Exercise 5.1

1. We did not notice huge variations but ones we saw are probably given by different things that we discussed at the lecture, such as:
    - Disturbances of external factors for example from other processes running on the computer
    - Cost of creating and starting threads.
    - Limitations in the number of cores, that's the reason why we don't see an increase of performance when reaching a large number of cores.,

    By increasing the number of times we test it, the result becomes more and more robust as we average more measurements.

2.

```
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 17.0.2
# CPU:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD; 8 "cores"
# Date: 2023-10-07T16:25:01+0200
Mark 7 measurements
hashCode()                      5,2 ns       0,44   67108864
Point creation                 54,8 ns       5,80    8388608
Thread's work                13553,8 ns    2255,97      32768
Thread create                 1760,0 ns      91,34     131072
Thread create start         132918,5 ns    5318,22       2048
Thread create start join    310393,5 ns   25754,77       1024
ai value = 716740000
Uncontended lock               24,2 ns       2,54    8388608
```

    The results are reminiscent of those obtained from the previous point, the difference from what was seen in class may be due to several reasons but for the most part to the different hardware used.

Exercise 5.2

1.

```
> Task :app:run
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 17.0.2
# CPU:  Intel64 Family 6 Model 78 Stepping 3, GenuineIntel; 4 "cores"
# Date: 2023-10-07T14:12:15+0200
countSequential              11850277,5 ns  871804,76        32
countParallelN       1        9961092,8 ns  369571,64        32
countParallelN       2        6696917,2 ns  510166,44        64
countParallelN       3        6568350,6 ns  496951,21        64
countParallelN       4        6171883,3 ns  752656,80        64
countParallelN       5        6934658,1 ns  766006,65        64
countParallelN       6        6511293,1 ns  464173,09        64
countParallelN       7        9132124,1 ns 1966764,47        32
countParallelN       8        9167698,8 ns 1695953,20        32
countParallelN       9        6846764,2 ns  847961,04        64
countParallelN      10        7033356,6 ns  860704,39        64
countParallelN      11        9417570,0 ns 3797677,81        64
countParallelN      12       12357017,2 ns 2424891,26        32
countParallelN      13        9607343,8 ns  870049,35        32
countParallelN      14       10986648,8 ns 2181403,31        32
countParallelN      15        9142860,0 ns  718556,67        32
countParallelN      16        9033910,6 ns  760171,67        32

BUILD SUCCESSFUL in 2m 9s
3 actionable tasks: 1 executed, 2 up-to-date
```

2. We benchmarked the code on a computer with 4 cores.

The results show a trend, with the increase of threads from 1 to 4 the execution time generally decreases fast. This is expected, as more cores can execute the threads more effectively.
the best execution time is as expected around 4 since the hardware has 4 cores. However, from 4 threads, the execution time does not decrease since we have already exploited the hardware to the max, over that we create more workload initializing and starting more threads.

Exercise 5.3

```
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 17.0.2
# CPU:  AMD64 Family 23 Model 24 Stepping 1, AuthenticAMD; 8 "cores"
# Date: 2023-10-07T16:48:30+0200
Normal add                          1,4 ns       0,10  268435456
Volatile add                       12,1 ns       0,99   33554432
```

Incrementing volatile int takes significantly more time, since each increment of the volatile variable is not saved in a cache but on the main memory (and accessing the memory is expensive timewise). So as we expected the volatile takes longer than the one without any synchronization.

Exercise 5.4

1. See LongCounter.java for implementation
2.

```
> Task :app:run
Array Size: 5697
# Occurences of ipsum :1430

BUILD SUCCESSFUL in 2s
```

We found something strange: using ctrl+f we find more instances. We do not think that this error is given by our synchronization since we found the same number in the sequential search and with the parallelized one.

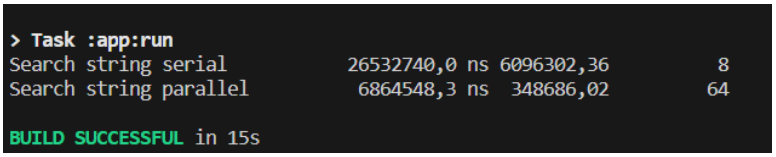
3.

```
> Task :app:run
Search string serial          22051567,5 ns 2261878,51        16

BUILD SUCCESSFUL in 9s
```

4. See TestTimeSearch.java for implementation

```
> Task :app:run
Thread has found 366 occurencies from 0 to 1424
Thread has found 358 occurencies from 4272 to 5697
Thread has found 353 occurencies from 1424 to 2848
Thread has found 353 occurencies from 2848 to 4272
# Occurences of ipsum :1430

BUILD SUCCESSFUL in 2s
```

5.



We noticed that the parallel searching is way faster since the threads do the search independently in different parts of the file, however even if we used 4 threads the time is not divided by 4 because initializing and starting threads takes time, and also does synchronization over the shared variable.