Homework 4
Ryan Margono

1.
a) since the prime factor program is not I/O bound, there will not be threads that are waiting while another operation finishes. Therefore, it is optimal to use the same number of threads as there are available cores, so that each thread can execute independently. Since there are four cores but two are being used for a game, then at minimum there should be two used for the prime program and at most four (depending if the game program has a lot of I/O so the prime program can run during that time). The reasoning is that anything less and there would be wasted cores and anything more and the threads wouldn't be working independently and context switching would lead to worse performance.
b) as i alluded to above, knowing if a program is mostly I/O or CPU bound can help determine how many threads to use for a program.

2.
The ABA problem occurs in lock-free data structures where a space is memory is intially freed then resued. When this point in memory is used in CAS, it can succeed even in cases where it is 'supposed' to fail. In order to fix this problem, freed pointers should tracked in a list with their counter. When a previously freed address is going to be resued, its counter is incremented so there is no repeats.

3.
a) have one mutex as a private variable and lock it at the start of each operation and unlock it at it.
b) yes, performance can be increased using reader writer locks. Writer locks in the push() and pop(), and a reader lock in the size(). This will make it so multiple threads can read the size() rather than one at a time while still being thread-safe.
c) it would be possible to have improvements in performance using CAS if the size() function was not a necessity. If CAS was correctly implemented in push() and pop(), then yes, the nodes in the queue would be in the right order. However, size() wouldn't work because was maintained by some counter in the datastructure, there is no gurantee that it will be accurate because a node can be added and the counter wouldn't increment it yet or vice versa.

4.
a) There is an atomicity violation. Thread 2's code could execute in between Thread 1's. The programmer assumes this won't happen but that is not correct.
b)
Mutex Solution:

lock these two sections of code with a mutex so only one can execute at a time, making them atomic.

‘Cache’ Solution:

save thr_glob -> proc_info in a local variable

if (local variable is not null) fputs(local variable)