Performance & Reliability

The test run was: `UnsafeMemory <Model> 8 10000 60 50 60 30 10 30`
If more swaps were done, Unsynchronized and GetNSet would lock and not finish due to race conditions. Also, It is much more reliable with the large spacing of numbers as the non-DRF code has issues with 0 as an array input.

Synchronized - `Threads average 17429.3 ns/transition`

Unsynchronized - `Threads average 10702.8 ns/transition`

GetNSet - `Threads average 13315.3 ns/transition`

BetterSafe - `Threads average 13838.1 ns/transition`

BetterSorry - `Threads average 14368.0 ns/transition`

Synchronized, BetterSafe, and BetterSorry were extremely reliable as they completed every test case I threw at them. Unsynchronized and GetNSet however, had an issue where if they ran with two many swaps and/or particular array elements, they would hang. However, these two models ran much faster with the lower reliability being a factor. Unsynchronized returned very unreliable sums off by about 10,000% while GetNSet would return sums that were off by about 30%.

Note: The non-DRF models would sometimes return the following error:
```
java.util.UnknownFormatConversionException: Conversion = 'l'
```

DRF or Non-DRF

Synchronized – DRF – No threads can swap at the same time

Unsynchronized – Non-DRF – No prevention to access of the values leads to many race conditions. Will fail most swaps over 100000 and more likely to fail with lower array elements.
Most likely will fail: `UnsafeMemory Unsynchronized 8 1000000 6 5 6 3 0 3`

GetNSet – Non-DRF – There are delays in the code from getting the value and updating it that lead to race conditions. Will fail most swaps over 100000 and more likely to fail with lower array elements.
Most likely will fail: `UnsafeMemory GetNSet 8 1000000 6 5 6 3 0 3`

BetterSafe – DRF – This model decreases and increases atomically forcing it to be 100% reliable and faster than Synchronized.

BetterSorry – DRF – A lock is used to stop race conditions that still hinder Unsynchronized. This may still be unreliable in the critical sections of the code however. Certain test cases may still hang on BetterSorry such as:

```
java UnsafeMemory BetterSorry 16 1000000 0 0 0 0 0 0
```

This test case is difficult to many models as the zeros are a hard number to handle in this program.

GDI's Favorite Model

The best model for GDI would be GetNSet because it is the perfect balance of speed with slight loss in reliability that they would be able to overlook.