Part3: Write Up

Coarse lock stack numbers

- Average total operations: 1,643,136
- Average total pops: 658,078
- Average total pushs: 434,218
- Average total peeks: 550,840

Big difference in variance between each operation. Upwards of 800k difference per running

RW lock stack numbers

- Average total operations: 2,017,840
- Average total pops: 14,582
- Average total pushes: 12,308
- Average total peeks: 1,990,949

Upwards of 20k difference per running

Swaptop stack numbers

- Average total operations: 7,436,072
- Average total pops: 35,845
- Average total pushes: 32,306
- Average total peeks: 7,351,378
- Average total swaptops: 16,541

Upwards of 10k difference per running

To provide a deadlock and race condition free solution for swaptop stack, I had to make sure that I locked and unlocked on the correct lines of code. Unintentionally, I first deadlocked my code by returning from a function before unlocking. By doing this, I caused the function to wait forever the next time it entered that function and tried to grab the lock. After fixing this problem I noticed that I was still having race conditions when trying to return `current->data` from

any of the functions even though I locked and unlocked at the correct points. While the pointer `current` was created locally, I did not realize that the memory it pointed to was shared between the threads. To solve this problem, I assigned the data `current` was holding and assigned it to another local variable and returned that instead.

My implementation for the `swaptop` function was simple. I took my `pop` and `push` functions, copied them into new `swap_pop` and `swap_push` functions and erased all mutex uses inside of them. That was there can be no deadlocking or race conditions when calling these functions.