# AN EMPIRICAL ANALYSIS OF THE UGSORT ALGORITHM

Tree, Ian J.

unaffiliated researcher

Author Note

Ian J. Tree, unaffiliated researcher

Eindhoven, the Netherlands

Email: ian.tree@acm.org

## Abstract

This paper provides the results of an empirical study of the performance envelope of a sample implementation of the UGSort merge sort algorithm.

*Keywords*: empirical, performance, UGSort, sort, merge

# An Empirical Study of the Performance of the UGSort Algorithm

This paper details an empirical study of the performance characteristics of a sample implementation of the UGSort merge sort algorithm. Different aspects of the performance profile of the algorithm are investigated using a common set of testing methodologies.

## Testing Methods and Materials

### The UGSort Application

The UGSort application is a testbed for an implementation of the UGSort merge sort algorithm. The application will sort text files based on a fixed length ascii key at a given offset in each record in the unsorted file. Sorted output will be written to a designated output file. The implementation is minimally optimised providing indicative timing for any implementation of the algorithm. The application is minimally instrumented to provide the ability to perform timing comparisons for different scenarios.

The application is a practical implementation of the UGSort algorithm rather than a simplified sort kernel implementation that would be used to explore the theoretical time complexity of the algorithm.

### Testing Protocol

All tests are performed using a common protocol. An individual test configuration is run ten times in succession the run time of each test is recorded using Measure-Command on Windows and the time command on Linux. The slowest three run time results are discarded and the average of each measure for the remaining seven runs are used as the results.

### Testing Configurations

#### Windows.

A dedicated laptop for development, testing and simulations.

| | | |
|---|---|---|
| Processor | AMD Ryzen 7 5800H with Radeon Graphics | 3.20 GHz |

| | |
|---|---|
| Installed RAM | 32.0 GB (31.9 GB usable) |
| System type | 64-bit operating system, x64-based processor |
| Edition | Windows 11 Home |
| Version | 22H2 |
| OS build | 22621.1992 |
| Disk | 1,000 GB SSD |
| Microsoft | Visual Studio Community 2022 |
| Version | 17.6.5 |
| Visual Studio. | 17.Release/17.6.5+33829.357 |
| Compilation: | /O2 /W4 |

**Linux.**

A development and testing virtual server.

| | |
|---|---|
| OS: | CentOS Linux 7 (Core) |
| Kernel: | 3.10.0-1160.76.1.el7.x86_64 #1 SMP |
| CPU(s): | 4 |
| Thread(s) per core: | 1 |
| Core(s) per socket: | 1 |
| Socket(s): | 4 |
| CPU MHz: | 2350.000 |
| BogoMIPS: | 4700.00 |
| L1d cache: | 32K |
| L1i cache: | 32K |
| L2 cache: | 512K |
| L3 cache: | 16384K |
| Memory: | 7820 |

gcc version: 4.8.5 20150623 (Red Hat 4.8.5-44) (GCC)

cmake version 2.8.12.2

Compilation: -std=c++11 –O2 -Wall

**Test Data**

Testing uses files that have been prepared for individual studies. The default test set comprises files of text records with a randomly generated 20 numeric character key at the start of each record, the files contain 250,000 to 5,000,000 records at 250,000 intervals.

Best-case test files are created from the random test files by sorting them on the test key into descending sequence.

## Studies

All timing measurements are given in milliseconds (ms) unless explicitly stated. The following sections describe each of the common timings that may be recorded in results tables.

1. T_LD – The time taken to load the test data into memory.
2. T_SI – The time taken to complete the partitioning of the input data into the array of double ended queues. This time excludes any time spent performing pre-emptive merges.
3. T_PM – The time taken performing pre-emptive merges during the sort input phase.
4. CSI – The cumulative time spent in the sort input phase i.e., T_SI + T_PM.
5. T_FM – the time spent in performing the final merge, resulting in the keys being in a single double ended queue.
6. CM – The cumulative merge time i.e., T_PM + T_FM.
7. T_SO – The time spent iterating the result queue and building the output buffer with the input data in the desired sequence.

8. T_SD – The time spent writing the output buffer to disk.
9. T_S – The total sort time excluding loading the input data and storing the output data.
10. RT – The total runtime of the test application, this is measured external to the application.

## 1. 64bit (x64) vs. 32bit (x86)

This study will compare the performance of 64-bit and 32-bit applications using a 1,000,000 random test dataset.

### Windows Results.

*Table 1. x64 vs x86 timing comparison on Widows*

| Arch | T_LD | T_SI | T_PM | CSI | T_FM | T_SO | T_SD | T_S | RT |
|---|---|---|---|---|---|---|---|---|---|
| x64 | 13.1 | 567.4 | 403.3 | 970.7 | 231.0 | 48.1 | 37.6 | 1254.4 | 1331.6 |
| x86 | 13.0 | 513.9 | 450.0 | 963.9 | 232.3 | 65.6 | 37.1 | 1265.1 | 1345.6 |

### Linux Results.

*Table 2. x64 vs x86 timing comparison on Linux*

| Arch | T_LD | T_SI | T_PM | CSI | T_FM | T_SO | T_SD | T_S | RT |
|---|---|---|---|---|---|---|---|---|---|
| x64 | 13.7 | 1560.4 | 818.0 | 2378.4 | 417.1 | 137.1 | 18.3 | 2791.0 | 2975.0 |
| x86 | 14.0 | 1752.6 | 856.4 | 2609.0 | 434.7 | 156.4 | 18.9 | 3201.1 | 3244.4 |

### Observations and Analysis

As expected, the Linux timings are much slower than the Windows timings as the test platform for Linux is much less powerful than the Windows test platform. In both cases there is a slight performance benefit from using the x64 (64 bit) application over the x86 (32 bit) application. Subsequent studies will use the x64 (64 bit) test application.

## 2. Random Keys

This study will examine the relationship between the number of keys sorted (n) and the sort time. Tests will examine the performance on a range of random input files from 250,000 keys up to 5,000,000 keys in 250,000 increments. The release x64 build of the UGSort application is used for all tests.

### Windows Results.

*Table 3. timing comparisons for different n on Windows*

| n (M) | T_LD | T_SI | T_PM | CSI | T_FM | T_SO | T_SD | T_S | RT |
|---|---|---|---|---|---|---|---|---|---|
| 0.25 | 2.1 | 91.0 | 52.7 | 143.7 | 42.1 | 9.1 | 4.0 | 199.0 | 227.4 |
| 0.5 | 5 | 215.6 | 158.9 | 374.4 | 99.6 | 21.9 | 14.4 | 498.6 | 541.1 |
| 0.75 | 10.7 | 372.7 | 287.6 | 660.3 | 151.6 | 34.6 | 25.6 | 851.4 | 912.1 |
| 1.0 | 15.4 | 566.0 | 407.9 | 973.9 | 230.3 | 48.3 | 40.3 | 1257.7 | 1339.3 |
| 1.25 | 14.4 | 812.9 | 552.7 | 1365.6 | 312.7 | 61.0 | 50.1 | 1741.7 | 1834.1 |
| 1.5 | 17.7 | 1066.6 | 698.4 | 1765.0 | 386.0 | 74.4 | 64.4 | 2230.7 | 2341.9 |
| 1.75 | 20.1 | 1337.9 | 829.4 | 2167.3 | 457.1 | 87.0 | 73.9 | 2717.4 | 2837.4 |
| 2.0 | 23.1 | 1588.9 | 975.1 | 2564.0 | 502.7 | 98.7 | 89.6 | 3173.9 | 3318.1 |
| 2.25 | 26.1 | 1972.7 | 1144.6 | 3117.3 | 577.3 | 113.1 | 100.0 | 3814.3 | 3977.4 |
| 2.5 | 29.1 | 2303.6 | 1296.7 | 3600.3 | 653.0 | 127.9 | 113.3 | 4387.1 | 4567.0 |
| 2.75 | 32.3 | 2677.7 | 1392.4 | 4070.1 | 783.3 | 141.4 | 122.3 | 5001.4 | 5196.9 |
| 3.0 | 34.7 | 2973.4 | 1492.9 | 4466.3 | 875.0 | 152.9 | 133.0 | 5502.0 | 5711.1 |
| 3.25 | 38.1 | 3456.0 | 1725.1 | 5181.1 | 887.1 | 164.6 | 146.7 | 6239.7 | 6468.9 |
| 3.5 | 41 | 3850.3 | 1851.7 | 5702.0 | 1014.0 | 180.3 | 159.7 | 6903.6 | 7149.4 |
| 3.75 | 44 | 4268.9 | 2003.7 | 6272.6 | 1100.7 | 190.6 | 169.3 | 7570.7 | 7831.0 |
| 4.0 | 47.1 | 4612.0 | 2177.3 | 6789.3 | 1096.9 | 208.9 | 183.9 | 8100.3 | 8379.6 |
| 4.25 | 49.1 | 5203.1 | 2304.0 | 7507.1 | 1282.1 | 218.1 | 193.0 | 9012.0 | 9304.9 |
| 4.5 | 52.4 | 5700.1 | 2447.4 | 8147.6 | 1362.7 | 235.6 | 209.1 | 9752.0 | 10065.7 |
| 4.75 | 56 | 6208.3 | 2666.7 | 8875.0 | 1388.4 | 245.4 | 220.3 | 10510.0 | 10839.9 |
| 5.0 | 57.4 | 6535.3 | 2772.9 | 9308.1 | 1517.0 | 258.7 | 227.0 | 11088.1 | 11429.1 |

### Linux Results.

*Table 4. timing comparisons for different n on Linux*

| n (M) | T_LD | T_SI | T_PM | CSI | T_FM | T_SO | T_SD | T_S | RT |
|---|---|---|---|---|---|---|---|---|---|
| 0.25 | 3.0 | 217.1 | 126.1 | 343.3 | 95.6 | 33.4 | 4.1 | 473.9 | 486.7 |
| 0.5 | 5.9 | 585.7 | 358.6 | 944.3 | 210.3 | 74.1 | 9.0 | 1230.0 | 1252.1 |
| 0.75 | 8.1 | 1049.1 | 641.4 | 1690.6 | 321.3 | 115.3 | 14.0 | 2128.0 | 2159.1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 11.0 | 1595.1 | 899.0 | 2494.1 | 460.9 | 157.9 | 18.3 | 3114.4 | 3154.1 |
| 1.25 | 16.4 | 2298.3 | 1198.6 | 3496.9 | 613.0 | 200.4 | 23.4 | 4311.6 | 4363.7 |
| 1.5 | 17.6 | 3006.1 | 1461.0 | 4467.1 | 749.4 | 248.7 | 29.9 | 5466.7 | 5527.7 |
| 1.75 | 21.0 | 3811.1 | 1731.7 | 5542.9 | 886.0 | 284.6 | 33.0 | 6714.7 | 6785.1 |
| 2 | 24.3 | 4682.6 | 2058.4 | 6741.0 | 978.3 | 332.1 | 39.4 | 8052.6 | 8133.9 |
| 2.25 | 26.9 | 5657.6 | 2378.1 | 8035.7 | 1111.3 | 371.3 | 44.7 | 9519.0 | 9612.7 |
| 2.5 | 30.9 | 6693.4 | 2695.1 | 9388.6 | 1251.6 | 421.4 | 50.4 | 11062.4 | 11168.9 |
| 2.75 | 32.4 | 7784.1 | 2856.3 | 10640.4 | 1509.3 | 467.0 | 55.0 | 12617.9 | 12735.6 |
| 3 | 34.9 | 8879.3 | 3036.0 | 11915.3 | 1653.3 | 499.3 | 59.1 | 14069.1 | 14195.1 |
| 3.25 | 38.9 | 10035.6 | 3436.6 | 13472.1 | 1649.0 | 541.9 | 64.3 | 15663.9 | 15800.3 |
| 3.5 | 41.4 | 11219.3 | 3668.3 | 14887.6 | 1893.3 | 581.4 | 69.4 | 17363.7 | 17510.3 |
| 3.75 | 44.3 | 12461.6 | 3896.3 | 16357.9 | 2042.3 | 618.6 | 75.0 | 19020.1 | 19177.6 |
| 4 | 47.0 | 13948.4 | 4247.0 | 18195.4 | 2033.9 | 652.4 | 78.0 | 20882.9 | 21067.9 |
| 4.25 | 50.9 | 15194.3 | 4425.0 | 19619.3 | 2355.1 | 699.7 | 86.1 | 22675.1 | 22854.1 |
| 4.5 | 53.0 | 16619.7 | 4737.6 | 21357.3 | 2503.4 | 731.9 | 88.3 | 24593.6 | 24780.9 |
| 4.75 | 56.3 | 18231.0 | 5199.1 | 23430.1 | 2554.7 | 779.7 | 94.6 | 26765.9 | 26964.6 |
| 5 | 57.1 | 19832.3 | 5392.6 | 25224.9 | 2708.3 | 815.1 | 96.3 | 28749.3 | 28952.3 |

## Observations and Analysis

A linear regression on the sort time (t) in milliseconds gave the following relationships with n as the number of input keys.

Windows: $t(ms) = (2350 * (n/1000000)) - 1130$

with $R^2 = 0.9968$.

Linux: $t(ms) = (6000 * (n/1000000)) - 3000$

with $R^2 = 0.9963$.

The approximate throughput rates for Windows and Linux were respectively 500,000 and 200,000 keys per second.

*Figure 1. best fit plots for t vs. n on Windows*

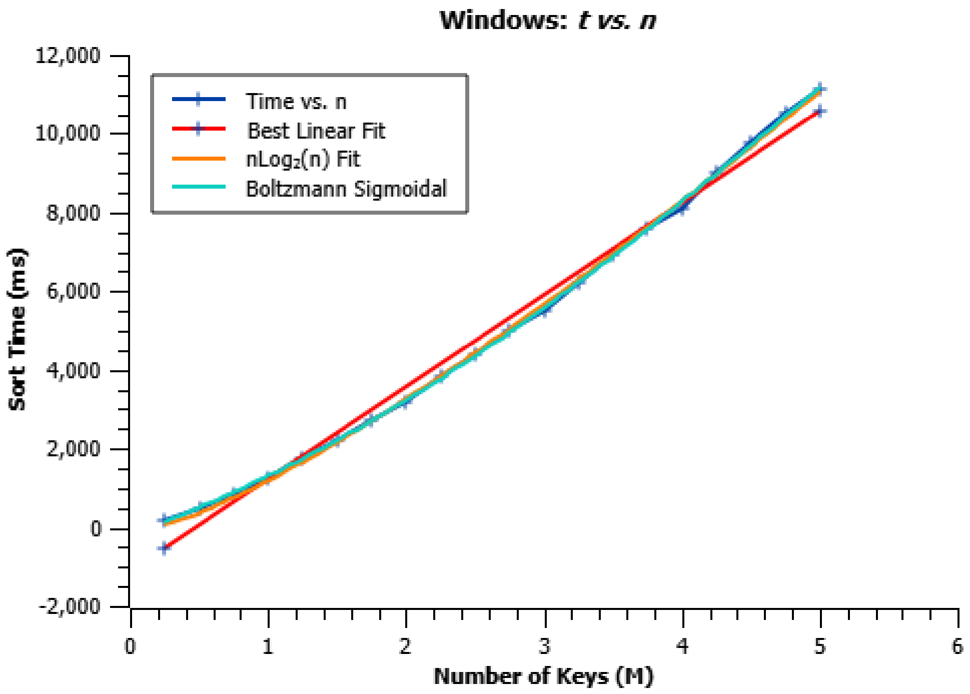
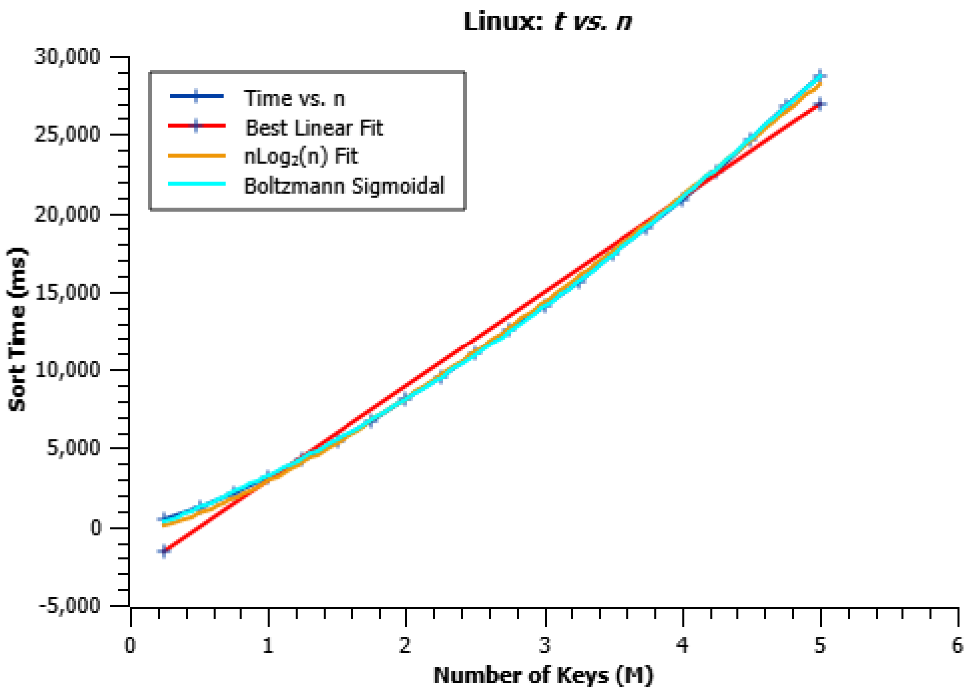


*Figure 2. best fit plots for t vs. n on Linux*



The plots show a typical logarithmic or sigmoidal deviation from the linear approximation. Sort algorithms based on merge typically show time complexity of nLog$_2$(n), therefore a best match is done on that basis.

Windows: $t(ms) = 450xLog_2(6.4x)$ with R$^2$ = 0.9998

Linux: $t(ms) = 1{,}200xLog_2(5.4x)$ with R$^2$ = 0.9997

The chart also includes a plot of the best fit for a Boltzmann Sigmoidal curve.

Windows: $t(ms) = (4{,}100 - 26{,}850)/\left(1 + e^{((x-5.1)/2.6)} + 26{,}850\right)$ R$^2$ = 0.9999

Linux: $t(ms) = (-12{,}000 - 87{,}600)/\left(1 + e^{((x-6.1)/3)} + 87{,}600\right)$ R$^2$ = 0.9999

For both Windows and Linux, the linear estimations for the sort time are as accurate as needed for run time estimations over the range being studied.

## 3. Best Case

This study will examine the performance profile for "best case" sample data. The data is constructed by pre-sorting the random samples into descending sequence. The release x64 build of the UGSort application is used for all tests.