CPU: 1,4 GHz Quad-Core Intel Core i5, 8 cores

**for k = 20**

| | naive | smart | parallel |
|---|---|---|---|
| **1.000** | 0.281112 | 0.029562 | 0.638237 |
| **10.000** | 0.958619 | 0.164268 | 0.939032 |
| **100.000** | 10.732146 | 0.116836 | 1.219417 |
| **1.000.000** | 84.040572 | 1.241737 | 1.232423 |
| **10.000.000** | 847.179149 | 11.945148 | 7.096457 |
| **100.000.000** | 9998.507268 | 117.622898 | 77.187602 |

**for k = 100**

| | naive | smart | parallel |
|---|---|---|---|
| **1.000** | 0.07964 | 0.063689 | 3.261356 |
| **10.000** | 0.949957 | 0.136041 | 1.54201 |
| **100.000** | 9.03653 | 0.319341 | 0.983597 |
| **1.000.000** | 79.302282 | 1.251296 | 1.567302 |
| **10.000.000** | 868.535534 | 12.023208 | 8.620499 |
| **100.000.000** | 9894.247563 | 120.566042 | 73.855185 |

As the tables above show, the sequential algorithms slow down as the n value increases, whereas the parallel one becomes more effective in comparison. The parallel code gives speedup >1 for the values up to 10.000 for k = 20, and barely for 100.000 for k = 100. As each thread often has a different number of iterations, the threads may need to wait for each other resulting in increased execution time.