Describe your implementation including any design decisions you made. Make sure to emphasize anything that was difficult or unexpected.

For roll back, I implemented the forward rollback. Since we have the tidToFirstLogRecord map, we could take advantage of it and simplify the rollback process. First, I used the tidToFirstLogRecord to find the startOffset of the aborted transaction. Then I could raf seek to the beginning offset and start reading from there. For the roll backs, the goal is to revert the page to the origin image before any updates. Thus, from the beginning record, I will read forward to find the first update record. Then the before image of the page stored with the first update record in the log file is the image I should revert back to. To deal with the case that the aborted transaction is changing multiple pages, I will continue reading the log till the end (eof error throwed by raf) and store the rolled back pages into the alreadyRolled Set to get the pages' before image from their first update record. By doing a forward roll back like this, I could reduce the cost by skipping the middle steps and directly revert to the earliest page image. I have also written a CLR record each time I have rollback a page. The CLR record's structure is similar to update record structure that contains record type, record id, dirty version of page, page after undone, and the start offset. In this way we could make the recover easier when redoing the CLR.

For recover, first, to find the checkpoint offset, I read the first record and the checkpoint offset is the first long. If checkpoint is specified I will first read the transaction id and start offset of the transaction from the checkpoint record and add them to the activeOffset map which records active transaction and their first offset. Then I will start read from the checkpoint. If the checkpoint is not specified I will start read from beginning. Because LogFile.logCheckpoint() have already flush all dirty page to disk so we are safe to start redo from checkpoint. Then I will read the log file till the end. While reading, I will add transactions and their start offsets to activeOffset for begin record and remove transaction from activeOffset for commit and abort records. For updates and CLR recorsd, I will redo the operation by set the page on disk to the page's before image stored in the update record and CLR record. For undo, I will start reading forward from the earlier record between the earliestActiveOffset and checkpoint offset. In this way, I could be sure to not miss any important log since the current active transaction opens.

The most difficult part for me is to get the offset right, however, debugging with the LogFile's print() method helps me solve the question a lot.

Discuss and justify any changes you made outside of LogFile.java.

I didn't made changes to other codes except adding the proposed line in the BufferPool.java. When flushing page, we are adding the logWrite and logForce for any transaction that creates dirty pages, thus we have make sure that the update log is on disk before page is push to disk. I had a transactionPageMap that contains the active transactions. Thus I only need to check this map to see if the transaction is commited. If transaction is active it means it is not commited and thus I could write the update log, if it is not active (committed) then the commit log is already

written. My evictPage() calls flushPage(). In flushPage(), I move the call of writePage outside of the limit that check the dirtyPage to allow STEAL. Thus it could write any page to disk including dirty pages. In transactionComplete() I also comment out the flushPage() to allow NO FORCE. I have also add the call to logWrite() to write the log and set the before image of the page when transaction complete.

Describe a unit test that could be added to improve the set of unit tests that we provided for this lab.

We don't have test for one transaction that process on multiple pages and abort. We should add them since recover a lot of pages is an necessary behavior to test. Also, we only test for adding a little bit data and then test recovery behavior. Thus, we should have test for adding more data to test efficiency of recovery.

Other declaration
I don't have feedback.