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

Revised 12/09/2023 for v1.15 of the application using binary search.

An Empirical Study of the Performance of the UGSort Algorithm[i]

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 (CRLF or LF terminated records) 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.

All tests were conducted with UGSort v1.15.0.

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.

Data collection and collation was performed in Microsoft Excel™. All curve fitting, analysis and charting was done using SciDAVis v2.7.

Testing Configurations

Windows.

A dedicated laptop for development, testing and simulations.

Processor AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz

Installed RAM32.0 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, padded with random and serial data to an average record length of 61 bytes, 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. Worst-case test datasets are prepared by taking the corresponding best-case file and emitting it in alternating tail and top sequence.

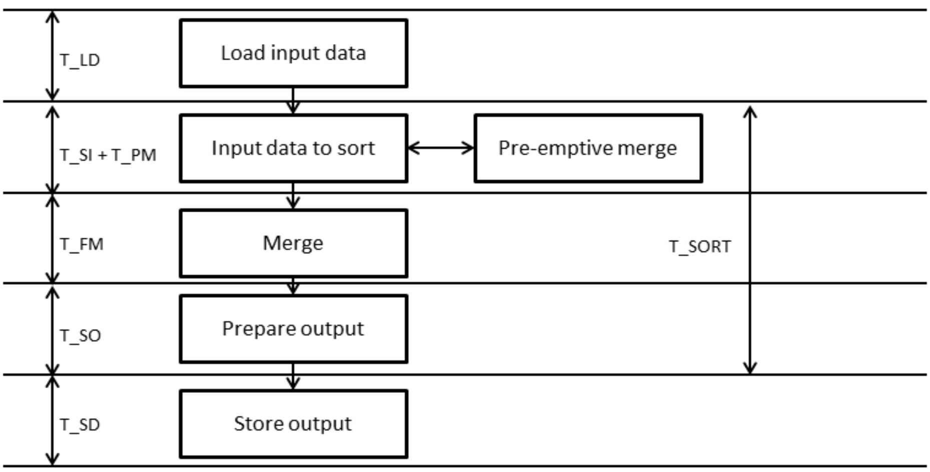
STUDIES

All timing measurements (t) are given in milliseconds (ms) unless explicitly stated. Key counts (n) are given in millions of keys. 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.

Figure 1. Timing Diagram



All tests are performed using the in-memory (fastest) mode of operation.

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

This study will compare the performance of 64-bit and 32-bit applications using a 5,000,000 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	59.0	1076.4	235.6	1312.0	1884.0	258.6	222.0	3458.4	3794
x86	58.3	942.6	234.9	1177.4	1710.3	357.7	222.6	3250.4	3591

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	82.0	2104.9	435.7	2540.6	3707.6	857.6	105.7	7107.0	7353
x86	86.7	2256.7	415.3	2672.0	3570.0	928.6	99.1	7171.9	7416

Observations and Analysis

As expected, the Linux timings are much slower than the Windows timings as the test platform for Linux is less powerful than the Windows test platform. 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 v1.15.0 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	3.0	32.9	4.0	36.9	57.4	9.4	4.0	110.4	138
0.50	5	70.3	10.9	81.1	135.1	21.6	14.6	244.3	287
0.75	8.0	108.4	25.7	134.1	212.9	34.7	32.1	389.0	454
1.00	11.0	150.4	53.4	203.9	277.0	47.0	43.1	536.1	617
1.25	14.3	203.6	60.0	263.6	386.7	60.6	52.7	716.1	811
1.50	17.1	251.7	55.7	307.4	497.9	74.4	66.7	888.0	1002
1.75	20.1	301.6	62.4	364.0	597.3	87.7	75.6	1057.1	1185
2.00	22.9	345.1	105.0	450.1	638.9	99.9	87.6	1195.0	1339
2.25	26.0	407.3	121.3	528.6	774.9	114.4	98.4	1426.6	1587
2.50	29.0	461.6	126.7	588.3	871.1	127.3	113.0	1593.0	1773
2.75	32.0	515.7	121.9	637.6	981.7	137.9	122.3	1764.3	1958
3.00	34.1	573.7	107.4	681.1	1086.1	151.9	134.4	1927.1	2137
3.25	37.9	633.1	128.9	762.0	1187.6	167.7	148.0	2124.3	2355
3.50	41.3	707.3	240.0	947.3	1172.4	181.6	160.9	2310.3	2557
3.75	44.9	768.7	133.6	902.3	1399.0	193.1	171.6	2501.0	2764
4.00	46.3	818.6	222.3	1040.9	1456.3	205.9	180.7	2709.4	2984
4.25	51.1	901.0	228.1	1129.1	1636.9	236.0	186.1	3010.7	3294
4.50	53.1	989.0	243.4	1232.4	1755.6	245.1	200.0	3240.9	3543
4.75	55.3	1015.7	263.1	1278.9	1758.1	240.9	213.7	3285.6	3606
5.00	56.7	1063.7	231.7	1295.4	1882.4	255.1	222.1	3440.4	3772

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	4.0	63.4	8.3	71.7	106.0	31.6	5.1	210.6	227
0.50	9.0	135.7	18.1	153.9	234.3	66.3	10.3	455.4	484
0.75	12.3	212.0	40.3	252.3	364.1	100.3	15.4	717.6	757
1.00	12.7	301.3	89.4	390.7	509.0	137.9	20.4	1039.0	1087
1.25	18.4	386.1	99.9	486.0	683.9	178.1	25.4	1349.1	1409
1.50	20.6	469.0	92.6	561.6	846.6	211.7	29.4	1620.9	1687
1.75	24.7	565.7	105.1	670.9	1030.1	249.6	34.1	1951.7	2028
2.00	23.9	657.4	181.0	838.4	1118.4	284.4	37.7	2242.6	2323
2.25	29.1	753.1	203.9	957.0	1347.3	329.6	44.1	2634.7	2729
2.50	32.1	854.1	213.4	1067.6	1503.9	362.0	47.9	2934.7	3039
2.75	37.6	961.0	215.4	1176.4	1733.7	410.0	54.0	3321.4	3438
3.00	33.6	1071.7	184.6	1256.3	1865.1	443.6	56.6	3566.3	3685
3.25	38.3	1190.4	214.9	1405.3	2022.3	470.4	62.9	3899.3	4030
3.50	45.9	1316.4	403.4	1719.9	2047.0	528.6	72.3	4301.4	4452
3.75	47.7	1462.1	224.0	1686.1	2424.4	565.4	73.7	4676.9	4834
4.00	47.7	1532.9	374.9	1907.7	2554.4	601.6	78.3	5064.9	5230
4.25	61.4	1654.6	368.6	2023.1	2667.9	640.3	85.1	5335.4	5526
4.50	61.0	1778.1	399.4	2177.6	2919.6	680.3	89.6	5778.7	5976
4.75	68.3	1898.4	441.0	2339.4	3024.4	711.1	93.4	6076.1	6286
5.00	63.4	2051.4	392.7	2444.1	3230.9	745.6	97.6	6421.7	6635

Observations and Analysis

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

$t = mn + c$

Where *m* is the slope and *c* the intercept.

For Windows m = 726 and c = -183, with R^2 = 0.9991.

For Linux m = 1,328 and c = -307, with R^2 = 0.9995.

The approximate throughput rates for Windows and Linux were respectively 1,500,000 and 800,000 keys per second.

Figure 2. best fit plots for t vs. n on Windows

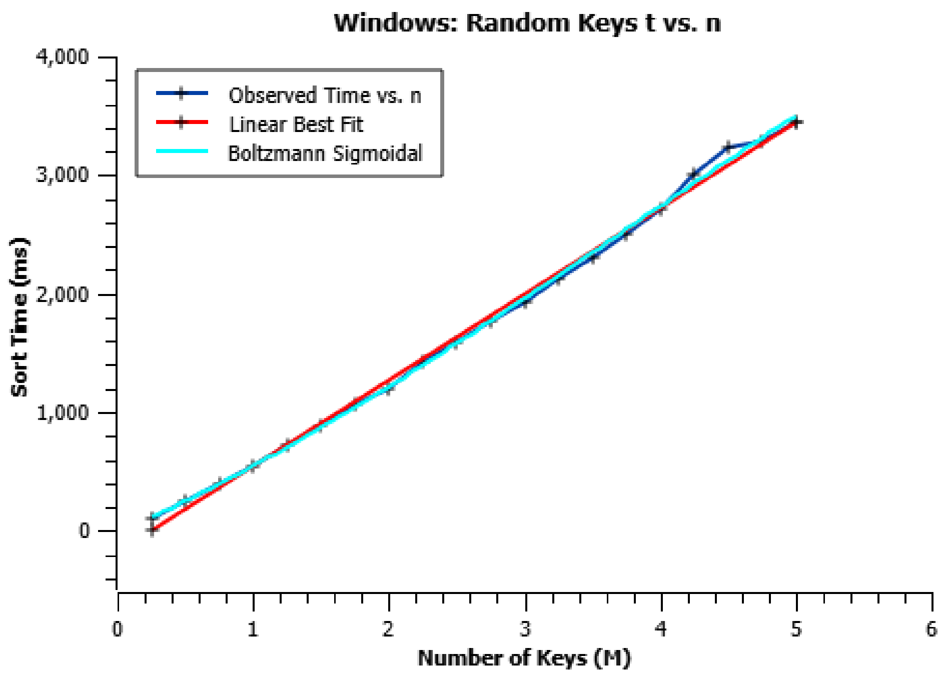
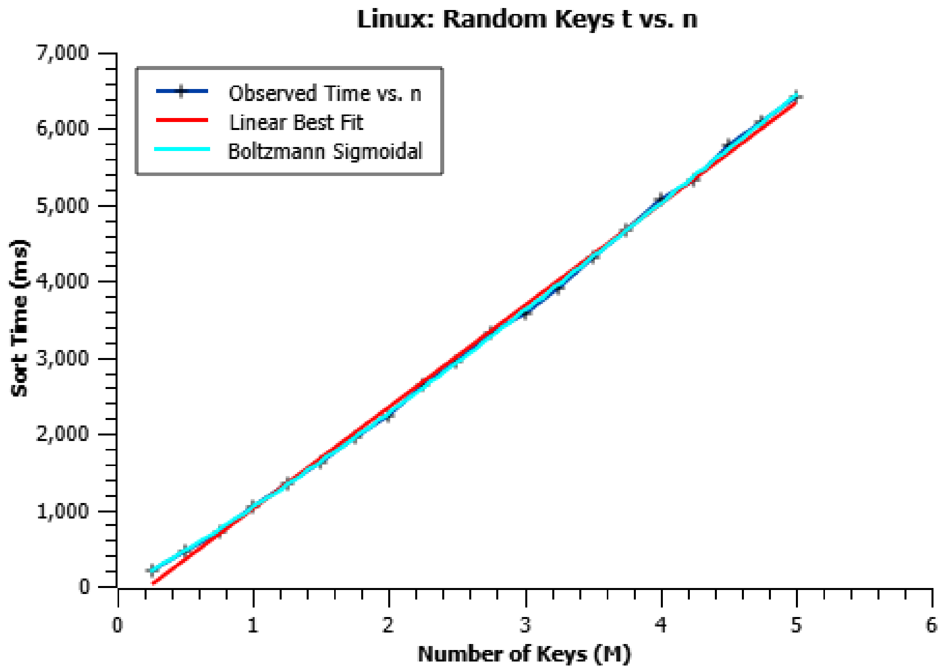


Figure 3. best fit plots for t vs. n on Linux



The plots show a typical logarithmic or sigmoidal deviation from the linear approximation. Sort algorithms based on merge typically show time complexity of $n\mathrm{Log}_2(n)$, therefore a best match was attempted on that basis, no match was possible.

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

$$t = ((t_1-t_2)/(1+e^{((n-n0)/dn)})) + t_2$$

Where t_1 is the initial value of t, t_2 the final value, n_0 is the mid-value of n and dn is the time constant.

For Windows t_1 = -1,760, t_2 = 6,750, n_0 = 3.69 and dn = 2.72 matches with R^2 = 0.9996.

For Linux t_1 = -4,734, t_2 = 15,200, n_0 = 4.2 and dn = 3.5 matches with 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 v1.15.0 of the UGSort application is used for all tests.

Windows Results.

Table 5. 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	2.3	5.0	0.0	5.0	0.0	3.1	4.0	11.9	41
0.50	5	13.4	0.0	13.4	0.0	7.0	14.7	26.3	69
0.75	8.0	24.1	0.0	24.1	0.0	11.0	25.4	42.4	100
1.00	11.0	39.3	0.0	39.3	0.0	14.1	36.9	60.6	135
1.25	14.3	59.1	0.0	59.1	0.0	18.9	49.1	85.0	177
1.50	17.0	78.1	0.0	78.1	0.0	22.0	60.3	106.4	213
1.75	20.0	103.4	0.0	103.4	0.0	25.3	71.6	136.7	259
2.00	23.0	133.6	0.0	133.6	0.0	29.9	81.3	171.0	307
2.25	26.1	165.9	0.0	165.9	0.0	33.6	95.4	203.9	360
2.50	29.0	203.0	0.0	203.0	0.0	37.6	106.9	243.7	419
2.75	32.0	233.0	0.0	233.0	0.0	41.0	117.0	278.7	467
3.00	34.6	278.3	0.0	278.3	0.0	45.3	127.3	330.4	533
3.25	37.9	324.4	0.0	324.4	0.0	47.9	141.7	375.1	597
3.50	40.6	376.4	0.0	376.4	0.0	52.0	153.4	432.6	670
3.75	43.9	431.3	0.0	431.3	0.0	59.1	169.1	494.3	752
4.00	45.7	492.1	0.0	492.1	0.0	58.6	177.7	555.6	826
4.25	49.9	548.6	0.0	548.6	0.0	62.9	186.7	607.7	895
4.50	52.4	608.3	0.0	608.3	0.0	68.1	199.9	679.4	983
4.75	54.7	675.7	0.0	675.7	0.0	71.7	219.6	753.3	1082
5.00	57.4	753.6	0.0	753.6	0.0	82.3	224.1	842.7	1178

Linux Results

Table 6. 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	3.3	7.1	0.0	7.1	0.0	6.0	4.7	14.0	28
0.50	6.9	15.6	0.0	15.6	0.0	11.4	10.3	27.6	53
0.75	9.6	25.1	0.0	25.1	0.0	16.0	14.7	42.0	76
1.00	12.1	37.4	0.0	37.4	0.0	21.0	19.4	59.1	102
1.25	15.6	50.0	0.0	50.0	0.0	29.1	24.9	79.6	134
1.50	17.4	60.9	0.0	60.9	0.0	35.7	29.4	97.4	159
1.75	20.4	75.6	0.0	75.6	0.0	42.1	34.4	118.4	189
2.00	22.4	92.6	0.0	92.6	0.0	48.7	38.7	141.9	222
2.25	26.0	110.4	0.0	110.4	0.0	53.7	44.7	164.7	257
2.50	30.6	126.7	0.0	126.7	0.0	59.3	49.7	186.7	289
2.75	33.0	145.1	0.0	145.1	0.0	69.9	54.1	215.3	329
3.00	36.6	167.7	0.0	167.7	0.0	80.6	57.7	248.9	373
3.25	41.0	190.9	0.0	190.9	0.0	91.6	64.9	282.9	424
3.50	45.1	216.1	0.0	216.1	0.0	103.3	69.9	320.0	475
3.75	50.6	236.1	0.0	236.1	0.0	113.4	73.3	350.1	515
4.00	51.7	267.1	0.0	267.1	0.0	121.6	76.3	389.4	561
4.25	56.4	291.3	0.0	291.3	0.0	134.6	83.0	426.7	615
4.50	59.6	313.6	0.0	313.6	0.0	141.6	89.7	455.9	652
4.75	59.0	347.6	0.0	347.6	0.0	146.1	93.0	494.3	696
5.00	59.9	381.9	0.0	381.9	0.0	153.9	95.9	536.6	744

Observations and Analysis

The first observation is that despite running on the less powerful platform the Linux tests bettered the Windows tests for all values of n. The best-case data sets do not require any merging as the data is pre-sorted and therefore is loaded to only a single partition, thus, T_PM and T_FM are 0 in all tests.

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

$t = mn + c$

Where m is the slope and c the intercept.

For Windows m = 171 and c = -128, with R^2 = 0.983.

For Linux m = 111 and c = -59, with R^2 = 0.993.

The approximate throughput rates for Windows and Linux were respectively 6,000,000 and 9,000,000 keys per second.

Figure 4. best fit plots for t vs. n on Windows

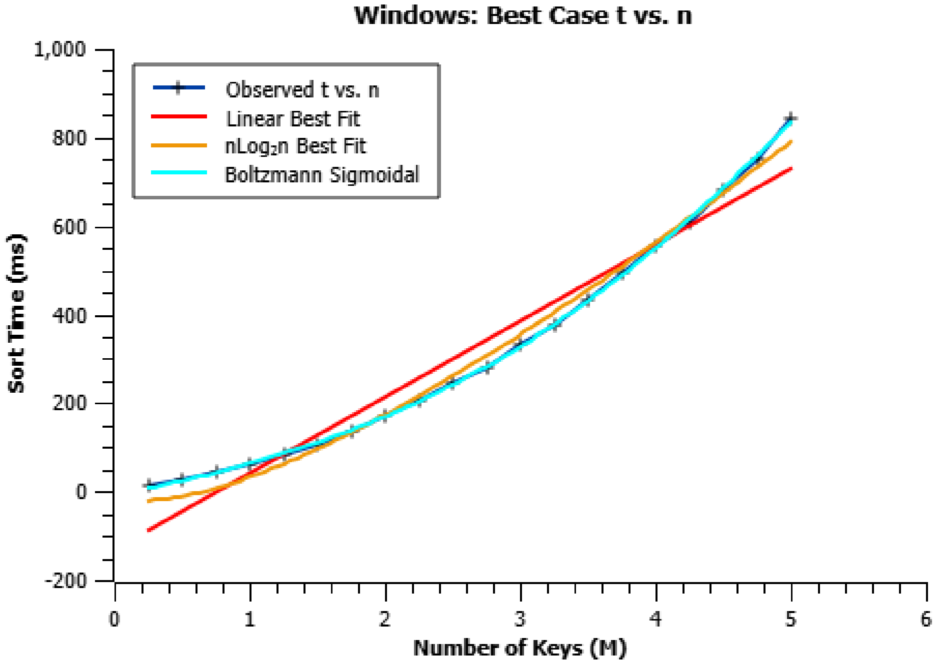
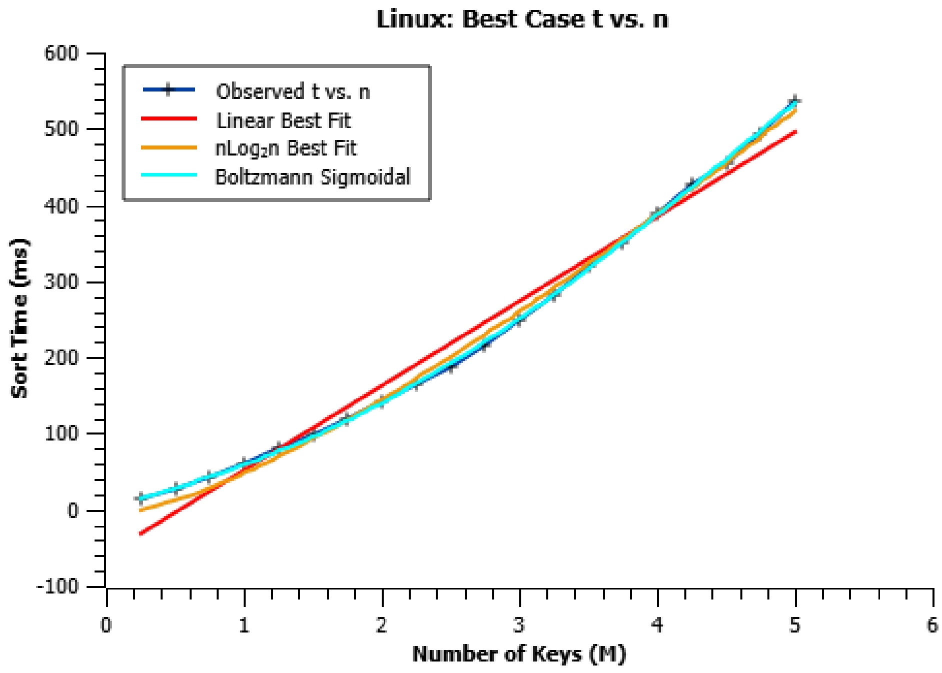


Figure 5. best fit plots for t vs. n on Linux



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.

$t = mnLog_2(kn)$

Where *m* is the scale and *k* a constant.

For Windows m = 54 and k = 1.5, with $R^2 = 0.9965$.

For Linux m = 24.6 and k = 3.8, with $R^2 = 0.9989$.

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

$t = ((t_1\text{-}t_2)/(1+e^{((n\text{-}n0)/dn)})) + t_2$

Where t_1 is the initial value of *t*, t_2 the final value, n_0 is the mid-value of *n* and *dn* is the time constant.

For Windows $t_1 = -118$, $t_2 = 2{,}628$, $n_0 = 6.26$ and $dn = 2.0$

matches with $R^2 = 0.9999$.

For Linux $t_1 = -88$, $t_2 = 1{,}044$, $n_0 = 4.6$ and $dn = 1.9$

matches with $R^2 = 0.9999$.

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.

4. Worst-Case

This study will examine the performance profile for "worst-case" sample data. Worst-case test datasets are prepared by taking the corresponding best-case file and emitting it in alternating tail and top sequence. The release x64 build v1.15.0 of the UGSort application is used for all tests.

Windows Results.

Table 5. 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	2.3	372.6	90.4	463.0	10.0	3.0	4.0	483.6	511
0.50	5	573.1	526.9	1100.0	23.9	7.3	14.9	1140.4	1184
0.75	8.0	745.7	1165.6	1911.3	35.1	12.0	27.6	1962.3	2023
1.00	11.0	891.3	1933.9	2825.1	51.0	15.7	38.9	2898.9	2976
1.25	14.1	1040.1	2893.0	3933.1	84.4	19.6	49.7	4044.4	4137
1.50	17.1	1154.7	3921.0	5075.7	88.0	26.6	61.9	5196.0	5306
1.75	20.0	1251.1	5082.1	6333.3	81.7	27.6	72.1	6448.9	6573
2.00	23.0	1390.7	6223.3	7614.0	102.9	31.6	82.9	7756.4	7897
2.25	26.0	1486.1	7570.4	9056.6	131.1	36.1	100.1	9229.7	9393
2.50	36.9	1617.1	8929.0	10546.1	129.7	43.0	113.3	10725.9	10914
2.75	40.3	1700.0	10381.0	12081.0	143.4	44.4	124.0	12276.3	12482
3.00	35.0	1811.9	11825.7	13637.6	164.7	52.7	135.3	13863.0	14075
3.25	39.4	1914.7	13465.6	15380.3	181.1	55.6	149.9	15626.3	15859
3.50	41.1	2009.6	15082.4	17092.0	217.9	61.7	168.3	17380.7	17635
3.75	44.0	2082.6	16890.4	18973.0	213.4	64.7	177.3	19258.4	19529
4.00	46.4	2182.7	18521.4	20704.1	231.4	63.4	191.9	21008.1	21296
4.25	49.0	2284.4	20409.0	22693.4	251.0	68.4	207.4	23019.9	23324
4.50	52.1	2391.1	22350.1	24741.3	304.3	79.1	213.4	25133.7	25449
4.75	70.9	2472.3	24364.9	26837.1	303.4	84.0	225.0	27233.9	27581
5.00	57.0	2547.3	25968.6	28515.9	287.4	79.6	228.3	28890.9	29230

Linux Results.

Table 6. 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	4.0	309.3	397.0	706.3	23.4	6.9	5.1	737.4	754
0.50	7.9	521.0	1296.4	1817.4	45.7	12.3	10.1	1876.6	1903
0.75	11.3	711.4	2351.0	3062.4	62.1	18.1	14.9	3143.9	3180
1.00	14.9	870.1	3617.9	4488.0	87.1	22.9	19.9	4598.7	4645
1.25	18.1	1041.3	5153.7	6195.0	110.0	31.9	26.0	6338.0	6395
1.50	18.4	1180.7	6709.3	7890.0	129.6	38.4	30.4	8059.1	8123
1.75	20.1	1325.1	8730.1	10055.3	139.0	44.9	35.0	10240.6	10312
2.00	27.6	1450.1	10138.9	11589.0	168.9	53.0	38.7	11812.3	11896
2.25	28.9	1588.7	12436.9	14025.6	189.6	56.4	46.4	14272.7	14369
2.50	34.9	1739.0	14503.0	16242.0	210.3	74.3	50.6	16527.6	16638
2.75	38.1	1863.4	16802.0	18665.4	227.7	92.4	55.6	18986.7	19107
3.00	42.9	1985.9	18785.0	20770.9	243.9	111.4	59.1	21127.4	21258
3.25	43.4	2090.9	21168.6	23259.4	266.0	120.9	64.6	23647.6	23788
3.50	46.0	2237.3	23993.0	26230.3	334.7	134.7	70.7	26700.9	26852
3.75	52.1	2361.9	27373.3	29735.1	396.4	145.3	73.7	30278.0	30444
4.00	47.3	2463.6	29002.3	31465.9	337.4	157.1	78.3	31961.9	32127
4.25	58.6	2577.3	31882.0	34459.3	394.9	168.3	88.0	35023.4	35216
4.50	67.4	2686.7	34733.0	37419.7	413.1	172.1	97.1	38006.1	38216
4.75	71.0	2821.7	37768.4	40590.1	407.7	189.9	100.4	41189.1	41414
5.00	81.3	2933.6	39885.6	42819.1	417.7	192.0	98.6	43430.0	43658

Observations and Analysis

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

$t = mn + c$

Where m is the slope and c the intercept.

For Windows m = 6,126 and c = -3,402, with R^2 = 0.995.

For Linux m = 9,255 and c = -4,896, with R^2 = 0.9957.

The approximate throughput rates for Windows and Linux were respectively 250,000 and 150,000 keys per second.

Figure 6. best fit plots for t vs. n on Windows

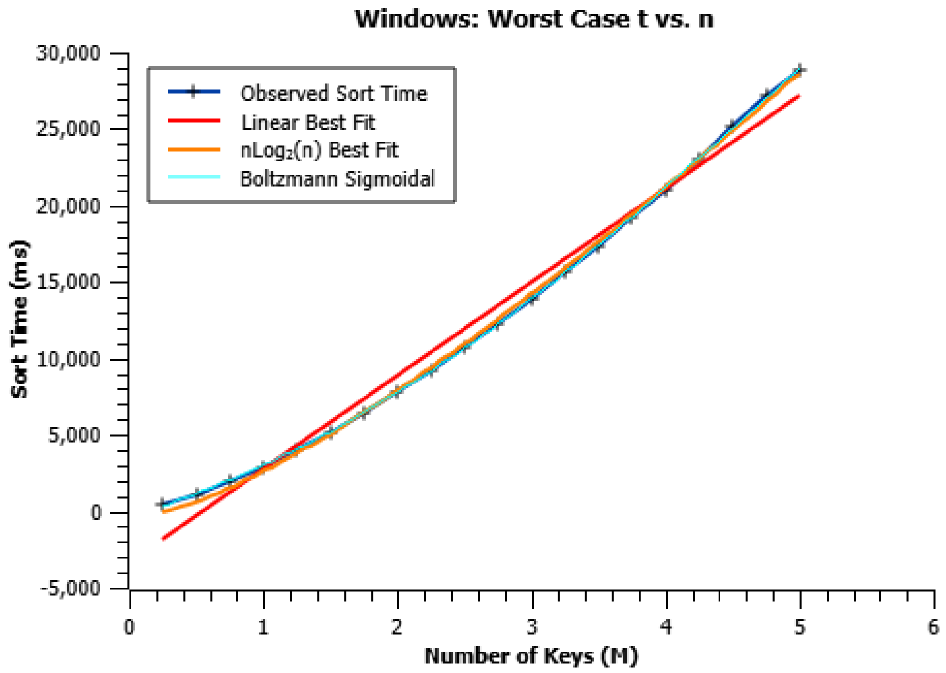
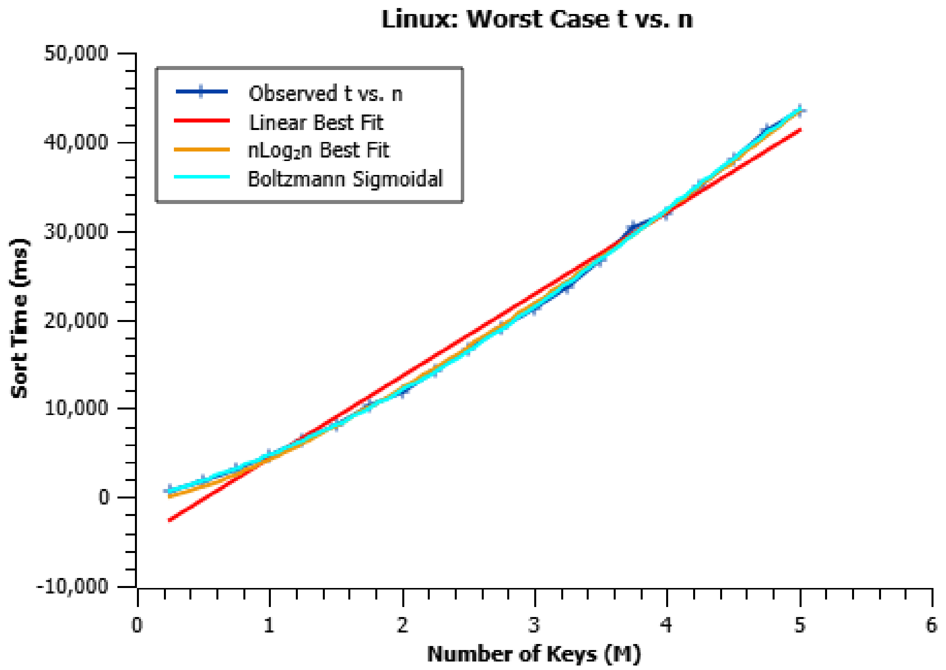


Figure 7. best fit plots for t vs. n on Linux



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.

$t = mnLog_2(kn)$

Where *m* is the scale and *k* a constant.

For Windows m = 1,351 and k = 3.8, with R^2 = 0.9996.

For Linux m = 1,931 and k = 4.5, with R^2 = 0.9996.

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

$t = ((t_1 - t_2)/(1+e^{((n-n0)/dn)})) + t_2$

Where t_1 is the initial value of *t*, t_2 the final value, n_0 is the mid-value of *n* and *dn* is the time constant.

For Windows t_1 = -7,179, t_2 = 60,347, n_0 = 4.69 and *dn* = 2.14 matches with R^2 = 0.9999.

For Linux t_1 = -9,804, t_2 = 80,014, n_0 = 4.24 and *dn* = 1.97 matches with R^2 = 0.9998.

5. Comparison with native OS Sort Utilities

This study compares the run time (RT) of different test sets (random, best-case and worst-case) of UGSort with the Sort utility provided with the OS. In each case the tests are run for the complete range of n (250,000 to 5,000,000) keys. Run times for the Sort utilities are measured using the time command on Linux and the Measure-Command PowerShell command on Windows.

Linux:> time sort *input file* >*output file*

Windows:> Measure-Command {sort.exe *input file* /O *output file*}

Windows Results.

Table 7. timing comparisons for different n on Windows

n (M)	Sort Rand	UGSort Rand	Sort Best	UGSort Best	Sort Worst	UGSort Worst
0.25	481	138	325	41	394	511
0.50	1061	287	665	69	868	1184
0.75	1696	454	994	100	1310	2023
1.00	2357	617	1374	135	1882	2976
1.25	3069	811	1737	177	2324	4137
1.50	3773	1002	2086	213	2994	5306
1.75	4581	1185	2489	259	3406	6573
2.00	5200	1339	2878	307	4078	7897
2.25	6162	1587	3310	360	4525	9393
2.50	6956	1773	3617	419	5160	10914
2.75	7566	1958	3931	467	5570	12482
3.00	8267	2137	4329	533	6322	14075
3.25	9086	2355	4729	597	6675	15859
3.50	9986	2557	5129	670	7474	17635
3.75	10581	2764	5541	752	7646	19529
4.00	11264	2984	5962	826	8671	21296
4.25	12374	3294	6415	895	8865	23324
4.50	13069	3543	6827	983	9729	25449
4.75	13893	3606	7249	1082	10016	27581
5.00	14784	3772	7488	1178	10898	29230

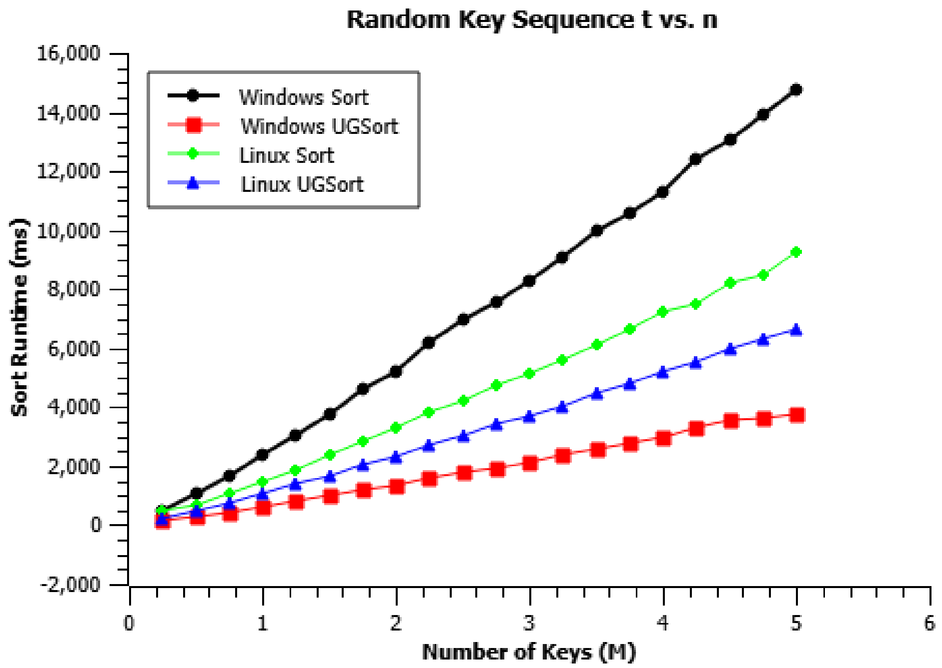
Linux Results.

Table 8. timing comparisons for different n on Linux

n (M)	Sort Rand	UGSort Rand	Sort Best	UGSort Best	Sort Worst	UGSort Worst
0.25	468	227	239	28	298	754
0.50	685	484	304	53	371	1903
0.75	1072	757	467	76	570	3180
1.00	1457	1087	624	102	750	4645
1.25	1876	1409	803	134	959	6395
1.50	2356	1687	1024	159	1229	8123
1.75	2839	2028	1238	189	1456	10312
2.00	3304	2323	1435	222	1753	11896
2.25	3797	2729	1685	257	2011	14369
2.50	4239	3039	1871	289	2202	16638
2.75	4732	3438	2148	329	2465	19107
3.00	5144	3685	2325	373	2738	21258
3.25	5619	4030	2535	424	2983	23788
3.50	6111	4452	2730	475	3234	26852
3.75	6636	4834	2907	515	3451	30444
4.00	7207	5230	3218	561	3763	32127
4.25	7519	5526	3383	615	3963	35216
4.50	8211	5976	3560	652	4269	38216
4.75	8484	6286	3850	696	4516	41414
5.00	9261	6635	4055	744	4697	43658

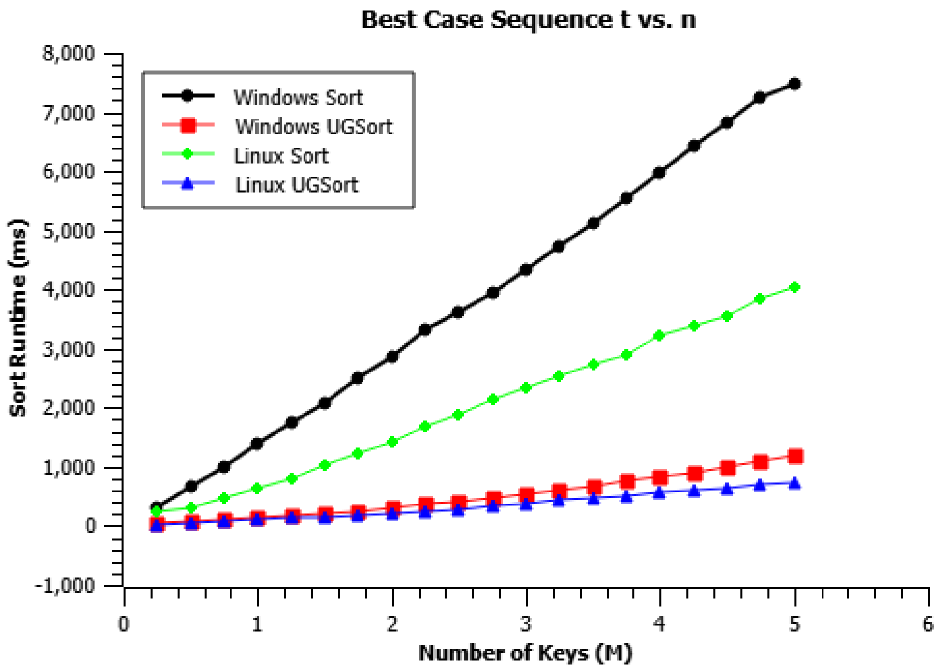
Observations and Analysis

Figure 8. comparison plots for random key sequence



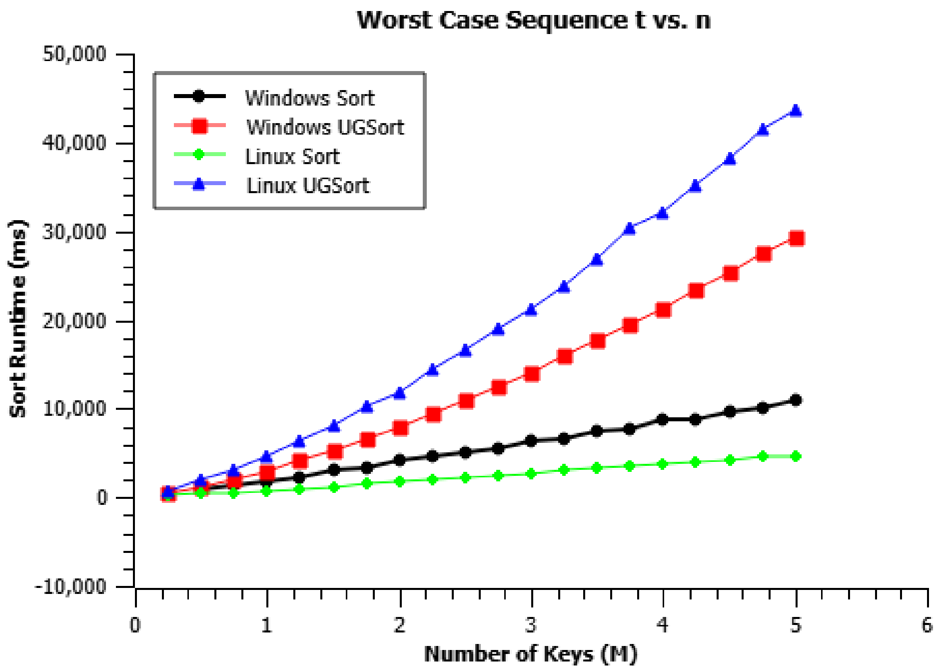
UGSort performed well on both Windows and Linux, outperforming the native Sort utilities by a significant margin.

Figure 9. comparison plots for best-case key sequence



The UGSort implementations on both Windows and Linux outperformed the native Sort utilities. The algorithm is well suited to exploiting the presortedness[ii] which is at a maximum in the best-case key sequence.

Figure 10. comparison plots for worst-case key sequence



UGSort on both Linux and Windows performed poorly on this sequence, which is not surprising as the sequence was designed to be highly toxic for the UGSort algorithm. The Sort utility on both platforms performed better than against the random key sequence runs, they can exploit the presortedness that is inherent in the worst-case key sequence.

CONCLUSION

The UGSort application performed well on both platforms, giving a near linear performance curve for random key sequences. Given that the application under test is only minimally optimised the performance is encouraging although, the Linux implementation did not perform as well as the Windows one. The performance on both platforms was outstanding for the best-case test sets, performing far better than the native Sort utilities. As expected, the worst-case test sets managed to significantly impair the performance of UGSort in comparison to the native Sort utilities.

The UGSort algorithm offers a predictable and acceptable performance cost over the range that was studied (250,000 to 5,000,000 keys).

The implementation of the binary search for partition selection has significantly improved the algorithm, reducing sort input times and the number of pre-emptive merges that are needed to maintain the performance.

FURTHER WORK

A theoretical study of the UGSort algorithm would underpin this study. The observed O(n) time complexity observed in the random key sequence tests should be explained. Such a study should resolve a relationship between sorting times and the degree of presortedness or sequence spoiling noted in the best and worst-case test sets.

REFERENCES

---

[i] The UGSort Algorithm, Tree Ian. J, 2023
https://github.com/UGSort-/docs/UGS-Algorithm.pdf
[ii] Sorting Presorted Files, Mehlhorn K, 1978