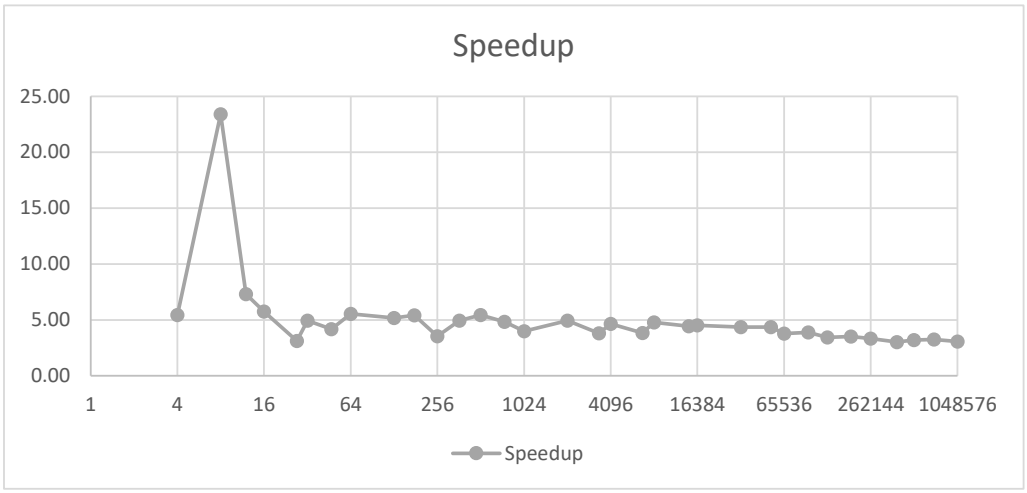
Host environment

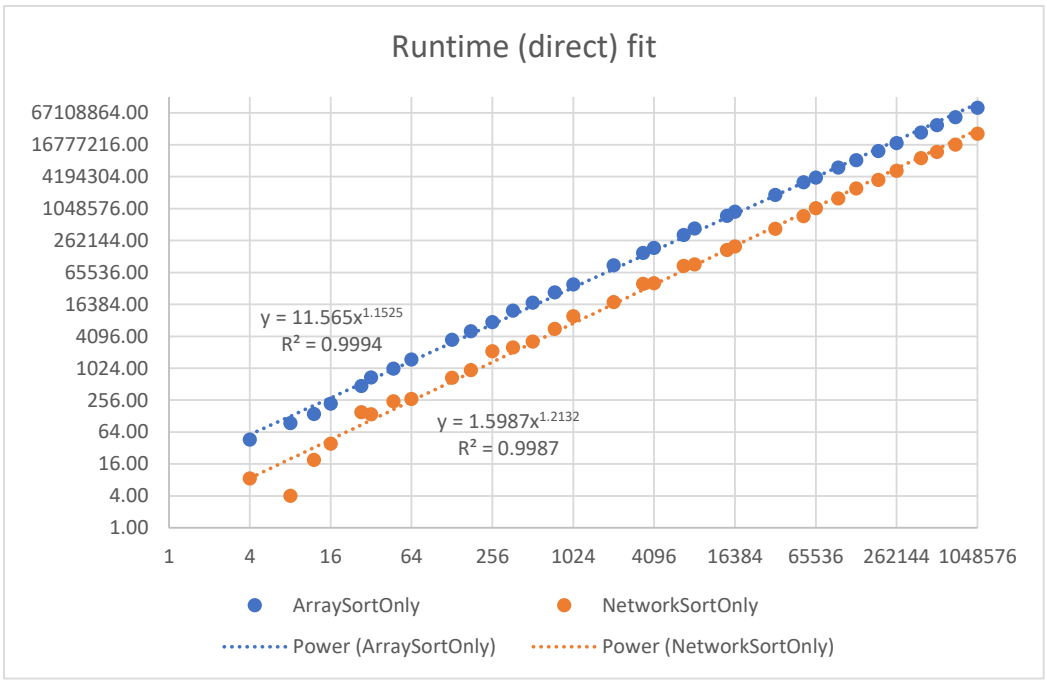
Parameter	Value
BenchmarkDotNetCaption	BenchmarkDotNet
BenchmarkDotNetVersion	0.13.1
OsVersion	Windows 10.0.22000
ProcessorName	Intel Core i7-8650U CPU 1.90GHz (Kaby Lake R)
PhysicalProcessorCount	1
PhysicalCoreCount	4
LogicalCoreCount	8
RuntimeVersion	.NET Core 3.1.21 (CoreCLR 4.700.21.51404, CoreFX 4.700.21.51508)
Architecture	X64
HasAttachedDebugger	FALSE
HasRyuJit	TRUE
Configuration	RELEASE
DotNetSdkVersion	5.0.303
ChronometerFrequency	[Table]
HardwareTimerKind	Unknown

IntBenchmark

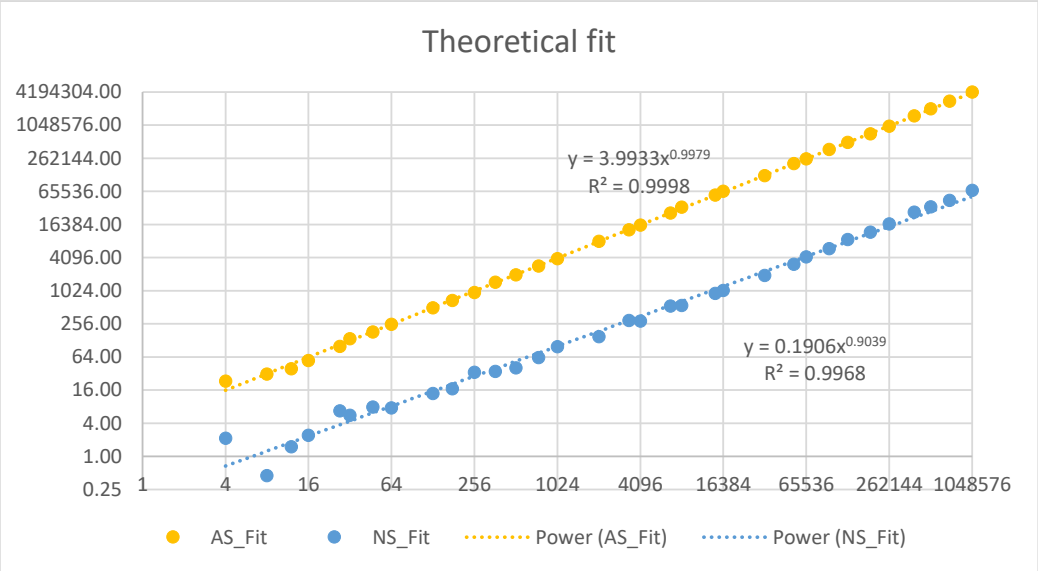
N	ArraySortOnly	NetworkSortOnly	Speedup	AS_Fit	NS_Fit
4	46.56	8.55	5.44	23.28	2.14
8	94.39	4.03	23.41	31.46	0.45
12	140.72	19.26	7.31	39.25	1.50
16	221.54	38.57	5.74	55.39	2.41
27	476.34	152.54	3.12	100.18	6.75
32	687.57	139.27	4.94	137.51	5.57
47	1010.13	242.81	4.16	181.86	7.87
64	1509.62	272.18	5.55	251.60	7.56
128	3518.51	680.44	5.17	502.64	13.89
177	5108.18	945.76	5.40	684.05	16.96
256	7615.99	2148.21	3.55	952.00	33.57
364	12481.21	2528.22	4.94	1467.03	34.93
512	17810.05	3279.07	5.43	1978.89	40.48
748	27608.06	5700.92	4.84	2891.84	62.55
1024	39204.33	9863.05	3.97	3920.43	98.63
2048	89523.85	18121.07	4.94	8138.53	149.76
3389	153672.22	40323.74	3.81	13104.54	293.23
4096	192227.15	41392.70	4.64	16018.93	287.45
6793	336424.01	87870.96	3.83	26428.00	542.25
8192	442995.70	92970.30	4.76	34076.59	550.12
14289	772210.68	174442.92	4.43	55946.68	915.65
16384	917559.78	202730.41	4.53	65539.98	1034.34
32768	1901066.20	437158.95	4.35	126737.75	1942.93
53151	3306483.18	759868.93	4.35	210633.42	3083.62
65536	4080930.52	1078806.45	3.78	255058.16	4214.09
96317	6306604.03	1628825.66	3.87	380937.03	5942.78
131072	8611666.26	2515913.92	3.42	506568.60	8705.58
191217	12811565.59	3658929.57	3.50	730217.97	11886.52
262144	18083484.64	5447663.17	3.32	1004638.04	16813.78
398853	28649635.31	9477385.58	3.02	1539847.84	27378.27
524288	39305314.66	12319792.19	3.19	2068700.77	34126.85
719289	55494180.42	17043838.68	3.26	2852260.26	45024.70
1048576	83604593.40	27344320.32	3.06	4180229.67	68360.80



This chart shows that sorting network is 3-6 times faster than `Array.Sort` for lengths of up to 1M elements (ignoring the outliers for N=8,12). An observable **anomaly** is the extreme speedup at which the network sorts 8-element vectors; the same anomaly is not observable for `Array.Sort`. (See the table and the chart below). I have no explanation for this phenomenon.



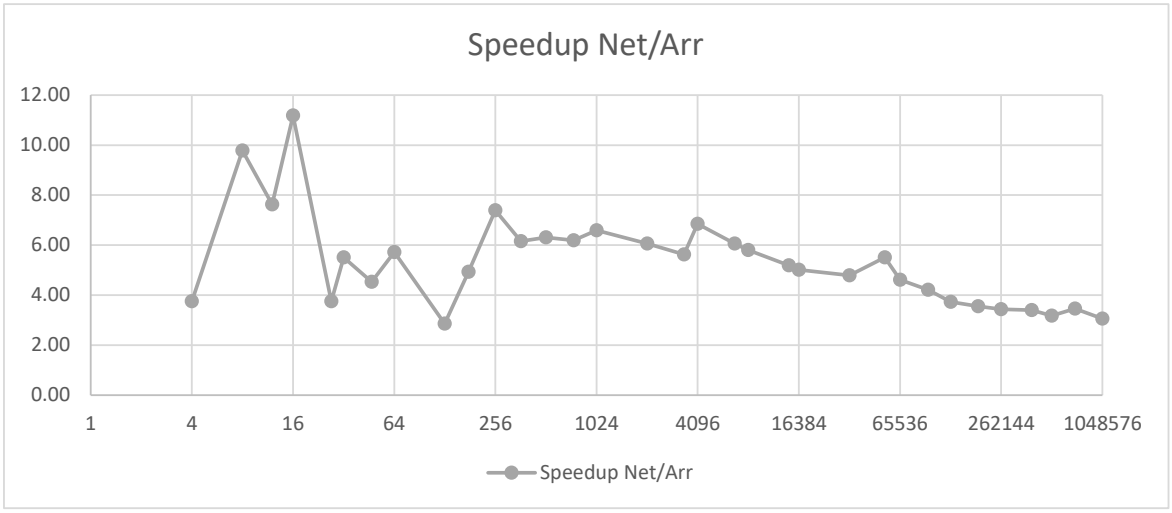
The above chart plots running time against element count on a log-log scale together with a power regression curve. The R2 value is close to 1 for both sorting methods, which indicates a very good fit. This is somewhat counter-intuitive as Array.Sort uses Introsort (https://en.wikipedia.org/wiki/Introsort) which has worst- and average-case running times O(n lg n), whereas a sorting network has O(n lg^2 n) running time.

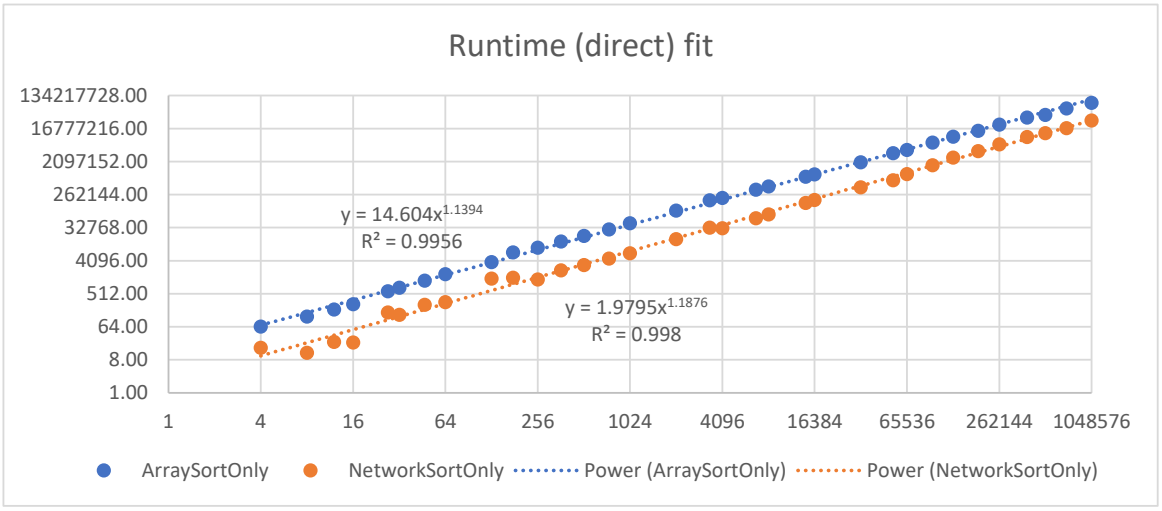


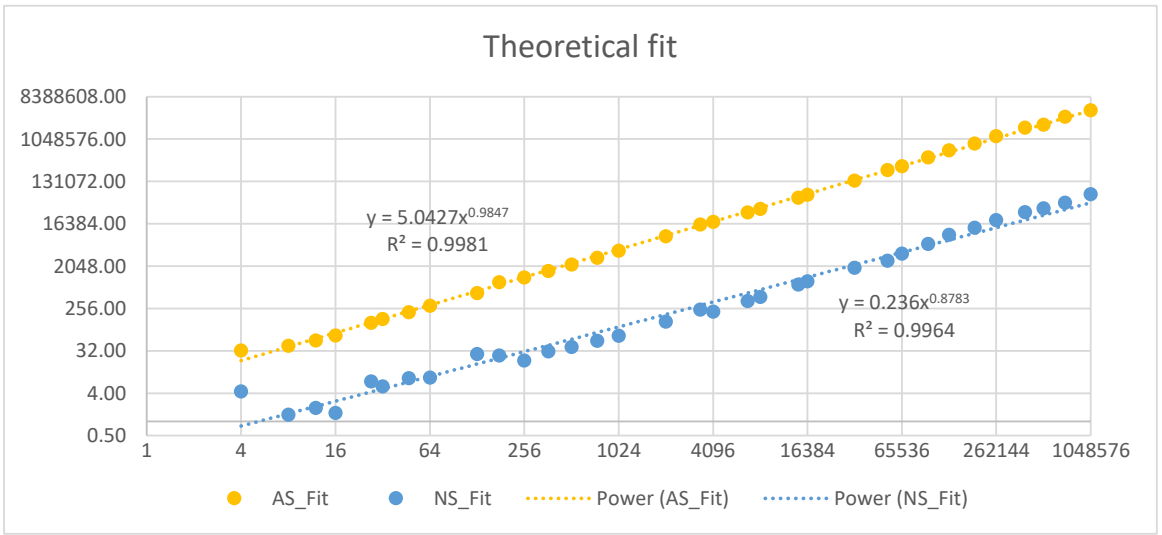
The above chart fits the experimental data to the theoretical complexity of Introsort (O(n lg n)) and network sort (O(n lg^2 n)) on a log-log scale. The direct fit from the previous chart indicates that the running time of network sort will exceed that of Array.Sort for large values of N (powers of x are 1.2132 and 1.1525 respectively), whereas the theoretical fit indicates the opposite (powers of x are 0.9979 and 0.9039). In fact, "direct fit" has better R2 value for network sort and "theoretical fit" has better R2 value for array sort. This discrepancy remains **unexplained**. It may come from cache effects which are not taken into account by classical complexity models where constat-time access to memory is assumed.

FloatBenchmark

N	ArraySortOnly	NetworkSortOnly	Speedup Net/Arr	AS_Fit	NS_Fit
4	64.74	17.25	3.75	32.37	4.31
8	121.77	12.44	9.79	40.59	1.38
12	189.78	24.89	7.62	52.94	1.94
16	268.97	24.06	11.18	67.24	1.50
27	598.92	159.66	3.75	125.96	7.06
32	757.05	137.47	5.51	151.41	5.50
47	1169.58	257.86	4.54	210.56	8.36
64	1752.78	306.58	5.72	292.13	8.52
128	3811.19	1333.55	2.86	544.46	27.22
177	6927.44	1403.84	4.93	927.67	25.17
256	9298.53	1258.63	7.39	1162.32	19.67
364	13714.35	2228.90	6.15	1611.97	30.79
512	19776.12	3131.99	6.31	2197.35	38.67
748	29498.64	4760.73	6.20	3089.87	52.23
1024	43715.94	6628.17	6.60	4371.59	66.28
2048	97104.88	16021.87	6.06	8827.72	132.41
3389	185531.73	32948.82	5.63	15821.38	239.60
4096	216382.39	31566.92	6.85	18031.87	219.21
6793	360588.70	59456.56	6.06	28326.27	366.91
8192	441965.04	76124.73	5.81	33997.31	450.44
14289	811680.50	156385.43	5.19	58806.27	820.87
16384	944085.43	188499.93	5.01	67434.67	961.73
32768	2020326.86	421466.59	4.79	134688.46	1873.18
53151	3586715.35	650593.59	5.51	228485.09	2640.17
65536	4429293.78	960243.94	4.61	276830.86	3750.95
96317	7032814.20	1667884.91	4.22	424802.21	6085.29
131072	10210969.13	2738639.00	3.73	600645.24	9476.26
191217	14588480.93	4109879.06	3.55	831496.42	13351.49
262144	21566531.03	6284282.84	3.43	1198140.61	19395.93
398853	33971822.63	9976650.30	3.41	1825902.39	28820.54
524288	39886374.90	12538006.32	3.18	2099282.89	34731.32
719289	60427514.12	17464913.09	3.46	3105821.12	46137.05
1048576	85302599.57	27872840.05	3.06	4265129.98	69682.10



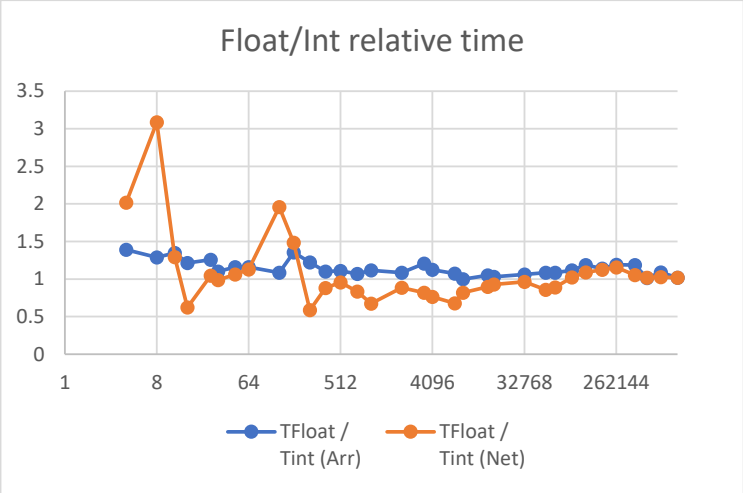




The above charts repeat the analyses from IntBenchmark. The same phenomena can be observed, the major difference being larger constants in the fitted models, which suggests that sorting floats is somewhat slower than sorting integers. This is also confirmed by IntVSFloat analysis.

Int vs Float

N	TFloat / Tint (Arr)	TFloat / Tint (Net)
4	1.390444	2.01619
8	1.290116	3.085943
12	1.348613	1.292773
16	1.214112	0.623855
27	1.257326	1.046642
32	1.101057	0.987081
47	1.157848	1.061986
64	1.161078	1.126365
128	1.083183	1.959827
177	1.356146	1.484349
256	1.220923	0.585899
364	1.0988	0.881608
512	1.110392	0.955146
748	1.068479	0.835082
1024	1.115079	0.672021
2048	1.084682	0.884157
3389	1.207321	0.817107
4096	1.12566	0.76262
6793	1.071828	0.676635
8192	0.997673	0.818807
14289	1.051113	0.896485
16384	1.028909	0.929806
32768	1.062734	0.964104
53151	1.084752	0.856192
65536	1.085364	0.890098
96317	1.115151	1.02398
131072	1.185714	1.088527
191217	1.138696	1.123246
262144	1.192609	1.153574
398853	1.185768	1.05268
524288	1.014783	1.017712
719289	1.088898	1.024705
1048576	1.02031	1.019328



This chart compares relative performance of sorting integers vs floats, both for arrays (blue curve) and sorting networks (orange curve). In both cases, sorting integers is somewhat faster than sorting floats, though the difference is much more pronounced for network sort. As the array size grows, the difference in performance diminishes, i.e., the ratio converges to 1, for both `Array.Sort` and network sort.

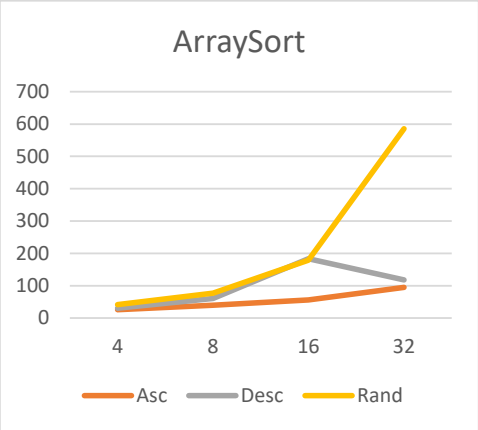
Specialized (PeriodicInt)

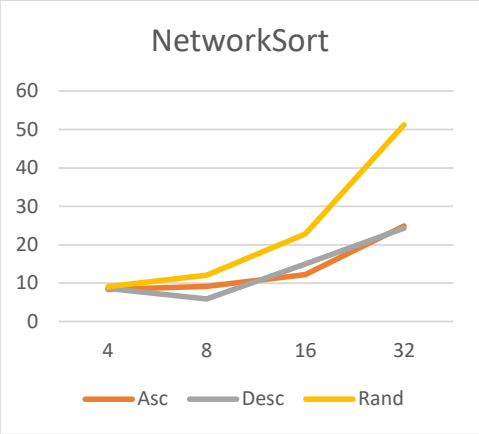
IntArraySort

N	Asc	Desc	Rand
4	25.735	30.328	41.451
8	39.532	60.039	77.281
16	56.214	183.428	179.013
32	94.773	117.784	585.454

IntNetworkSort

N	Asc	Desc	Rand
4	8.343	8.591	9.096
8	9.138	5.884	12.096
16	12.226	14.974	22.77
32	24.884	24.312	51.234





Asc Input in ascending order
Desc Input in descending order
Rand Randomized input

This benchmark compares difference in performance when presented with different input patterns. It is observable that ascending order is most favorable both for `Array.Sort` and network sort. Descending order is more favorable for network sort.

That network sort performs better for ascending or descending sequences is an **unexplained anomaly**: the algorithm is data-oblivious and always runs the same number of operations for an input of a given size, so it should run in the same time regardless of any patterns in the input. In other words, there seems to exist a side-channel that leaks information about the input sequence.