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.

Revised 05/11/2023 for v1.16 of the application using tail suppression for pre-emptive merges.

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 v1.16.1.

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.

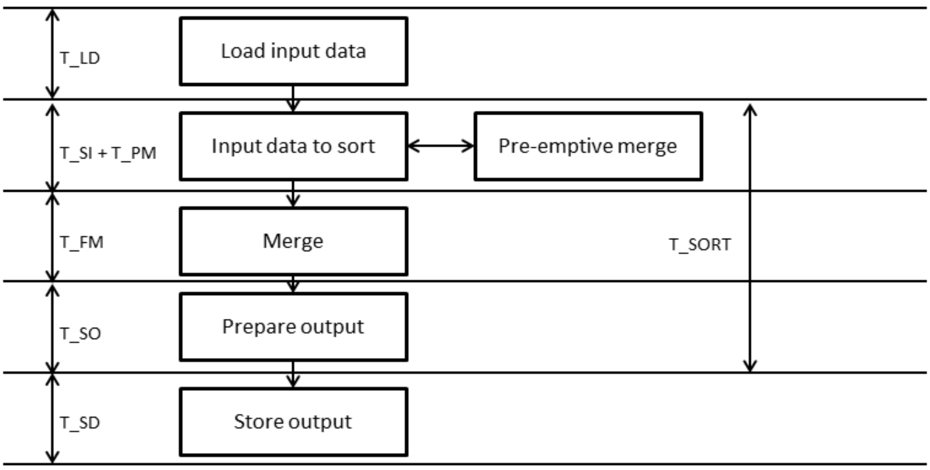
STUDIES

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. From v1.16.1 this time is accumulated with microsecond resolution but remains being reported as milliseconds.
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. These tests were performed using v1.15.0 of the application.

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 v1.16.1 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	25.1	171.6	196.7	46.7	9.0	4.0	259.3	287
0.50	5	50.0	202.6	252.6	117.0	23.4	14.7	400.1	443
0.75	8.0	75.9	217.6	293.4	190.6	38.3	32.0	528.4	593
1.00	11.1	104.0	250.9	354.9	260.3	51.3	38.0	669.9	745
1.25	14.0	135.0	255.6	390.6	343.1	66.3	53.3	806.3	902
1.50	17.1	165.9	292.0	457.9	436.9	81.3	60.1	983.1	1091
1.75	20.0	196.9	306.3	503.1	519.3	94.1	71.3	1119.3	1242
2.00	23.4	223.3	344.1	567.4	590.0	107.9	86.7	1268.4	1413
2.25	26.0	258.0	370.1	628.1	683.6	122.9	98.6	1439.0	1600
2.50	29.3	289.0	403.6	692.6	811.6	137.9	107.4	1648.7	1823
2.75	32.0	319.3	428.3	747.6	903.3	153.6	124.7	1810.6	2007
3.00	34.4	348.4	461.6	810.0	969.4	165.9	129.0	1951.9	2156
3.25	38.3	386.4	493.3	879.7	1056.3	180.4	145.7	2120.7	2348
3.50	40.7	420.3	519.7	940.0	1159.4	196.7	155.9	2300.4	2542
3.75	43.7	444.4	547.3	991.7	1221.0	208.9	168.1	2422.9	2682
4.00	45.6	482.7	594.9	1077.6	1303.7	225.6	180.1	2608.6	2882
4.25	50.1	516.1	625.9	1142.0	1400.4	240.0	189.4	2789.3	3080
4.50	52.6	553.7	670.4	1224.1	1489.9	251.0	206.6	2971.0	3282
4.75	55.0	583.7	701.9	1285.6	1569.9	267.4	215.0	3129.1	3453
5.00	57.4	615.1	737.1	1352.3	1674.3	277.7	221.7	3310.3	3643

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	3.1	49.6	256.1	305.7	79.9	30.7	4.0	417.4	430
0.50	6.3	102.3	288.6	390.9	175.0	67.0	9.0	633.9	657
0.75	10.1	154.9	327.6	482.4	292.9	105.1	13.4	881.3	915
1.00	12.0	209.7	373.1	582.9	394.9	141.7	18.0	1120.6	1164
1.25	18.6	268.6	405.3	673.9	520.7	188.9	23.9	1384.0	1443
1.50	24.0	327.1	453.4	780.6	662.4	231.3	29.0	1675.3	1747
1.75	29.7	385.1	495.0	880.1	783.4	276.9	33.6	1941.4	2027
2.00	34.4	449.9	536.4	986.3	923.4	321.1	39.4	2232.1	2331
2.25	37.0	513.4	581.6	1095.0	1058.7	366.7	43.3	2521.3	2629
2.50	45.4	573.6	630.9	1204.4	1320.1	408.7	48.4	2934.3	3059
2.75	43.4	640.4	699.4	1339.9	1460.1	444.1	54.0	3245.1	3375
3.00	38.9	690.7	734.6	1425.3	1483.0	467.0	57.9	3376.3	3508
3.25	46.4	756.3	799.9	1556.1	1620.3	500.6	65.7	3678.0	3835
3.50	63.4	820.6	839.1	1659.7	1781.6	550.7	72.6	3993.6	4180
3.75	70.7	887.9	894.9	1782.7	1924.3	594.3	75.3	4302.6	4500
4.00	69.1	954.9	945.6	1900.4	2027.4	633.7	78.6	4562.3	4760
4.25	74.0	1021.1	1008.4	2029.6	2186.1	680.6	86.1	4901.4	5113
4.50	76.7	1069.6	1055.4	2125.0	2313.9	720.1	89.0	5160.1	5381
4.75	81.1	1155.9	1130.9	2286.7	2506.1	769.7	95.7	5563.9	5800
5.00	82.6	1223.4	1183.3	2406.7	2691.1	818.9	98.9	5922.1	6159

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 = 648 and c = 25, with R^2 = 0.9997.

For Linux m = 1,157 and c = -16, with R^2 = 0.9996.

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

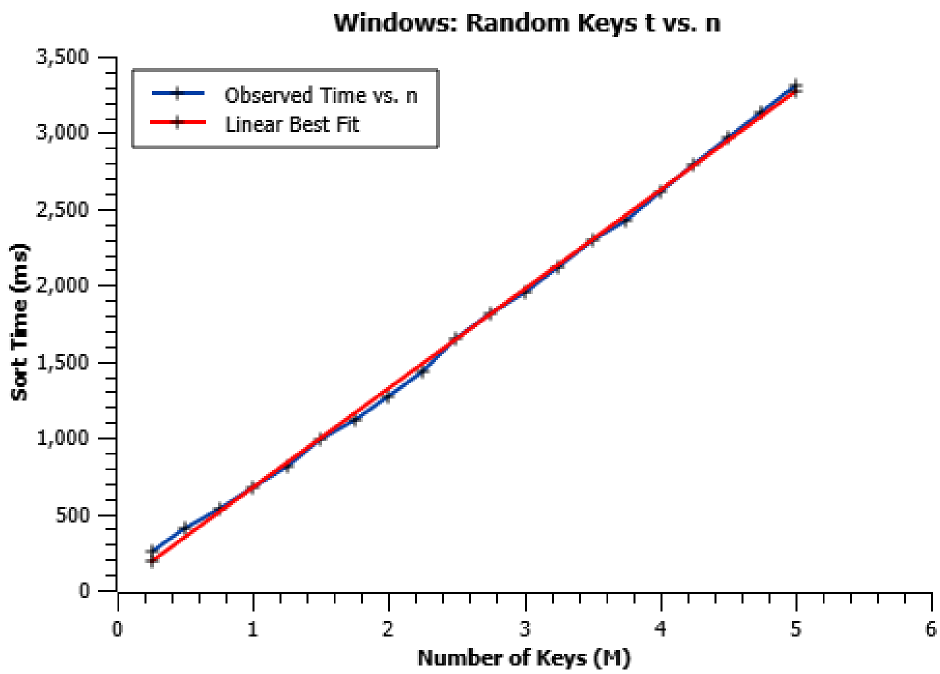
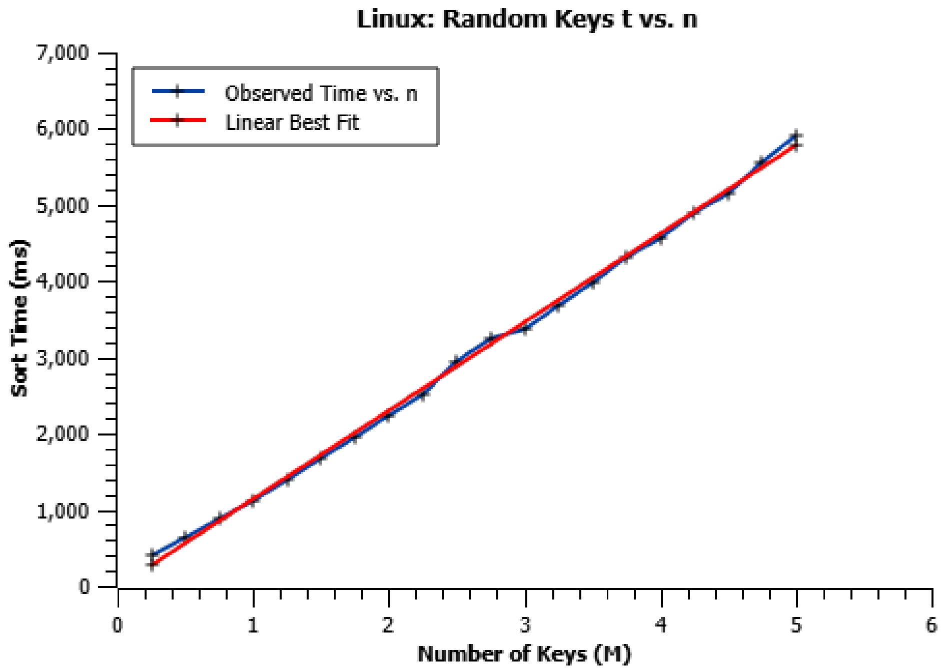


Figure 3. best fit plots for t vs. n on Linux



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 v1.16.1 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	3.4	4.1	11.6	39
0.50	5.1	13.1	0.0	13.1	0.0	7.0	14.9	23.9	67
0.75	8.0	24.4	0.0	24.4	0.0	11.0	25.3	42.3	101
1.00	11.0	38.6	0.0	38.6	0.0	14.9	36.7	60.9	135
1.25	14.0	58.6	0.0	58.6	0.0	19.1	48.9	85.4	179
1.50	17.1	78.4	0.0	78.4	0.0	23.0	60.3	106.6	215
1.75	20.1	104.7	0.0	104.7	0.0	26.1	71.9	132.9	257
2.00	22.9	133.9	0.0	133.9	0.0	30.1	80.9	168.6	306
2.25	26.6	166.7	0.0	166.7	0.0	34.6	93.4	204.3	359
2.50	29.1	204.6	0.0	204.6	0.0	38.9	104.1	248.0	418
2.75	31.7	233.4	0.0	233.4	0.0	42.6	116.4	282.4	470
3.00	34.1	277.1	0.0	277.1	0.0	47.7	125.9	328.9	529
3.25	38.3	328.6	0.0	328.6	0.0	49.3	142.0	380.7	604
3.50	40.9	372.9	0.0	372.9	0.0	53.7	154.4	430.6	670
3.75	43.6	434.0	0.0	434.0	0.0	61.4	164.3	498.0	752
4.00	45.1	485.7	0.0	485.7	0.0	60.3	174.3	552.4	817
4.25	49.4	533.4	0.0	533.4	0.0	65.3	192.7	600.9	892
4.50	52.9	607.1	0.0	607.1	0.0	72.4	200.9	684.3	987
4.75	54.7	657.1	0.0	657.1	0.0	76.9	218.6	741.6	1066
5.00	56.6	744.6	0.0	744.6	0.0	76.0	223.1	824.4	1157

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	7.7	0.0	7.7	0.0	6.4	4.6	14.9	28
0.50	6.3	15.0	0.0	15.0	0.0	11.9	9.7	27.6	51
0.75	9.0	25.4	0.0	25.4	0.0	17.0	14.0	43.0	76
1.00	12.1	37.0	0.0	37.0	0.0	22.0	18.6	59.4	101
1.25	14.4	49.3	0.0	49.3	0.0	30.9	24.3	80.7	133
1.50	17.4	62.1	0.0	62.1	0.0	42.1	28.6	104.9	167
1.75	20.6	75.6	0.0	75.6	0.0	54.0	33.9	130.3	205
2.00	22.1	90.9	0.0	90.9	0.0	65.4	38.9	157.0	239
2.25	29.1	110.0	0.0	110.0	0.0	80.7	43.9	191.6	292
2.50	39.0	128.1	0.0	128.1	0.0	88.4	48.6	217.1	337
2.75	47.6	150.0	0.0	150.0	0.0	102.3	54.1	252.7	394
3.00	59.0	173.9	0.0	173.9	0.0	112.7	61.3	287.0	455
3.25	66.3	191.7	0.0	191.7	0.0	118.1	66.1	310.6	491
3.50	69.3	216.3	0.0	216.3	0.0	125.3	71.0	342.3	532
3.75	69.3	234.0	0.0	234.0	0.0	132.3	75.6	367.0	562
4.00	70.1	260.0	0.0	260.0	0.0	137.7	77.0	398.4	596
4.25	79.1	280.4	0.0	280.4	0.0	145.7	83.4	427.1	644
4.50	83.1	308.1	0.0	308.1	0.0	154.1	88.4	463.0	690
4.75	86.6	334.9	0.0	334.9	0.0	161.7	92.7	497.0	734
5.00	83.6	366.9	0.0	366.9	0.0	169.1	95.7	536.7	774

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 = 170 and c = -126, with R^2 = 0.984.

For Linux m = 112 and c = -50, with R^2 = 0.998.

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

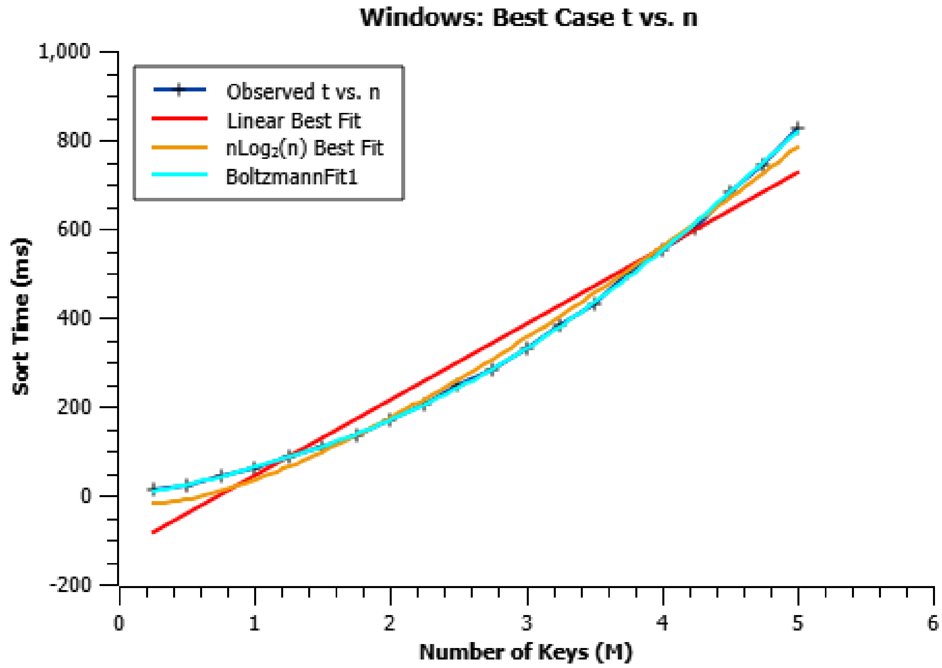
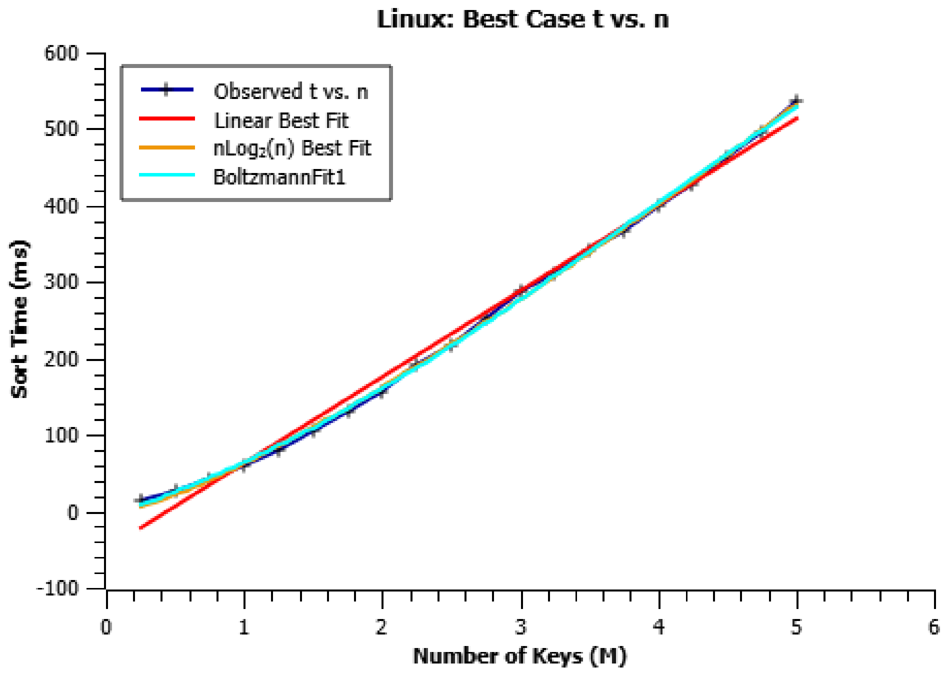


Figure 5. best fit plots for t vs. n on Linux



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 = 53 and k = 1.5, with $R^2 = 0.9972$.

For Linux m = 19 and k = 9.56, with $R^2 = 0.9997$.

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

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

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 = -105, t_2 = 1,975, n_0 = 5.4 and *dn* = 1.8

matches with $R^2 = 0.9998$.

For Linux $t_1 = -153$, $t_2 = 904$, $n_0 = 3.8$ and $dn = 2.1$

matches with $R^2 = 0.9996$.

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 v1.16.1 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	2.0	32.4	540.6	573.0	13.0	4.3	4.0	591.9	618
0.50	5.1	60.6	593.4	654.0	27.9	8.0	14.0	694.6	736
0.75	8.1	89.9	632.0	721.9	45.3	11.9	25.7	784.3	842
1.00	11.0	118.1	672.7	790.9	59.7	16.6	39.4	870.1	947
1.25	14.4	153.6	797.1	950.7	76.1	20.9	48.3	1054.9	1144
1.50	18.6	186.3	873.6	1059.9	90.1	24.3	59.9	1180.6	1287
1.75	20.0	216.9	912.0	1128.9	101.6	30.1	72.0	1266.4	1390
2.00	22.3	249.9	967.0	1216.9	122.1	32.4	81.9	1377.4	1515
2.25	25.9	283.0	1038.7	1321.7	135.1	41.3	94.7	1503.9	1660
2.50	28.9	320.4	1124.6	1445.0	164.0	41.1	106.1	1656.0	1827
2.75	31.4	350.9	1187.0	1537.9	178.0	45.0	120.3	1768.3	1957
3.00	33.3	386.3	1243.6	1629.9	190.0	49.9	130.3	1874.7	2076
3.25	37.0	418.7	1321.4	1740.1	206.9	54.4	146.4	2008.7	2232
3.50	40.0	451.0	1395.7	1846.7	229.0	58.3	157.7	2140.0	2380
3.75	42.9	486.7	1463.4	1950.1	247.1	62.4	170.3	2267.9	2524
4.00	46.4	524.0	1540.1	2064.1	262.1	69.0	179.1	2402.3	2673
4.25	48.6	555.1	1620.6	2175.7	276.7	70.0	195.0	2530.3	2821
4.50	52.7	594.4	1729.1	2323.6	304.1	75.0	202.9	2704.1	3013
4.75	71.9	639.1	1851.1	2490.3	333.7	85.6	210.4	2910.9	3248
5.00	55.9	658.6	1848.9	2507.4	327.7	85.0	220.4	2927.1	3253

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	2.9	62.4	928.4	990.9	16.9	6.9	4.4	1016.1	1029
0.50	5.6	123.6	988.3	1111.9	34.3	13.4	10.0	1160.6	1184
0.75	8.3	182.7	1080.9	1263.6	55.0	18.9	14.6	1338.6	1371
1.00	14.3	246.1	1149.7	1395.9	74.4	24.7	19.4	1496.0	1540
1.25	15.1	307.3	1199.1	1506.4	92.3	32.4	25.3	1632.1	1685
1.50	19.9	372.6	1303.1	1675.7	116.1	43.9	29.7	1836.9	1901
1.75	22.9	437.3	1395.4	1832.7	136.3	64.4	34.3	2034.6	2110
2.00	25.1	502.9	1482.7	1985.6	152.7	86.7	38.4	2226.1	2311
2.25	29.4	569.3	1599.7	2169.0	187.0	94.6	43.9	2452.0	2550
2.50	32.0	630.0	1672.4	2302.4	258.0	107.4	48.7	2668.9	2777
2.75	38.4	706.1	1807.7	2513.9	293.1	123.1	56.6	2931.3	3060
3.00	41.6	764.0	1877.6	2641.6	298.9	124.6	58.7	3066.1	3201
3.25	44.9	826.7	1987.1	2813.9	334.3	138.3	62.9	3287.1	3433
3.50	54.1	895.4	2086.4	2981.9	360.0	149.4	68.9	3492.9	3657
3.75	60.7	967.1	2210.6	3177.7	388.1	158.4	73.7	3725.3	3906
4.00	69.4	1040.6	2331.0	3371.6	418.4	164.6	78.9	3955.7	4152
4.25	82.0	1110.6	2478.3	3588.9	463.4	185.3	92.1	4238.7	4468
4.50	84.3	1197.1	2601.1	3798.3	463.7	193.6	92.6	4456.3	4689
4.75	86.6	1246.9	2659.0	3905.9	487.4	203.9	98.9	4603.4	4846
5.00	82.1	1337.0	2807.1	4144.1	518.9	208.7	100.0	4872.9	5114

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 = 503.8 and c = 403.1, with R^2 = 0.9995.

For Linux m = 825 and c = 658, with R^2 = 0.9995.

The approximate throughput rates for Windows and Linux were respectively 1,400,000 and 900,000 keys per second.

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

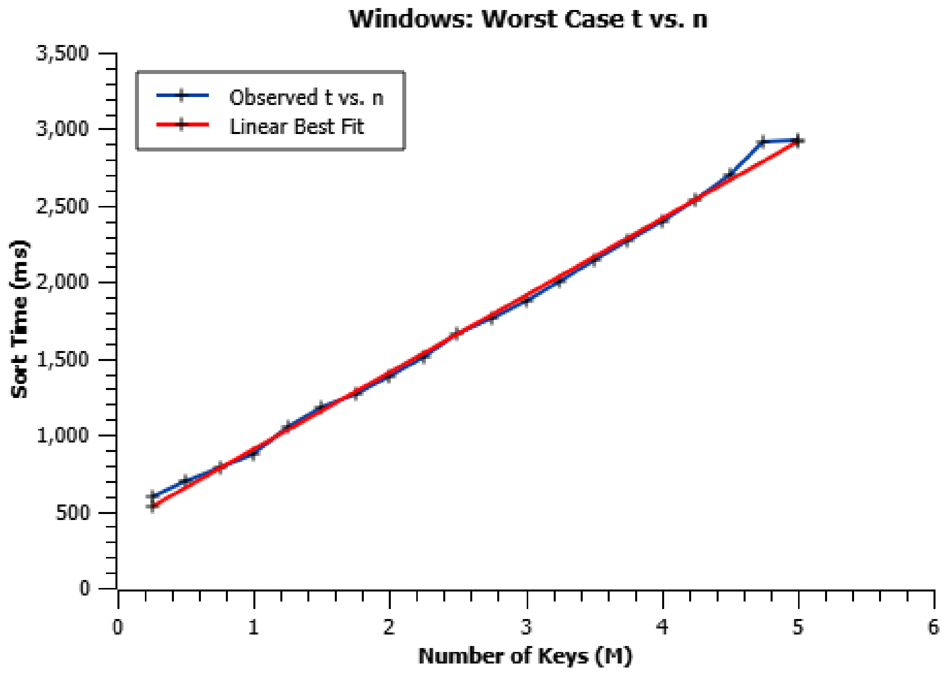
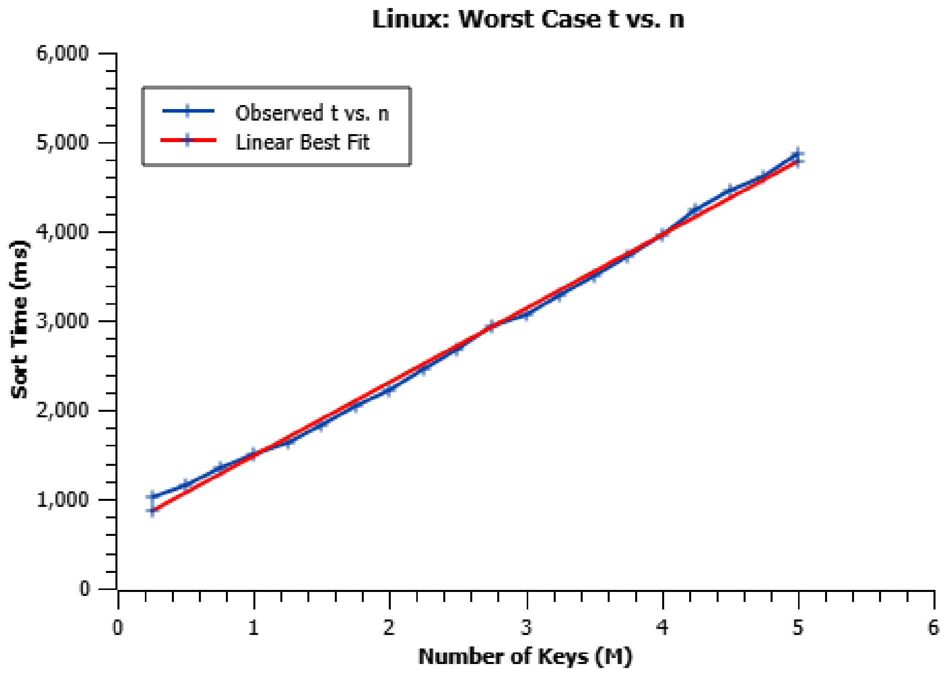


Figure 7. best fit plots for t vs. n on Linux



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	287	325	39	394	618
0.50	1061	443	665	67	868	736
0.75	1696	593	994	101	1310	842
1.00	2357	745	1374	135	1882	947
1.25	3069	902	1737	179	2324	1144
1.50	3773	1091	2086	215	2994	1287
1.75	4581	1242	2489	257	3406	1390
2.00	5200	1413	2878	306	4078	1515
2.25	6162	1600	3310	359	4525	1660
2.50	6956	1823	3617	418	5160	1827
2.75	7566	2007	3931	470	5570	1957
3.00	8267	2156	4329	529	6322	2076
3.25	9086	2348	4729	604	6675	2232
3.50	9986	2542	5129	670	7474	2380
3.75	10581	2682	5541	752	7646	2524
4.00	11264	2882	5962	817	8671	2673
4.25	12374	3080	6415	892	8865	2821
4.50	13069	3282	6827	987	9729	3013
4.75	13893	3453	7249	1066	10016	3248
5.00	14784	3643	7488	1157	10898	3253

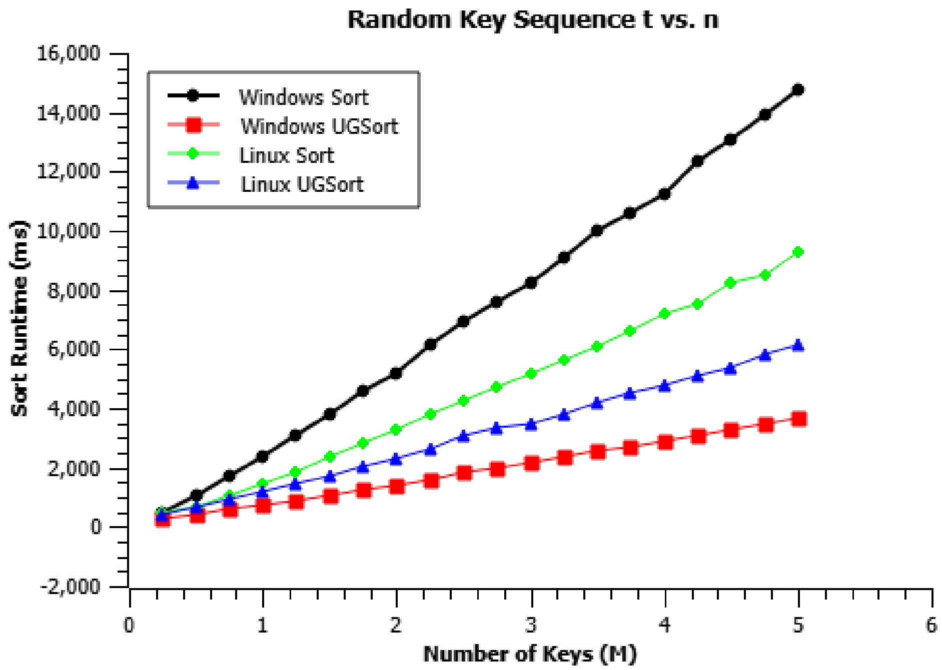
Linux Results.

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	430	239	28	298	1029
0.50	685	657	304	51	371	1184
0.75	1072	915	467	76	570	1371
1.00	1457	1164	624	101	750	1540
1.25	1876	1443	803	133	959	1685
1.50	2356	1747	1024	167	1229	1901
1.75	2839	2027	1238	205	1456	2110
2.00	3304	2331	1435	239	1753	2311
2.25	3797	2629	1685	292	2011	2550
2.50	4239	3059	1871	337	2202	2777
2.75	4732	3375	2148	394	2465	3060
3.00	5144	3508	2325	455	2738	3201
3.25	5619	3835	2535	491	2983	3433
3.50	6111	4180	2730	532	3234	3657
3.75	6636	4500	2907	562	3451	3906
4.00	7207	4760	3218	596	3763	4152
4.25	7519	5113	3383	644	3963	4468
4.50	8211	5381	3560	690	4269	4689
4.75	8484	5800	3850	734	4516	4846
5.00	9261	6159	4055	774	4697	5114

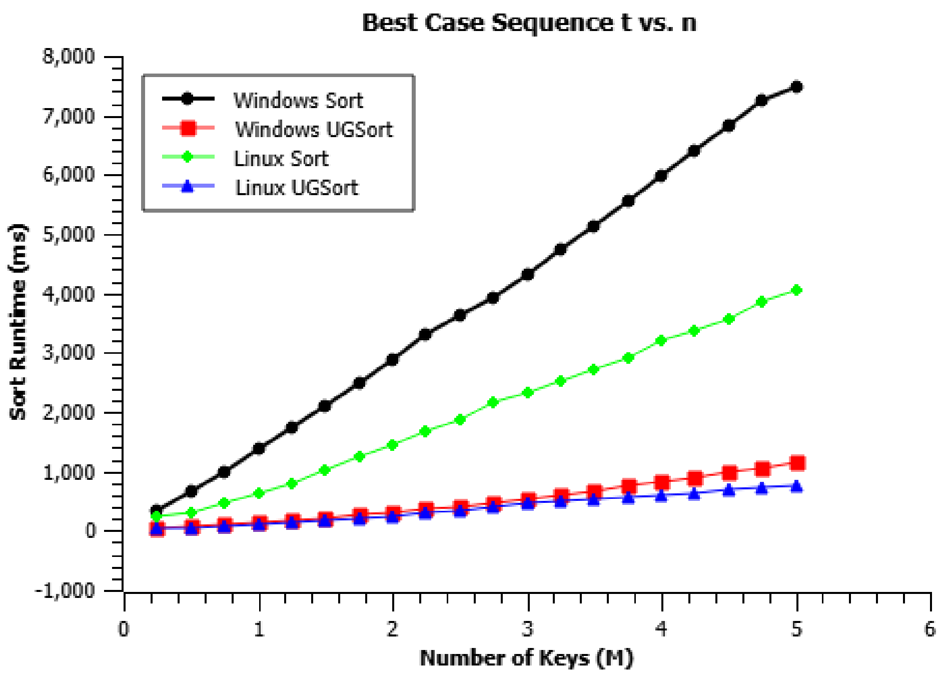
Observations and Analysis

Figure 8. comparison plots for random key sequence



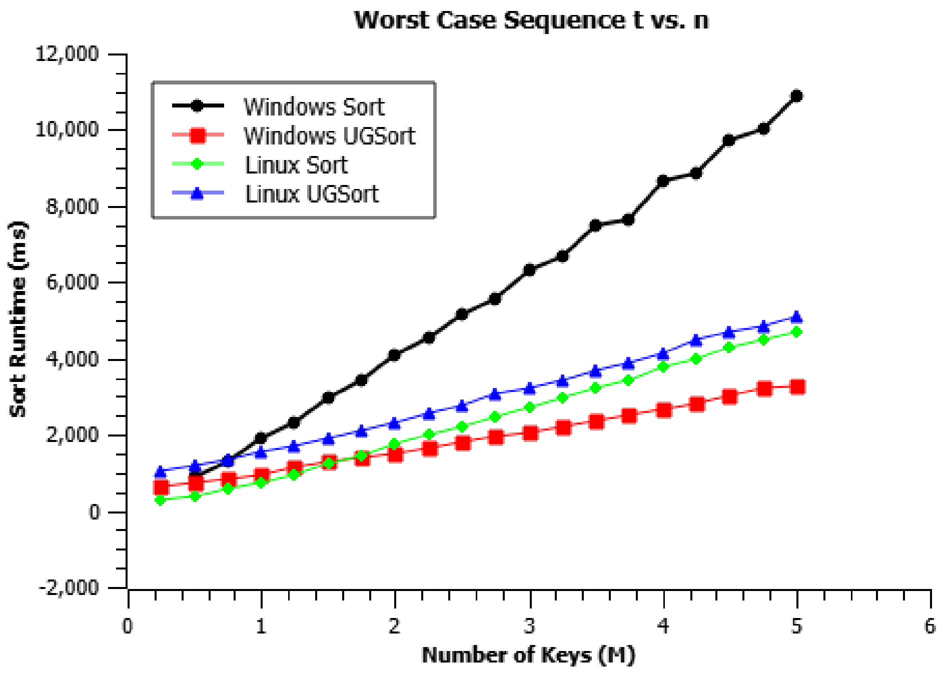
UGSort performed well on both Windows and Linux, outperforming the native Sort utilities by a significant margin.

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 well on this sequence, which is surprising as the sequence was designed to be highly toxic for the UGSort algorithm.

6. Version Comparisons

The charts in this section shows the comparison of the timings for the random and worst case test sets.

Figure 11. comparison plots for random key sequence

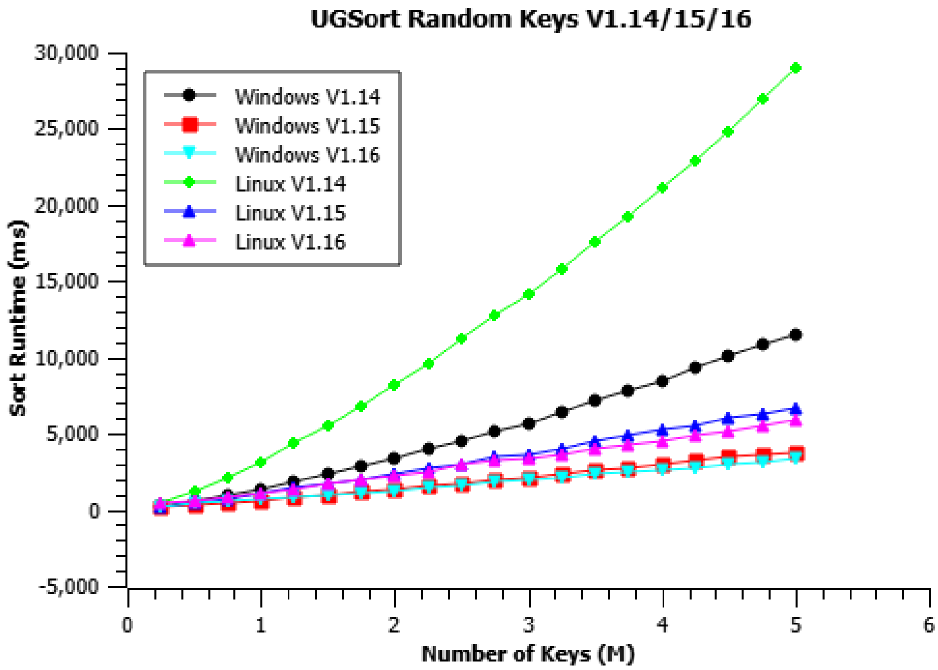
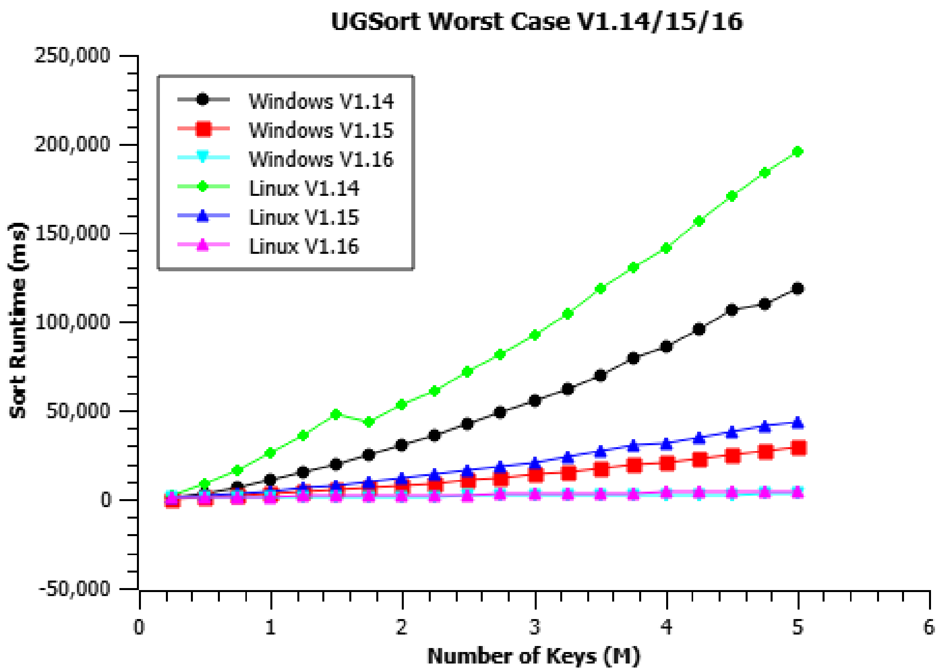


Figure 12. comparison plots for worst case key sequence



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 all test sets, performing far better than the native Sort utilities on Windows and on a par with the Linux sort.

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.

The replacement of the pre-emptive sort pattern of alternate merges with the tail suppression pattern has addressed the apparent weakness of the worst case tests.

FURTHER WORK

A theoretical study of the UGSort algorithm would underpin this study. The apparent O(n) time complexity observed in the random and worst case 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