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[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.

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 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, 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.

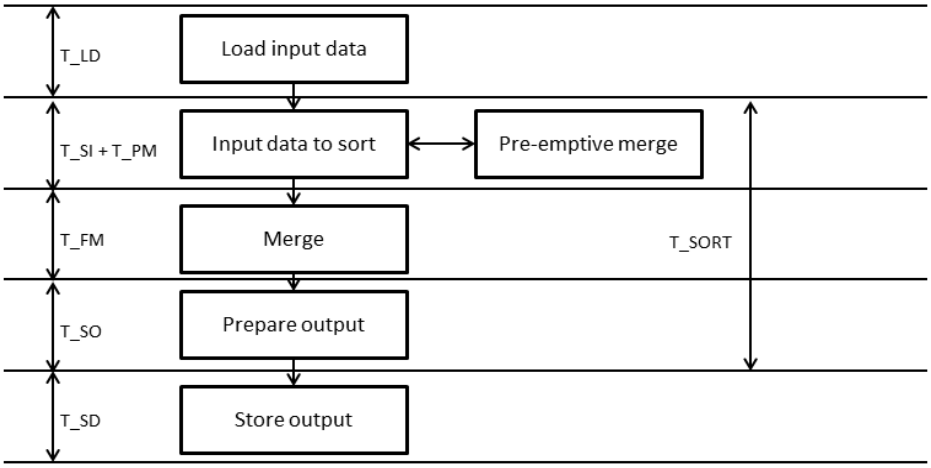
STUDIES

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



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
x86	13.0	513.9	450.0	963.9	232.3	65.6	37.1	1265.1	1345

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
x86	14.0	1752.6	856.4	2609.0	434.7	156.4	18.9	3201.1	3244

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. 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 v1.14.0 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	2.1	91.0	52.7	143.7	42.1	9.1	4.0	199.0	227
0.5	5	215.6	158.9	374.4	99.6	21.9	14.4	498.6	541
0.75	10.7	372.7	287.6	660.3	151.6	34.6	25.6	851.4	912
1.0	15.4	566.0	407.9	973.9	230.3	48.3	40.3	1257.7	1339
1.25	14.4	812.9	552.7	1365.6	312.7	61.0	50.1	1741.7	1834
1.5	17.7	1066.6	698.4	1765.0	386.0	74.4	64.4	2230.7	2342
1.75	20.1	1337.9	829.4	2167.3	457.1	87.0	73.9	2717.4	2837
2.0	23.1	1588.9	975.1	2564.0	502.7	98.7	89.6	3173.9	3318
2.25	26.1	1972.7	1144.6	3117.3	577.3	113.1	100.0	3814.3	3977
2.5	29.1	2303.6	1296.7	3600.3	653.0	127.9	113.3	4387.1	4567
2.75	32.3	2677.7	1392.4	4070.1	783.3	141.4	122.3	5001.4	5197
3.0	34.7	2973.4	1492.9	4466.3	875.0	152.9	133.0	5502.0	5711
3.25	38.1	3456.0	1725.1	5181.1	887.1	164.6	146.7	6239.7	6469
3.5	41	3850.3	1851.7	5702.0	1014.0	180.3	159.7	6903.6	7149
3.75	44	4268.9	2003.7	6272.6	1100.7	190.6	169.3	7570.7	7831
4.0	47.1	4612.0	2177.3	6789.3	1096.9	208.9	183.9	8100.3	8380
4.25	49.1	5203.1	2304.0	7507.1	1282.1	218.1	193.0	9012.0	9305
4.5	52.4	5700.1	2447.4	8147.6	1362.7	235.6	209.1	9752.0	10066
4.75	56	6208.3	2666.7	8875.0	1388.4	245.4	220.3	10510.0	10840
5.0	57.4	6535.3	2772.9	9308.1	1517.0	258.7	227.0	11088.1	11466

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	487
0.5	5.9	585.7	358.6	944.3	210.3	74.1	9.0	1230.0	1252
0.75	8.1	1049.1	641.4	1690.6	321.3	115.3	14.0	2128.0	2159
1	11.0	1595.1	899.0	2494.1	460.9	157.9	18.3	3114.4	3154
1.25	16.4	2298.3	1198.6	3496.9	613.0	200.4	23.4	4311.6	4364
1.5	17.6	3006.1	1461.0	4467.1	749.4	248.7	29.9	5466.7	5528
1.75	21.0	3811.1	1731.7	5542.9	886.0	284.6	33.0	6714.7	6785
2	24.3	4682.6	2058.4	6741.0	978.3	332.1	39.4	8052.6	8134
2.25	26.9	5657.6	2378.1	8035.7	1111.3	371.3	44.7	9519.0	9613
2.5	30.9	6693.4	2695.1	9388.6	1251.6	421.4	50.4	11062.4	11169
2.75	32.4	7784.1	2856.3	10640.4	1509.3	467.0	55.0	12617.9	12736
3	34.9	8879.3	3036.0	11915.3	1653.3	499.3	59.1	14069.1	14195
3.25	38.9	10035.6	3436.6	13472.1	1649.0	541.9	64.3	15663.9	15800
3.5	41.4	11219.3	3668.3	14887.6	1893.3	581.4	69.4	17363.7	17510
3.75	44.3	12461.6	3896.3	16357.9	2042.3	618.6	75.0	19020.1	19178
4	47.0	13948.4	4247.0	18195.4	2033.9	652.4	78.0	20882.9	21068
4.25	50.9	15194.3	4425.0	19619.3	2355.1	699.7	86.1	22675.1	22854
4.5	53.0	16619.7	4737.6	21357.3	2503.4	731.9	88.3	24593.6	24781
4.75	56.3	18231.0	5199.1	23430.1	2554.7	779.7	94.6	26765.9	26965
5	57.1	19832.3	5392.6	25224.9	2708.3	815.1	96.3	28749.3	28952

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 = 2,350 and c = -1,130, with R^2 = 0.9968.

For Linux m = 6,000 and c = -3,000, with R^2 = 0.9963.

The approximate throughput rates for Windows and Linux were respectively 500,000 and 200,000 keys per second. The sort input time (T_SI) on Linux is approximately three times that of Windows.

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

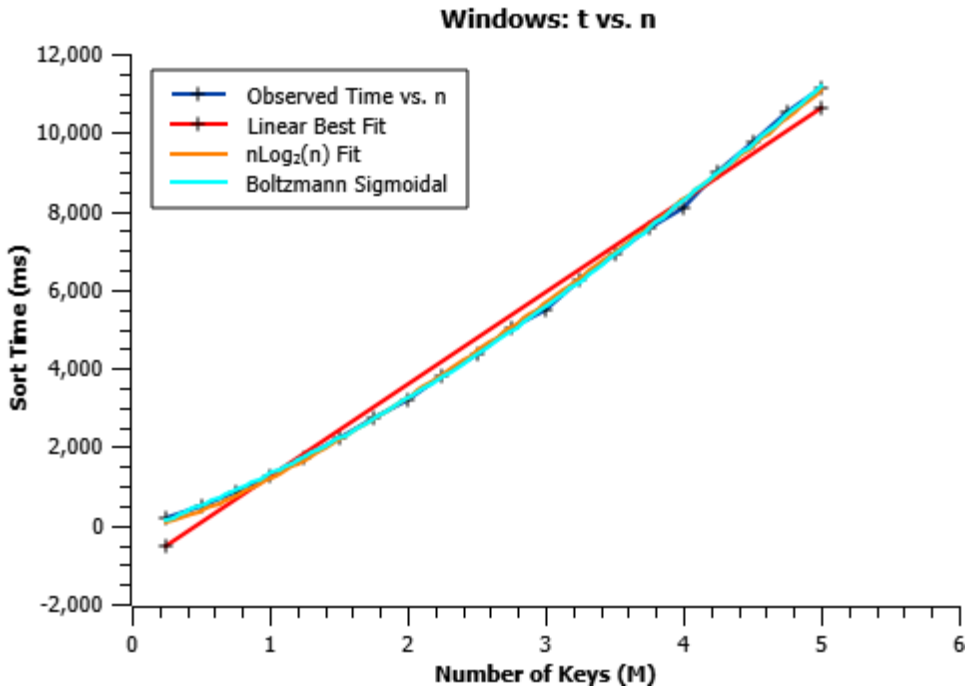
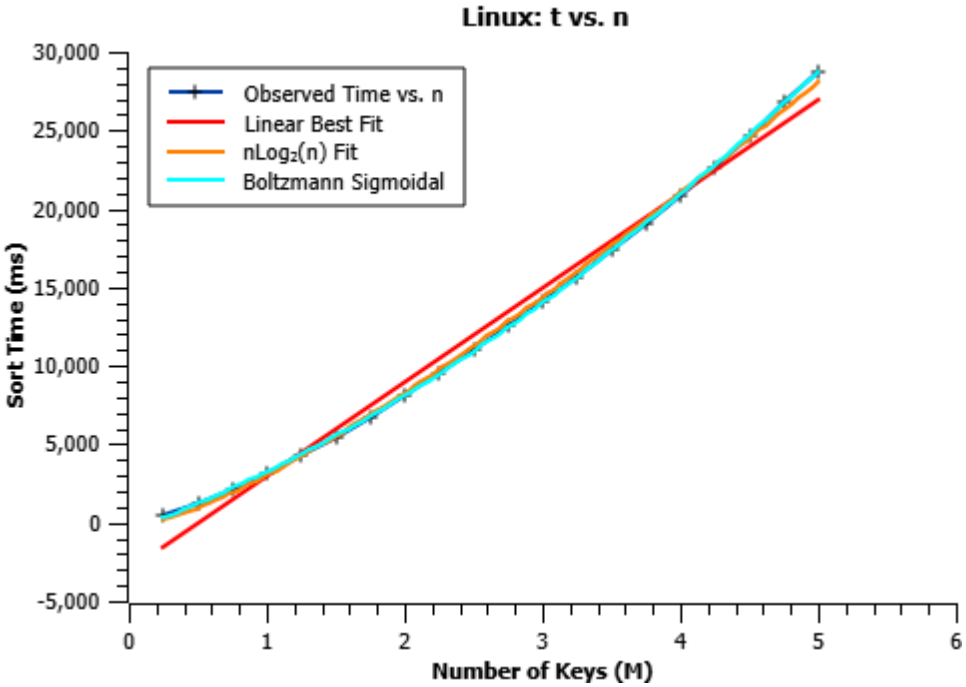


Figure 3. best fit plots for t vs. n on Linux



The plots show a typical logarithmic or sigmoidal deviation from the linear approximation. Sort algorithms based on merge typically show time complexity of $nLog_2(n)$, 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 = 450 and k = 6.5, with $R^2 = 0.9998$.

For Linux m = 1,200 and k = 6.5, with $R^2 = 0.9997$.

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 = -4,100, t_2 = 26,850, n_0 = 5.0 and dn = 2.6

matches with $R^2 = 0.9999$.

For Linux t_1 = -12,000, t_2 = 87,600, n_0 = 6.1 and dn = 3.0

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 v1.14.0 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	2.1	5.0	0.0	5.0	0.0	3.6	4.0	12.3	39
0.5	5	12.6	0.0	12.6	0.0	7.0	14.7	20.7	63
0.75	8.1	24.0	0.0	24.0	0.0	10.9	25.1	36.3	95
1.0	14.7	41.4	0.0	41.4	0.0	15.3	48.3	57.9	147
1.25	14.0	58.1	0.0	58.1	0.0	18.6	48.6	79.4	171
1.5	17.1	76.9	0.0	76.9	0.0	22.7	60.3	102.3	210
1.75	20.0	104.9	0.0	104.9	0.0	26.4	71.3	133.6	256
2.0	22.9	134.6	0.0	134.6	0.0	33.0	82.4	170.1	308
2.25	26.0	173.4	0.0	173.4	0.0	38.4	93.9	216.1	371
2.5	28.6	218.7	0.0	218.7	0.0	43.6	107.9	266.1	439
2.75	31.7	255.7	0.0	255.7	0.0	46.3	119.0	305.7	495
3.0	34.3	311.4	0.0	311.4	0.0	54.1	128.7	368.0	570
3.25	37.3	368.1	0.0	368.1	0.0	53.0	139.1	424.0	640
3.5	40.7	428.9	0.0	428.9	0.0	53.6	149.3	485.6	715
3.75	44.4	495.4	0.0	495.4	0.0	59.0	159.3	558.1	803
4.0	48.6	557.7	0.0	557.7	0.0	59.9	170.4	622.6	884
4.25	48.7	610.3	0.0	610.3	0.0	64.0	184.1	680.1	958
4.5	52.3	693.1	0.0	693.1	0.0	69.9	195.9	767.1	1060
4.75	55.7	766.4	0.0	766.4	0.0	74.0	207.1	842.3	1152
5.0	60.7	839.6	0.0	839.6	0.0	78.1	211.4	919.4	1239

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	3.0	6.1	0.0	6.1	0.0	6.0	5.0	12.9	27
0.5	7	13.1	0.0	13.1	0.0	11.1	9.1	25.0	48
0.75	9.1	21.6	0.0	21.6	0.0	16.3	14.4	38.6	71
1	12.1	33.0	0.0	33.0	0.0	21.3	19.1	54.6	97
1.25	14.6	44.3	0.0	44.3	0.0	28.9	24.1	73.6	124
1.5	17.1	54.6	0.0	54.6	0.0	35.9	28.4	91.3	150
1.75	20.1	70.4	0.0	70.4	0.0	42.0	32.9	113.3	183
2	24.0	86.6	0.0	86.6	0.0	48.9	38.7	136.0	218
2.25	29.1	101.9	0.0	101.9	0.0	53.6	44.3	156.3	252
2.5	31.0	121.6	0.0	121.6	0.0	59.6	49.7	181.7	286
2.75	34.4	137.0	0.0	137.0	0.0	64.4	55.0	202.1	320
3	40.6	155.3	0.0	155.3	0.0	71.6	58.4	227.6	352
3.25	38.4	176.3	0.0	176.3	0.0	74.9	64.7	251.9	385
3.5	43.9	204.3	0.0	204.3	0.0	80.4	71.7	285.1	435
3.75	50.6	227.6	0.0	227.6	0.0	85.6	77.0	313.6	478
4	55.0	252.0	0.0	252.0	0.0	92.3	77.7	345.0	512
4.25	52.0	270.0	0.0	270.0	0.0	95.0	81.6	365.1	535
4.5	55.6	294.0	0.0	294.0	0.0	99.6	86.7	394.1	573
4.75	57.9	327.6	0.0	327.6	0.0	105.0	90.7	433.1	620
5	58.0	361.3	0.0	361.3	0.0	112.3	93.7	474.3	665

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. The sort input time (T_SI) on Linux is approximately half that of the Windows runs while in the random keys tests it was three times.

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 = 200 and c = -150, with R^2 = 0.9975.

For Linux m = 100 and c = -45, with R^2 = 0.9958.

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

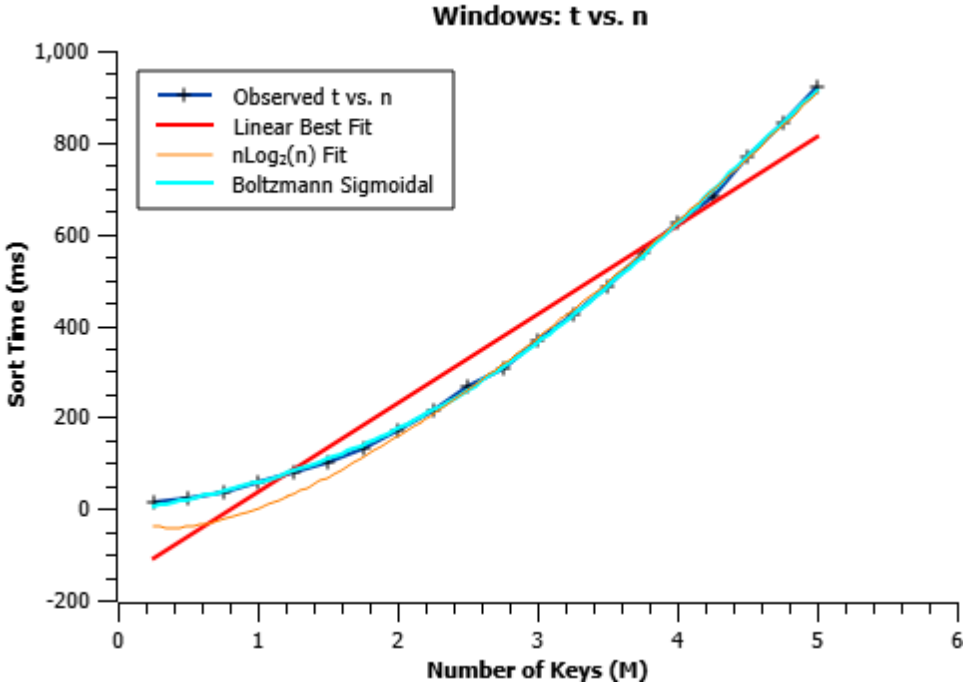
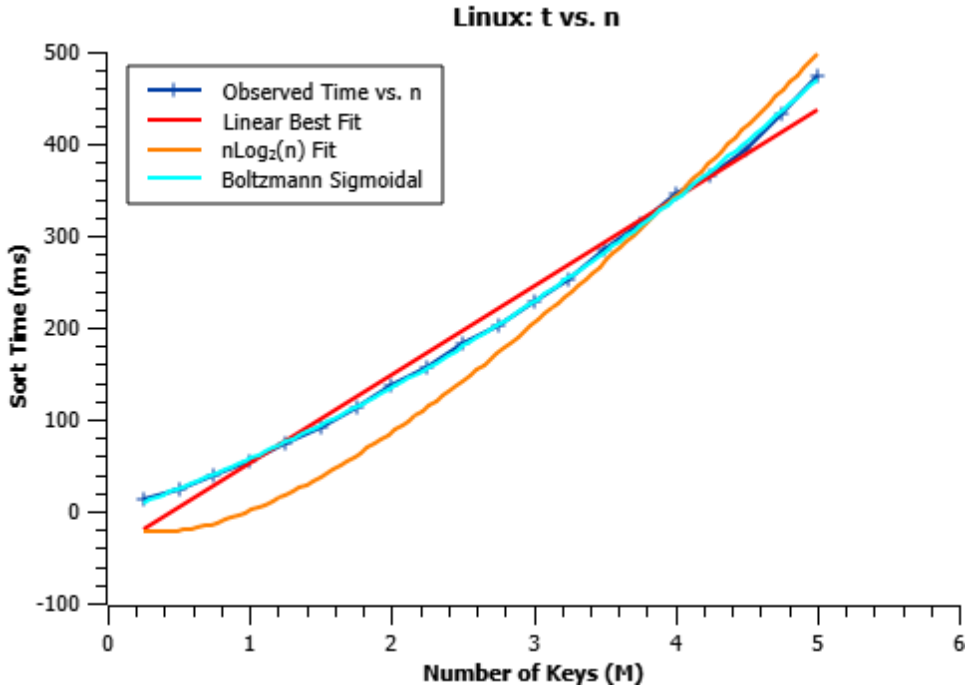


Figure 5. best fit plots for t vs. n on Linux



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 top and tail sequence. The release x64 build v1.14.0 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	3.1	1024.9	147.3	1172.1	8.7	3.4	4.0	1187.3	1217
0.5	5	1756.3	1779.7	3536.0	23.7	8.0	15.0	3574.7	3618
0.75	8.1	2472.4	4182.1	6654.6	41.3	12.3	27.4	6714.9	6775
1.0	14.0	3226.9	7145.6	10372.4	40.9	15.0	37.6	10432.0	10510
1.25	14.1	3946.7	10816.9	14763.6	63.7	23.4	52.6	14858.1	14954
1.5	17.1	4726.9	14803.4	19530.3	77.9	26.3	61.9	19640.4	19750
1.75	20.1	5467.0	19098.9	24565.9	82.1	27.6	72.9	24683.7	24809
2.0	22.9	6253.4	23676.0	29929.4	105.0	31.6	85.9	30075.0	30218
2.25	32.6	6945.7	28951.4	35897.1	112.6	37.6	100.4	36052.1	36221
2.5	36.3	7762.6	34090.6	41853.1	117.3	46.9	111.3	42022.1	42206
2.75	31.6	8535.9	39759.4	48295.3	129.4	46.7	122.9	48479.6	48665
3.0	43.0	9401.1	45335.6	54736.7	150.7	51.0	134.4	54946.7	55159
3.25	37.9	10217.3	51775.7	61993.0	153.6	54.0	151.3	62207.4	62434
3.5	51.4	11083.9	58237.7	69321.6	204.4	56.1	163.9	69588.9	69845
3.75	45.3	12200.9	66724.9	78925.7	192.4	70.0	201.1	79196.0	79485
4.0	58.9	12928.0	72524.0	85452.0	221.3	68.1	193.0	85750.9	86047
4.25	63.3	13934.4	81591.3	95525.7	247.7	80.9	265.6	95864.4	96238
4.5	53.7	14845.7	90928.3	105774.0	249.1	85.1	251.6	106115	106471
4.75	55.9	15791.0	94125.0	109916.0	269.4	76.0	223.3	110270	110609
5.0	58.1	16707.1	101080	117787.1	276.1	85.9	232.0	118158	118505

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	3.9	1150.6	1292.9	2443.4	21.1	10.0	5.1	2475.7	2491
0.5	8.6	2216.1	6152.7	8368.9	50.4	20.0	9.9	8440.1	8469
0.75	13.3	3263.6	12723.6	15987.1	85.6	31.3	14.7	16106.6	16148
1	18.4	4354.1	21070.3	25424.4	104.9	42.1	20.3	25572.7	25630
1.25	28.0	5498.1	29892.1	35390.3	143.6	49.6	24.4	35584.4	35656
1.5	28.7	6663.0	40434.1	47097.1	173.0	60.3	30.9	47331.6	47413
1.75	23.4	7974.0	35660.4	43634.4	132.3	64.4	34.7	43832.1	43909
2	25.3	9256.6	43617.3	52873.9	157.3	81.7	39.6	53114.1	53200
2.25	27.7	10519.0	50462.7	60981.7	162.4	59.6	43.3	61204.7	61294
2.5	34.3	11805.1	59591.7	71396.9	181.1	67.1	50.6	71646.0	71752
2.75	36.9	13280.1	68364.9	81645.0	198.9	73.0	53.6	81918.0	82029
3	36.1	14759.0	77618.7	92377.7	216.6	76.3	57.9	92671.6	92789
3.25	44.1	16250.0	88291.6	104541.6	233.9	84.0	63.9	104860.7	104993
3.5	46.1	17746.9	100056.1	117803.0	281.6	91.9	72.7	118177.6	118326
3.75	55.0	19353.9	110330.9	129684.7	269.7	111.9	75.9	130067.7	130232
4	49.9	21068.9	120031.0	141099.9	316.7	121.9	78.6	141535.7	141700
4.25	62.6	22705.7	132871.3	155577.0	341.7	128.6	86.1	156048.4	156253
4.5	62.4	24444.7	145625.1	170069.9	319.7	141.7	90.7	170532.6	170726
4.75	70.1	26258.0	156918.7	183176.7	380.7	158.7	96.0	183717.1	183926
5	75.7	28074.4	166480.6	194555.0	391.0	162.4	98.4	195109.6	195327

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 = 25,500 and c = -16,000, with $R^2 = 0.9935$.

For Linux m = 40,500 and c = -20,000, with $R^2 = 0.9944$.

The approximate throughput rates for Windows and Linux were respectively 60,000 and 35,000 keys per second. The sort input time (T_SI) and the pre-emptive merge time on Linux is approximately twice that of Windows.

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

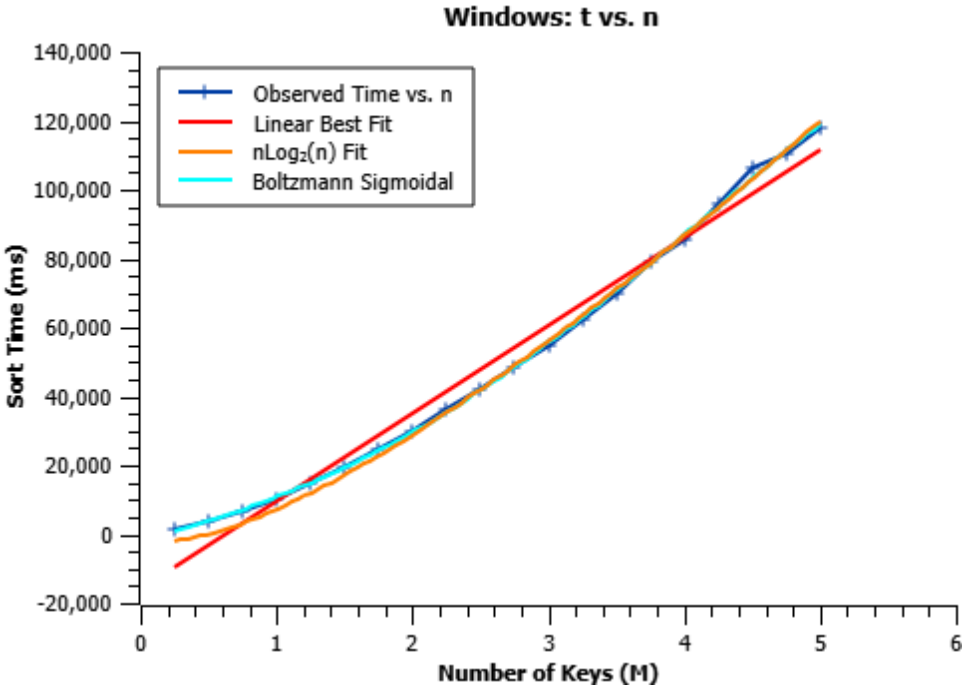
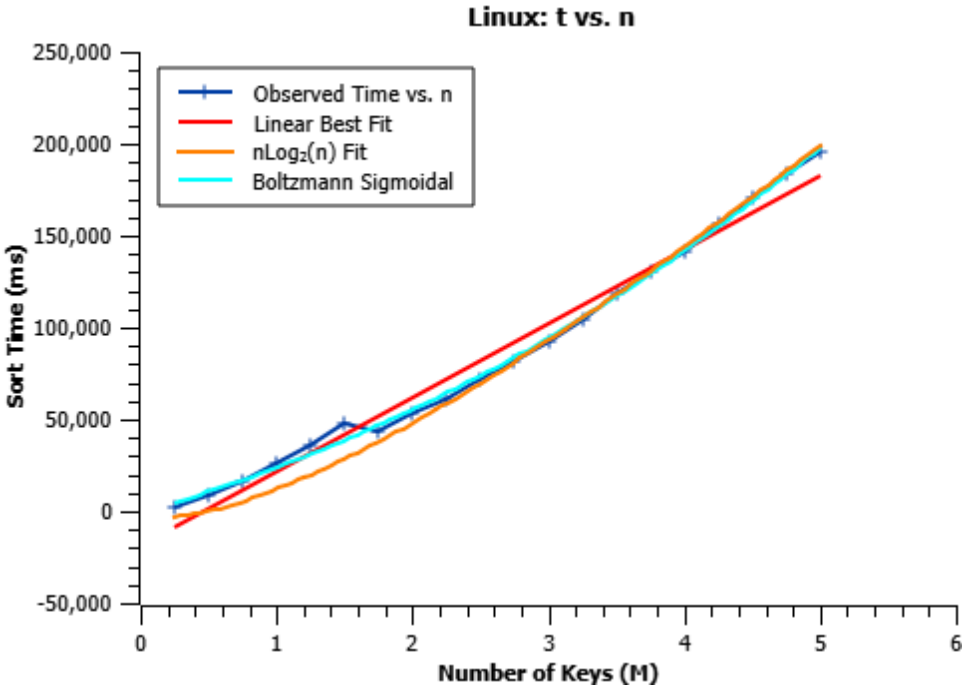


Figure 7. best fit plots for t vs. n on Linux



On the Linux plot a curious inflection is obvious at 1.5 million keys. This inflection is not apparent on the Windows plot.

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 = 7,000 and k = 2.0, with $R^2 = 0.9989$.

For Linux m = 12,000 and k = 2.0, with $R^2 = 0.9949$.

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 = -20,000, t_2 = 200,000, n_0 = 4.2 and *dn* = 1.7 matches with $R^2 = 0.9998$.

For Linux t_1 = -81,000, t_2 = 1,000,000, n_0 = 8.4 and *dn* = 3.4 matches with $R^2 = 0.9992$.

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	227	325	39	394	1217
0.5	1061	541	665	63	868	3618
0.75	1696	912	994	95	1310	6775
1.0	2357	1339	1374	147	1882	10510
1.25	3069	1834	1737	171	2324	14954
1.5	3773	2342	2086	210	2994	19750
1.75	4581	2837	2489	256	3406	24809
2.0	5200	3318	2878	308	4078	30218
2.25	6162	3977	3310	371	4525	36221
2.5	6956	4567	3617	439	5160	42206
2.75	7566	5197	3931	495	5570	48665
3.0	8267	5711	4329	570	6322	55159
3.25	9086	6469	4729	640	6675	62434
3.5	9986	7149	5129	715	7474	69845
3.75	10581	7831	5541	803	7646	79485
4.0	11264	8380	5962	884	8671	86047
4.25	12374	9305	6415	958	8865	96238
4.5	13069	10066	6827	1060	9729	106471
4.75	13893	10840	7249	1152	10016	110609
5.0	14784	11466	7488	1239	10898	118505

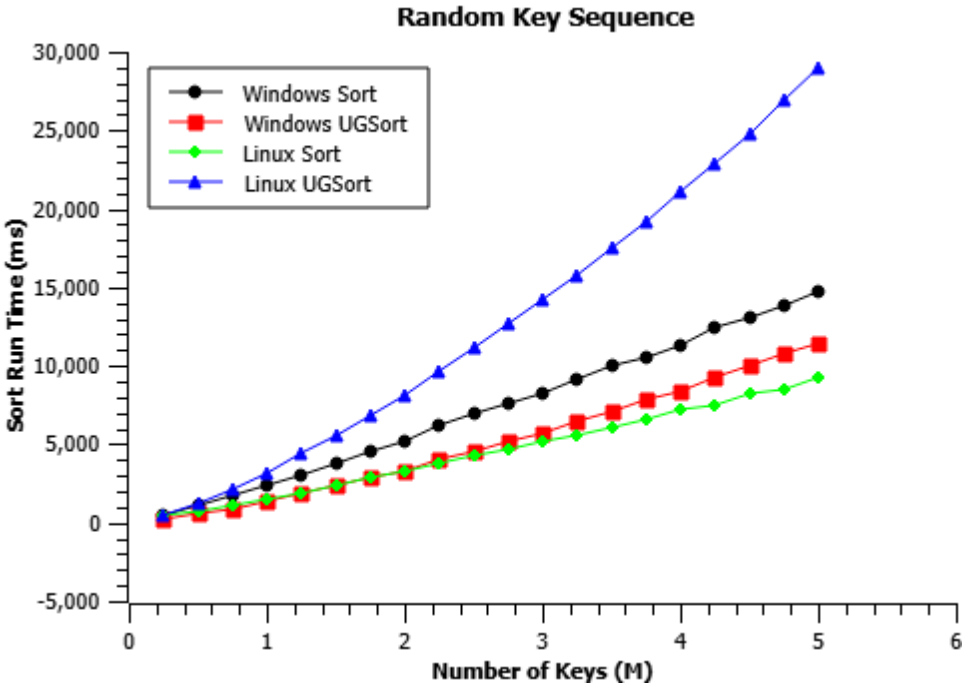
Linux Results.

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	487	239	27	298	2491
0.5	685	1252	304	48	371	8469
0.75	1072	2159	467	71	570	16148
1.0	1457	3154	624	97	750	25630
1.25	1876	4364	803	124	959	35656
1.5	2356	5528	1024	150	1229	47413
1.75	2839	6785	1238	183	1456	43909
2.0	3304	8134	1435	218	1753	53200
2.25	3797	9613	1685	252	2011	61294
2.5	4239	11169	1871	286	2202	71752
2.75	4732	12736	2148	320	2465	82029
3.0	5144	14195	2325	352	2738	92789
3.25	5619	15800	2535	385	2983	104993
3.5	6111	17510	2730	435	3234	118326
3.75	6636	19178	2907	478	3451	130232
4.0	7207	21068	3218	512	3763	141700
4.25	7519	22854	3383	535	3963	156253
4.5	8211	24781	3560	573	4269	170726
4.75	8484	26965	3850	620	4516	183926
5.0	9261	28952	4055	665	4697	195327

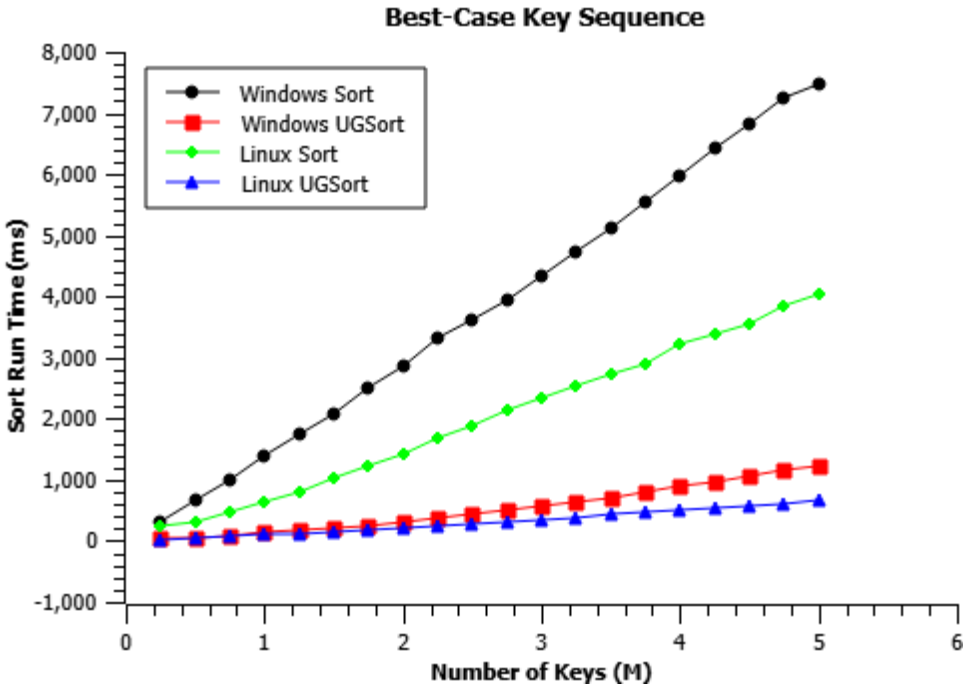
Observations and Analysis

Figure 8. comparison plots for random key sequence



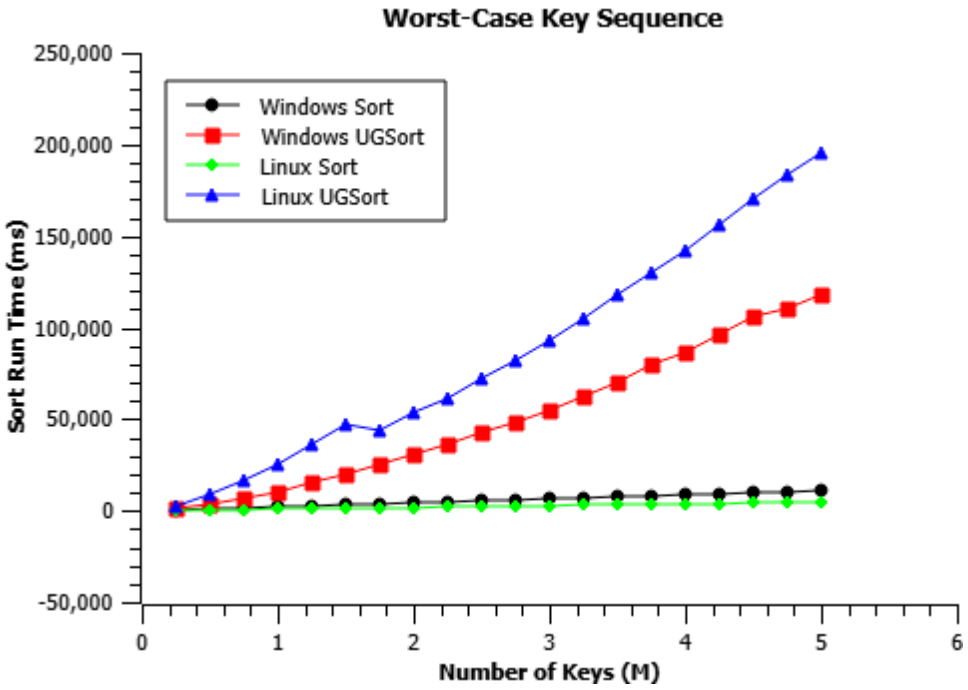
UGSort performed well on Windows, outperforming the native Sort utility. On Linux the UGSort implementation performed poorly compared to the native Sort utility and compared to that on Windows.

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.

The cause of the inflection in the UGSort on Linux plot is unknown but, may be related to the minimum value for the maximum number of double ended queues before pre-emptive merge is triggered (100).

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).

FURTHER WORK

There is significant scope for optimisation of the implementation of the application. Parallel pre-emptive and final merge should provide some improvements and better memory and object management should further improve sorting times.

A theoretical study of the UGSort algorithm would underpin this study. Such a study should resolve a relationship between sorting times and the degree of presortedness or sequence spoiling noted in the 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