of threads in the system. Plot your results both for low-capacity hash sets (less than 20 slots) as well as for high-capacity hash sets (more than 1000 slots). Discuss.*

I used two slot capacities 10 and 1024. For each experiment, I kept the total amount of the work constant and only vary the number of threads. For each hashset, I will do 131072 insertions and contains and time them and obtained the following result.

Observations:

* When the number of threads is equal to 1, the performances of the 3 sets are almost identical.
* The performance of the coarse-grained hash set is always much slower than the other two with increased number of threads. It does not vary much with number of the threads because no two threads can access the set at the same time so it essentially has the same performance as a sequential program.
* The fine-grained hash set is usually faster than transactional hash set because on top of the locks per array index, the transactional hash has to do commit to ensure the blocks are executed atomically.
* The performance of fine-grained hash set and transactional hash set increased dramatically when the number of the threads exceeded 4. For small capacity hash set, the performance for the two hash set continue to increase with large number of threads where as for large capacity hash set, the performance stay almost constant.
* The performance is generally faster with a large capacity hash set because of less collision.

1. *Plot the performance of your transactional hash table while varying the length of the exponential backoff. Include a version in which the exponential backoff is disabled. For this experiment, you should use more threads than you have processors; this makes the time wasted retrying doomed transactions more obvious (since there might be other threads waiting to use the processor). Discuss.*

In order to better understand the data, I also included the number of aborted transactions (using the code from question 5) for each run so that we know if the which performance changes are due to backoff time change.

Observations:

* At higher initial backoff time, there will always be runs that has zero aborted transactions, they are the points that stays on the x-axis. Such points will tend to be more at higher initial backoff time.
* For runs that do have some aborted transactions, the optimal initial backoff seems to be 100 ms to 1000ms. If the backoff time is too small, there are too many aborted transactions. If the backoff time is too large, the threads are waiting too long, resulting in very long running time.
* With larger capacity the performance is still better because of less collisions.

1. Modify your code so that it prints the number of transactions committed and aborted for each thread. How does varying the initial length of the exponential backoff change these statistics? Include a version in which the exponential backoff is disabled. Again, use more threads than processors. Discuss.

I experimented with 5 threads and 2000 insertions or contains method calls and only vary the backoff time and count the number of the aborted transactions. Since the total number of successful commit does not vary, so it would be sufficient to just examine the ratio of aborted transactions to the successful transactions. The following data was obtained:

Observations:

* At small initial backoff time, the number of aborted transactions is very large and decreased dramatically by just increase the initial backoff time by 2 to 6 milliseconds.
* At higher initial backoff time, there are more frequent zero failed transactions runs.