**Statement and proof of correctness**

Part 1:

The program will not deadlock or starve during execution, however the method of acquisition for locks is incorrect. It uses getAndSet() operations instead of checkAndSet(), which does not guarantee correctness.

Part 2:

The program will not deadlock after some period of execution, but will starve. This will happen because when one chopstick is raised and the second chopstick is found to already be in used, the program will not reset the first chopstick. This will cause the first chopstick to be raised without any thread claiming ownership or responsibility for lowering it, which will cause some people to be unable to eat.

Part 3:

The program will not deadlock or starve. The program will wait until it looks as if it can acquire both chopsticks. When it can, it will acquire one chopstick, then the other. If one or both chopsticks were acquired by a different thread in the interim, then the thread will set back down any chopsticks that it owns for others to use and attempt acquisition once again. This system ensures that no chopsticks are ever up without a thread claiming responsibility.

Part 4:

This program works with any number of threads greater than one. For one or less threads, there are not enough chopsticks available for even one philosopher to eat assuming each one brings one chopstick to the table.

**Efficiency**

The final product of this program uses a TTAS system to wait until it has a chance to acquire chopsticks. Most of the program’s execution time is spent spinning in the loop attempting to acquire both chopsticks, which has a fair runtime but lacks scalability.

**Experimental evaluation**

The method of evaluation that I used was simply letting the program run for 30 seconds using between 2 and 10 threads. If the program had frozen, there was clearly a deadlock and changes to the program had to be made. After the time had elapsed, I quit the program and determined from recent output if all the threads were getting a chance to eat. If not, then there was a starvation scenario and the program would again have to be changed.