OS HW 3 Dry part

Q1-1

Search, get a lock on first element, get lock on next, release first than lock 2nd next and so on and so on. (this way aquire lock in sequence no deadlock)

Insert, search place to insert, lock the prev and next of new place, create new node and lock it, put it in between prev and next, then release prev.

Remove, search curr aquire prev next and toRemove unlink to Remove and delete it.

Q1-2

Obs1, if T2 is handling p2 and T1 handling p1, than T2 must have already tried to handle p1 and than upon reaching p1^2, saw that someone else was handling p1. Meaning T1 is ahead of T2 in the least (cause it got to p1^2 first), meaning current element T2 is locking is smaller than curr element T1 is locking.

Obs2 if T2 removed p^2 than its candidate must divide p^2, since the only natural number dividing p^2 is p, T2 must have chosen p as its candidate.

Obs3, if p is a prime candidate of T, due to the explanation of obs1, T must have tried to handle all numbers between 1 and p and either succeeded or realized some other T' is handling them.  
for each q in [2,p) in the list, if some T' already removed it than q does not divide p. if someone is working on q as a candidate than by Obs1 it already passed p and thus did not remove it meaning q does not divide p.

Q1-3

If threads don’t check for previous removal of p^2 once they selected p as a candidate, then they will continue to look for divisors of p until the end of the list. Thus, the first string to choose to as a candidate will also be the first one to reach the end of the list, and will also so be the first one to choose 3 as a candidate (since other threads cant pass him in the list) and then by induction this thread will be the first in selecting a candidate for every single candidate and so he is the only one who will remove elements. This need not be the same thread in every run of the algorithm. Any thread has a chance to get to 2 first, but once a certain thread did, it will be the only one to remove elements.

Q2-1

A different sync method that allows the given case:  
All threads use hand over hand read/write locking. Using read locks to search and write locks to remove from the list. When a thread sees an element locked, it assumes that another thred will handle this candidate. In this manner we can have T1 choosing p2 as a candidate, while T1 is reading it, T2 reads it and sees it is already taken, thus continue to 3 and chooses it as a candidate.  
  
In this scenario T2 is ahead of T1 but p2>p1.

Q2-2

Lets expand the different scenario, T1 chose 2 and is reading it T2 chose 3 and is reading it, along comes a new T3, reads 2 and sees that its taken, reads 3 and sees that its taken, and so reads 4, sees it isn't chosen and picks it as a prime candidate.  
since 4 is not prime, we see that a prime candidate is not necessarily prime under this synch method.

Q3-1

If 2 threads wanting to change the same data will try to upgrade to write simultaneously, they might both check to see no other writer exist and then change themselves to writers and that way you would have to simultaneous writers, resulting in undefined behavior.