Assignment 3

Stuart Mashaal
260639962

Oliver Tse Sakkwun
260604362

Due: December 4, 2018

Question 1

1.1

1.2

1.3

1.4

1.5

Question 2

2.1

See in code

2.2

The contains method scans the list to find the pair of nodes (pred,curr) reachable from head such that pred.next == curr,pred.key ¡ key and curr.key ¿= key.The traversal uses hand-over-hand locking.

- Item is not in the list
  When curr.key == key is false that is curr.key ¿ key. From the sortedness invariant of the list, pred.next == curr, and pred.key ¡ key we conclude that item cannot be in the set.
- Item is in the list
  When curr.key == key is true, then from the uniqueness of keys that curr.item = item. Hence, item is in the set.

Question 3

3.1

Working on that

3.2

The difficulty is to define the criteria where the indexes are valid to be read or written. We had to add two more atomic variables: the first representing how many elements can still be inserted; the second represents the valid indices for reading.

Question 4

4.3

Given the massive overhead of all the state needed for all the **tasks** that were run on the threads, the parallel algorithm actually ran slower given the non-infinite number of processors available ($\lceil 5 \cdot log_2(2000) \rceil = 55$ threads on 8 virtual processors to be exact).

Specifically, the parallel algorithm ran in 10ms while the sequential ran in 2ms. Not very good speedup, to say the least.

However, the theoretical speedup is linear so long as the critical path is not the bottleneck.

4.4

First, the implementation breaks up the matrix-by-vector multiplication into its various vector-vector dot products. It does this by breaking the matrix into two halves, then halving again and again until it reaches a half that consists of only one row. At this point, it initiates a dot product on that row.

Note that the number of *work nodes* of this step is therefore $\Theta(2N)$, the number of nodes in a fully-balanced binary tree containing N leaves.

Each "leaf" of this binary tree is a dot product to be computed. The dot product is broken up into the sum of two *sub-dot-products*, halving all the way down as before. At the bottom, there is a single two-integer multiplication to do. Once again, this is $\Theta(2N)$ work nodes.

Because $2N \cdot 2N \in \Theta(N^2)$, the implementation achieves work $\Theta(N^2)$.

Also note that the longest possible path through this graph is all the way to bottom of one tree, then another, then back up both of them. That's $\Theta(4 \cdot log_2(N)) \in \Theta(log_2(N))$, so the implementation meets both the *work* and *critical path* requirements.

Since the critical path is $\Theta(log_2(N))$ and the *work* is $\Theta(N^2)$, the parallelism is therefore $\Theta(\frac{N^2}{log_2(N)})$, which is notably much closer to $\Theta(N^2)$ than to $\Theta(N)$. In other words, the parallelism is proportional to N^2 which is also what the work is proportional to, so with ∞ processors this highly-parallel algorithm would run much faster.