1. Drew Sadler
2. pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;///
3.
4. I think since it will be synchronized it will always end up as 0 since it would be alternating between the 2 functions, probably taking 1.5 to 2 times as long to execute, since it can't repeatedly execute 1 function and must take the time to switch.

5.
   ```
   asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   ```
   Yes it meets my expectations, because we locked the tread before modifying the variable, so I wouldn't assume that the value has changed at all

6.
   ```
   [asadler1@hopper3 Studio_10]$ ./race
   final value is: 4971380
   Time elapsed is 1646101157 seconds
   [asadler1@hopper3 Studio_10]$ ./race
   final value is: -9610255
   Time elapsed is 1646101162 seconds
   [asadler1@hopper3 Studio_10]$ ./race
   final value is: 18856765
   Time elapsed is 1646101164 seconds
   ```

   Average time: 1646101161 seconds

7.
   ```
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   Time elapsed is 1646103287 seconds
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   Time elapsed is 1646103292 seconds
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   Time elapsed is 1646103297 seconds
   ```

   Average Time: 1646103292 seconds
   Took longer by: 2131 seconds

8.
   ```
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   Time elapsed is 1646102204 seconds
   [asadler1@hopper3 Studio_11]$ ./mutex
   final value is: 0
   ```

```
Time elapsed is 1646102227 seconds
[asadler1@hopper3 Studio_11]$ ./mutex
final value is: 0
Time elapsed is 1646102230 seconds
```

Average time: 1646102220.33333 seconds
Took longer by: 1059.33333 seconds

9. The results would obviously take longer for #7 as each time it's having to take extra seconds as each change in N would mean it would have to go through both unlock and lock. While with #8 can save 1,068.666667 seconds just by calling mutex less and spending less overall to compute faster due to less amount of calls
10. Per-iteration calling would be more efficient when you are making a program that would need to make sure it is valid before modifying, for instance like a bank account where we would want to be certain every single instance and multiple functions like withdrawal, deposit, and bunch of other functions that could modify, but we would want to wait on a lock for each part for security. While the Per-thread locking strategy would be most efficient in account/profile based things, where we can verify through a initial protection of password or something and then can let them edit whatever rather than wait for each single part or function to lock/wait