William Lopez

CS-361L-001

University of New Mexico

Department of Computer Science

Implementation and Analysis Classic Sorting Algorithms

Introduction:

The goal of this project was to implement modified versions of three classic sorting algorithms. They are the quad heapsort, 3-way mergesort, and random-pivot quicksort. All of which share an average asymptotic runtime of nlog(n). Each takes a divide and conquer approach, meaning they efficiently sort by splitting a large problem into many smaller problems. Their time complexity was benchmarked for various input sizes. They were tested on arrays of random integers, random double precision floating point numbers, random integers with a high chance for duplicates, and reverse sorted numbers. Benchmarking for many different scenarios allowed for a better understanding of the sorting algorithm's strengths and weaknesses.

All three sorting algorithms were implemented using the C++ programming language and compiled with the GNU Compiler Collection. All Algorithms and benchmarks were run on a 64-bit Windows laptop with the following processor and memory.

Processor: Intel Core it-1145G7

Physical Memory: 16 GB

Virtual Memory: 18.1 GB

Page Size: 2.38 GB

Cache size: L1 = 320 KB, L2 = 5 MB, L3 = 5 MB

The Three Algorithms:

The random pivot quicksort sorts values by randomly picking a value in the array as it's pivot. All values less than the pivot are moved to a left subarray, and all greater values are placed in a right subarray.

This is done recursively on the subarrays until they contain only one value. This process leaves a sorted array. The version implemented was the only algorithm to use recursion while still being able to handle large input values. Because of how C++ handles arrays as pointers it was also possible to implement the random pivot quicksort without needing additional storage. While It's average runtime is bounded by $\Theta(n\log(n))$ the worst-case scenario is bounded by $O(n^2)$. There is a very small chance for the worst-case scenario to occur. The main purpose of using a random pivot is to prevent an adversarial user from purposely causing the worst-case scenario.

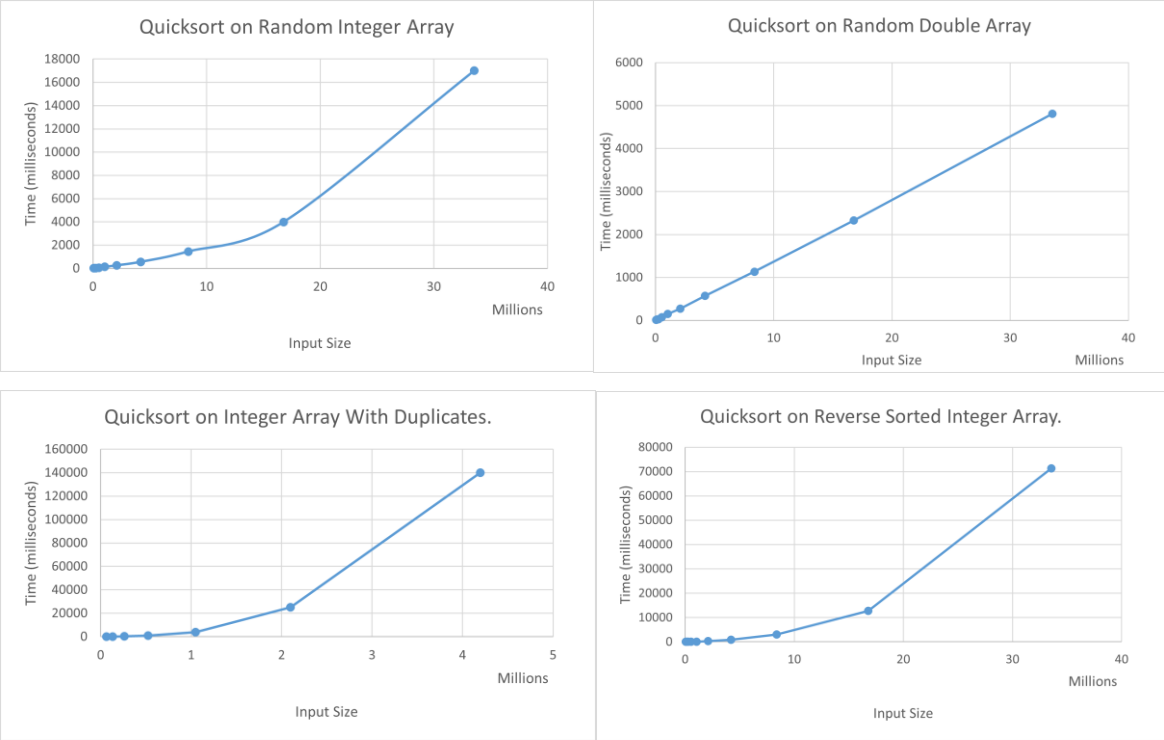
The quad heapsort uses a max heap and the ability to represent a heap as an array to efficiently sort values. It starts by creating a max heap from an array. This puts the largest value at the front of the array. The largest value is then taken and placed at the end of the array. The heap is then shrunk, excluding the last value, and restored to a max heap. Iteratively repeating this process until the heap is empty generates a sorted array. The version implemented uses a quad heap meaning a parent node has four children. Each child is indexed by 4i + (1 to 4), where i is the index of the parent. The worst-case runtime of the algorithm is bounded by $O(n\log(n))$. The height of a regular binary heap is bounded by $\log_2(n)$, by using a quad heap the height is instead bounded by $\log_4(n)$. This should make the quad heapsort slightly faster than a normal heapsort.

The 3-way mergesort works by splitting the array into smaller sorted subarrays then merging them together. The version implemented takes an iterative bottom-up approach. It starts by considering each element in the array as a subarray. Since they only have one value the subarrays are already sorted. Then they are put into groups of three and merged, creating a larger sorted subarray. This is repeated until the entire array is sorted. Unlike the other algorithms the implementation of the 3-way mergesort requires additional space equal to the size of the input. The extra space is needed as storage space for the merge operation. Its asymptotic runtime is bounded by $O(n\log(n))$. The standard mergesort splits an array into two, giving it a recurrence relation of $T(n) = 2T(n/2) + n$ giving it a division depth of $\log_2(n)$. The 3-way mergesort has a recurrence relation of $T(n) = 3T(n/3) + n$ and division depth $\log_3(n)$. This should make the 3-way mergesort slightly faster than the standard mergesort.

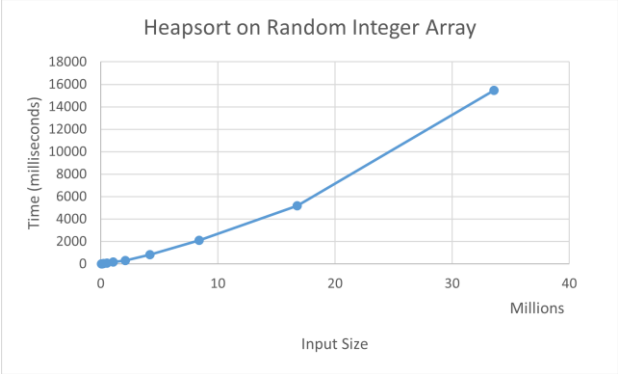
Performance and Analysis:

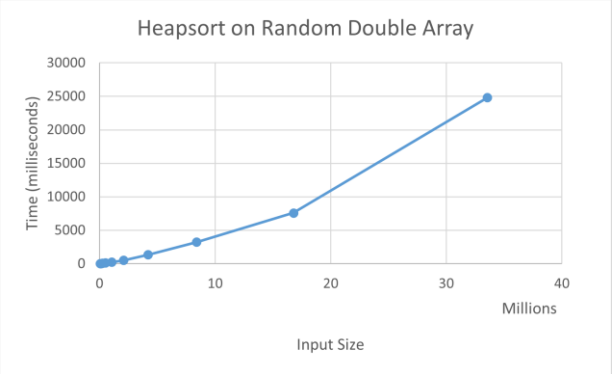
The implementation of each algorithm came with their own unique challenges. The random pivot quicksort was the easiest to implement and it caused the least trouble when debugging. Most of its operations are straight forward. The largest coding complexity came from efficiently moving values around the pivot without additional storage. The quad heapsort needed reasonable coding complexity.

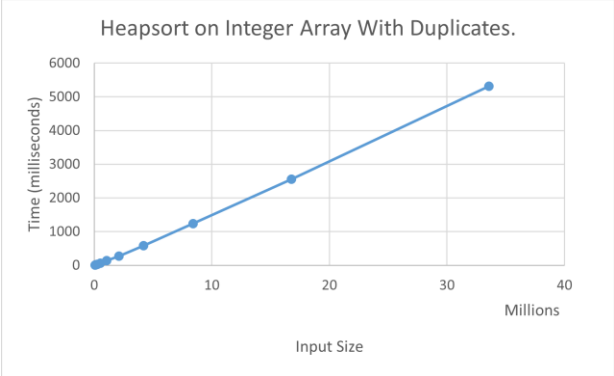
Most of the trouble and complex code came from modifying heap operations to work with a quad heap. While the 3-way mergesort was logically simple it required the most debugging and coding complexity. Since C++ lets arrays index values beyond their size there were a few bugs where values from another subarray were taken and sorted into another. Many checks were required to ensure no sub array grabbed values from another, or from some other place is memory.

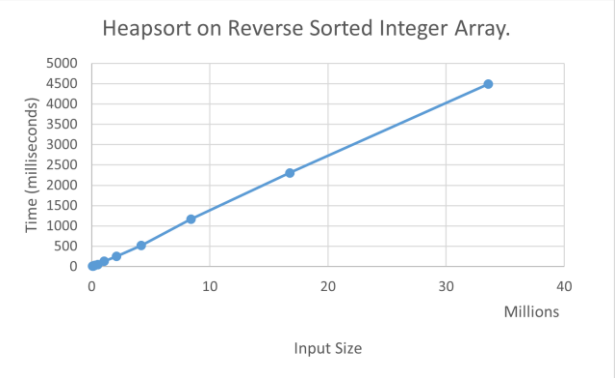


In general, the random pivot quicksort is the fastest of the three algorithms when given a smaller input. However larger values consistently caused increasing slowdowns. Compared to the other algorithms the random pivot quicksort is the worst at sorting an array containing multiple duplicate values, requiring several minutes to sort 2^{22} values. It seems that duplicate values bring the runtime closer to the quicksort's worst case asymptotic runtime. The algorithm is especially good at sorting an array of double precision floating-point numbers. It even took less time to sort an array of doubles than an array of integers. This is very unexpected as the other sorting algorithms took longer to sort the array of doubles. It's possible that the method used to generate a random integer array created more duplicates than the randomly generated double array. The random pivot quicksort was also tested for a presorted integer array of size 2^{30}. While it never crashed the program it also never finished, even after a few hours. The actual runtime of the random pivot quicksort matched the asymptotic runtime for smaller input values, but not for a very large input.









The quad heapsort performed best for the array of duplicate values and the reverse sorted array. This makes sense for the reverse sorted array as it is already a max heap. The random value arrays took several times longer to sort, especially for large input sizes. It is possible some slowdowns come from cache misses. The heapsort accesses values across the array, without any sort of locality. The cache would constantly miss and need to get data from main memory. This could be why the quad heapsort sorted the array of doubles the slowest. Less of the larger datatype could fit in cache memory. Based on the data the heapsort works best when values have some structure to them, and worst when they are random. When tested for a presorted integer array of size 2^{30} the quad heapsort took 193224.7979 milliseconds to run. The actual runtime of the quad heapsort roughly matched its asymptotic runtime.

Of the sorting algorithms tested the 3-way mergesort showed the most consistency across the four tested arrays. Runtime for each grew at the expected rate. No specific input caused the algorithm to perform slower or better. This consistency is reflected in runtime standard deviation, which remained low compared to the other algorithms. Even for input size 2^{25} the standard deviation of the runtime remained around 500 milliseconds. The 3-way mergesort a bit longer to sort doubles than integers, but doubles caused no change in runtime growth. Like the quad heapsort the increased runtime could be from cache misses caused by the larger datatype. When tested for a presorted array of size 2^{30} the 3-way mergesort took 291799.3681 milliseconds to sort it. The actual runtime of the algorithm matched its asymptotic runtime.

An interesting quirk about the mergesort is that it exploits spatial locality. This makes the algorithm efficient in terms of the cache as it minimizes cache misses. This could explain why it can handle arrays of a very large size better than the other algorithms. This advantage should not be overlooked especially if programming to optimize cache memory.

Conclusion:

When choosing one of these classic sorting algorithms it is important to think about the input it will receive and the limitations of hardware. While they have a similar asymptotic runtime, differences between the algorithms become clear when tested on a real machine. The random pivot quicksort could perform the fastest, but certain conditions could also cause it to perform the worst. The quad heapsort was very good at working with structured values but did worse when sorting random values. The 3-way mergesort appeared to be the most consistent in terms of runtime and made good use of spatial locality. The downside was that came at the cost of a difficult implementation and required additional space. The three algorithms present numerous advantages and disadvantages when it comes to their time efficiency, space efficiency, and implementation.

Bibliography:

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms, 3rd Edition. MIT Press 2009

Contributions:

Implementation, analysis, and benchmarking of the three algorithms was done by William L. Lopez.

FULL DATA ATTACHED BELOW.

Random Pivot Quicksort:

Random Integer Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	10.80554	13.76961	25.90037	48.99099	124.1902	261.0429	558.4067	1448.72	3989.697	16995.64
Standard Deviation	5.114449	3.84215	7.079812	6.301979	26.59402	33.5273	67.70091	240.8606	425.524	4694.834
Runtime (milliseconds	8.0353	10.1277	21.4885	49.644	94.6191	210.1471	451.0855	1151.167	3197.792	10541.31
	6.0156	12.5525	20.0048	39.9295	94.062	210.1889	452.7536	1157.384	3270.752	11754.49
	15.7201	15.6404	25.9095	47.0333	141.725	309.6773	671.2816	1458.91	4161.935	22437.18
	15.51	15.6237	31.828	46.8775	110.1723	294.3935	598.832	1366.577	4146.75	14281.35
	6.5328	15.6204	33.8123	63.1338	109.873	282.9357	571.9628	1424.409	4465.386	20778.97
	7.1975	5.0594	31.368	53.4056	125.4339	271.4972	613.1817	1461.444	4344.11	22249.54
	15.6251	15.7865	15.6281	47.0095	157.7718	250.9624	534.28	1507.613	4125.299	20675.32
	13.9869	15.6332	31.7002	47.3076	174.1366	246.0854	549.4688	2020.65	4212.069	14393.24
	16.4806	18.6443	31.3721	52.11	108.443	282.6191	565.54	1539.047	4062.47	12589.67
	2.9515	13.008	15.8922	43.4591	125.6652	251.9227	575.6813	1399.996	3910.409	20255.31

Random Double Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	13.65738	16.49056	29.83818	74.00638	153.385	273.6775	570.5074	1134.745	2327.075	4805.459
Standard Deviation	3.795929	7.926361	8.14578	22.12248	22.29997	26.55341	45.57593	27.74474	123.9316	233.1364
Runtime (milliseconds	11.616	13.1812	31.4098	78.7886	167.3436	283.5676	660.9588	1124.884	2365.35	4855.005
	15.7111	12.3506	31.3603	47.5351	156.9756	311.267	621.4601	1137.393	2406.288	4885.514
	15.7447	8.0115	47.2886	63.2717	192.833	313.8727	587.8963	1162.424	2466.942	4348.547
	15.6378	8.0055	31.2767	47.4926	125.4113	241.4337	548.8304	1145.94	2403.695	4937.001
	16.0453	15.6353	25.2433	109.9176	157.0462	267.1304	533.7043	1115.558	2294.242	4600.871
	15.6335	15.6223	22.3262	109.9888	153.8509	251.0571	518.9984	1084.126	2115.134	4756.073
	15.6338	31.5428	31.2496	78.803	125.3893	298.3563	538.27	1101.351	2480.896	5079.513
	4.4	29.2939	15.6258	78.1017	172.3531	266.7261	533.7763	1147.607	2326.573	5106.797
	15.6379	15.6826	31.3529	63.2015	156.8321	251.9473	564.0786	1164.012	2151.078	4608.212
	10.5137	15.5799	31.2486	62.9632	125.8144	251.4168	597.1005	1164.151	2260.55	4877.053

Random Integer Array With Many Duplicates

Input Size (2^x)	16	17	18	19	20	21	22
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304
Mean Runtime	21.12751	74.59415	269.2072	914.4491	3758.261	25186.06	139891.5
Standard Deviation	7.975515	13.38402	40.96342	85.9013	627.8158	1850.413	
Runtime (milliseconds	11.9468	95.3263	329.3162	1081.292	3771.501	23877.62	139891.5
	16.7107	68.8667	307.0905	950.6573	3583.18	26494.5	
	21.5542	78.7803	281.553	990.6744	3317.472		
	20.6637	82.1729	330.372	887.302	3294.753		
	31.2786	58.8907	266.1625	896.752	4824.398		
	15.6252	63.1023	235.2497	818.2675			
	37.3172	62.5124	235.2081	981.3688			
	24.8123	62.8963	220.4545	808.6541			
	15.639	78.9449	251.2723	848.2742			
	15.7274	94.4487	235.3927	881.2484			

Reverse Sorted Integer Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	8.59953	11.74855	15.0953	39.46554	98.23893	279.8443	873.675	3037.736	12679.45	71341.1
Standard Deviation	5.31303	4.049552	1.951012	16.25091	22.20546	31.50389	69.94055	402.6122	3110.853	32742.69
Runtime (milliseconds	5.7484	6.0091	11.8436	29.5224	79.0091	239.6593	758.8529	2346.972	8889.805	44045.95
	3.4437	7.5145	12.7134	19.761	72.368	254.7665	767.5232	2329.146	9219.897	48425.66
	15.6377	13.0815	16.5774	31.2948	109.5363	308.8361	958.8069	3412.282	13730.67	112970.5
	15.6443	15.8944	18.2086	78.9932	124.3024	345.6906	941.7643	3460.439	11053.34	108299.8

5.5111	15.6375	13.0661	42.4243	95.8019	282.8458	864.5372	3036.34	11813.09	106701.8
10.1743	16.1094	15.662	35.3456	139.2311	282.9896	926.5263	2937.789	11342.41	45238
4.1726	8.5118	15.6152	31.673	94.3013	252.0114	880.554	3307.901	11335.07	109260.2
15.9358	7.5349	15.6496	46.8733	110.082	266.0282	917.1452	3271.096	16616.5	44415.42
6.7115	11.5278	15.5988	31.6652	78.949	298.7833	824.9577	3177.878	14213.66	45865.54
3.0159	15.6646	16.0183	47.1026	78.8082	266.8318	896.0827	3097.519	18580.1	48188.19

Quad Heapsort

Random Integer Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	11.24891	14.58148	33.91502	70.18362	153.8955	311.3382	838.6822	2095.775	5175.883	15447.69
Standard Deviation	4.606338	2.804165	12.31253	10.08383	33.09837	33.41374	84.73279	135.873	468.1734	3175.751
Runtime (milliseconds	7.9972	9.7341	24.7715	66.136	126.1888	305.5965	788.2789	2165.313	4409.804	10693.88
	7.5464	10.0676	26.3242	70.7802	151.2406	340.1601	771.5057	1780.437	4503.459	10794.02
	15.6524	15.6287	31.1015	64.443	219.3349	346.1856	987.5801	2168.677	5183.161	19229.47
	15.6534	15.6001	43.1115	77.1256	141.6935	366.8199	814.933	2169.519	5126.579	17752.37
	16.6139	16.0358	63.0424	95.6183	204.5131	300.9403	992.4223	2060.868	6102.58	16771.06
	7.6589	15.6364	31.1766	70.4929	141.9248	283.3215	833.3157	2056.289	5233.946	17510.41
	7.1927	15.7967	23.6349	63.0779	157.0311	298.0854	818.6303	2264.492	5242.857	13941.4
	4.5788	15.6437	31.2615	62.4028	116.63	330.0283	761.1816	2198.934	5326.494	17604.28
	15.0103	18.6451	41.2295	69.2092	130.8602	282.0637	848.545	1999.667	5354.737	17708.97
	14.5851	13.0266	23.4966	62.5503	149.5376	260.1809	770.4289	2093.551	5275.21	12471.03

Random Double Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	12.12544	22.68071	48.98749	102.0977	234.7784	501.0275	1315.032	3203.795	7576.35	24784.99
Standard Deviation	4.075015	7.013375	17.99818	34.18192	43.16211	115.5083	131.5125	188.0612	512.6527	5923.301
Runtime (milliseconds	14.0403	26.7117	96.247	114.7209	266.997	270.7753	1537.427	3434.714	7832.423	34061.46
	6.7179	15.3041	41.7199	174.4537	266.9075	627.007	1558.727	3253.838	8658.526	18828.04
	16.1425	14.5134	50.0147	126.6127	283.4497	619.1409	1316.269	2983.539	7243.528	27190.93
	15.9342	15.8082	47.7758	78.5317	283.0615	612.2068	1209.96	2985.769	7526.399	26238.09
	10.3566	16.046	47.1761	78.5713	224.3383	517.7406	1195.349	3439.55	7031.133	28515.82
	8.1663	31.2354	46.8866	82.2572	264.2449	439.163	1270.499	3078.032	7621.808	15817.84
	6.5123	31.2549	31.7101	94.1717	204.5804	433.4747	1244.337	3141.557	7192.826	26541.94
	15.6378	19.8966	46.8901	78.3426	204.5558	438.5346	1242.3	3026.706	7844.95	16227.81
	15.6211	29.1663	31.2525	62.5175	176.3456	600.4357	1225.739	3443.847	6937.584	25819.24
	15.6267	26.8705	50.2021	130.7979	173.3028	451.7959	1349.718	3250.396	7874.32	28608.68

Random Integer Array With Many Duplicates

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	10.33409	13.63885	31.60741	57.44426	140.3274	274.072	585.7624	1240.811	2558.509	5311.412
Standard Deviation	4.882991	4.662027	9.44355	10.40316	20.06036	31.01521	49.33209	102.1562	110.9621	852.0711
Runtime (milliseconds	5.36	15.639	31.4905	62.4915	167.9597	307.1663	661.6146	1423.283	2702.171	5087.284
	6.9667	8.6613	20.613	55.3891	156.8054	272.0231	660.6176	1335.839	2707.265	5137.034
	15.8307	17.0526	27.15	63.7104	169.1247	251.5358	565.2524	1273.418	2508.278	4928.247
	15.6432	14.8185	31.2804	62.9308	126.3059	334.0832	612.2169	1178.82	2468.933	5001.93
	7.5149	15.6331	46.857	46.9775	125.4178	279.8599	533.5255	1147.815	2483.062	5265.34
	8.3423	15.6237	17.6302	47.3877	125.5256	299.4788	596.7988	1243.45	2612.688	5089.12
	8.4236	15.6332	31.2618	78.3229	133.8535	250.7578	597.131	1131.752	2701.394	4995.991
	15.9735	2.0649	31.2607	47.3682	156.7173	248.9671	544.7726	1255.881	2452.573	5028.664
	15.6342	15.6365	31.6389	47.3753	116.0481	236.5837	566.1088	1099.652	2513.545	4864.815
	3.6518	15.6257	46.8916	62.4892	125.5155	260.2644	519.5856	1318.197	2435.176	7715.693

Reverse Sorted Integer Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	11.60208	13.08692	29.08353	50.57206	133.9619	254.3644	520.3161	1162.806	2305.356	4486.383
Standard Deviation	4.894641	5.65445	10.65774	8.120013	33.57803	39.38894	60.9679	254.6917	375.4422	508.0358
Runtime (milliseconds	4.056	9.5819	19.5946	39.2514	98.8621	195.473	408.8553	844.3993	1738.846	3551.8
	6.0322	10.5024	23.0143	43.0692	94.1822	200.3154	425.8148	861.5798	1756.726	3563.948
	15.8427	15.9672	31.7701	45.9707	109.9031	257.9235	550.5966	1243.256	2328.15	4473.485
	15.6383	13.1104	43.3623	47.2784	109.8443	287.1469	612.7356	1115.131	2421.11	4902.739

5.0144	3.0079	46.8612	52.2969	125.639	282.7808	549.53	1037.424	2405.754	4798.368
11.012	16.366	15.6554	57.8558	188.4314	298.7894	534.2979	1193.172	3082.861	4743.365
15.6486	15.0136	31.9272	46.8851	141.0148	235.5473	502.7286	1255.427	2291.644	4537.716
15.6356	23.6766	31.2693	62.9517	173.0138	219.7359	533.6671	1178.643	2404.153	4853.968
11.4213	8.0072	16.0447	62.6109	172.6773	266.9005	534.6062	1138.809	2248.406	4631.198
15.7197	15.636	31.3362	47.5505	126.0514	299.031	550.3289	1760.215	2375.914	4807.245

3-Way Mergesort

Random Integer Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	13.01054	20.44383	44.44392	85.65083	169.7454	349.3852	656.3648	1371.884	2841.992	5591.98
Standard Deviation	3.972269	7.025788	8.453242	16.69574	22.94733	45.81012	61.95184	138.0416	310.1984	612.8555
Runtime (milliseconds	4.0081	16.2829	33.8896	67.7451	127.9745	266.0892	556.586	1148.604	2241.891	4503.361
	10.1753	19.2667	34.2316	67.9528	140.0297	289.9118	550.5957	1110.361	2311.57	4485.271
	14.7114	15.0083	46.1195	92.9882	189.4271	423.8788	707.2471	1465.493	2798.645	5711.299
	15.6437	31.6872	47.4797	94.9177	171.5544	392.2595	675.4996	1351.667	3049.637	5778.421
	15.7501	31.2898	46.8597	89.6899	173.0312	345.8552	738.106	1492.95	2906.69	6148.663
	15.9732	15.7033	62.985	125.129	172.9161	345.6997	692.6538	1412.705	2953.801	5684.51
	12.4738	15.6138	47.2755	78.7934	172.9978	344.9904	692.1246	1368.378	3034.499	5962.596
	9.7367	28.2343	46.8893	78.513	204.0913	345.6327	675.2884	1397.151	3016.177	5714.068
	15.9972	15.7274	39.3415	78.5176	157.0312	377.9526	630.1071	1509.921	2998.82	6258.399
	15.6359	15.6246	39.3678	82.2616	188.4009	361.5817	645.4392	1461.612	3108.191	5673.216

Random Double Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	14.58048	16.58863	46.50379	115.5987	197.3642	408.2908	845.4562	1790.689	3587.231	7132.027
Standard Deviation	8.039759	2.625942	14.25117	27.26507	18.70897	17.90758	97.56061	173.9225	192.1042	364.6909
Runtime (milliseconds	12.5457	22.1438	47.5483	137.2688	204.7313	424.7359	941.9369	2183.451	3788.862	7641.128
	7.1607	14.7272	31.2665	107.6139	220.1645	424.2502	1052.015	1947.675	3992.86	7703.274
	16.0327	14.0096	47.055	126.1004	204.1996	424.1848	927.8215	1844.168	3410.667	6781.853
	15.6768	15.779	63.1976	78.9565	204.9149	408.5937	801.6931	1870.19	3529.926	6979.696
	34.4297	16.0138	74.444	87.3771	204.8075	393.5926	737.6813	1686.868	3429.261	6957.237
	12.5143	15.6203	46.8835	90.0129	220.378	376.9541	773.9858	1698.734	3472.831	7518.056
	4.0085	15.6278	31.2489	141.4627	189.7624	392.5443	815.9269	1617.184	3628.093	6703.647
	12.0082	15.6523	29.1485	94.4144	163.6835	420.5717	785.6683	1696.183	3411.246	6892.654
	15.8164	20.6508	46.8715	135.4554	172.5516	392.7548	784.9576	1650.254	3501.268	6933.665
	15.6118	15.6617	47.3741	157.3248	188.4486	424.7259	832.8755	1712.185	3707.295	7209.066

Random Integer Array With Many Duplicates

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	10.31158	16.23596	36.66447	72.35491	149.3667	294.3377	631.4182	1228.966	2523.979	4742.595
Standard Deviation	6.409252	5.48011	11.39529	25.49401	16.78677	22.85706	30.991	126.1244	347.3922	254.415
Runtime (milliseconds	2.0162	17.3189	61.4271	65.7316	151.7147	315.3273	627.9245	1271.916	2479.834	4903.379
	1.0288	14.7032	25.1672	56.2804	151.268	267.2484	599.5322	1544.585	2420.304	4935.06
	10.5181	4.0105	28.1663	58.3782	172.8364	329.9082	609.6113	1288.352	2385.035	4488.024
	15.6602	15.6346	31.5592	57.7897	171.3066	298.083	598.9335	1084.645	2358.942	4665.233
	15.6723	15.6272	47.3389	46.8753	141.2646	271.9464	643.345	1162.371	2390.322	4560.372
	18.7205	16.7166	46.9124	65.7473	148.2936	313.4725	630.1193	1161.852	2356.547	4401.07
	8.7588	15.1212	31.7322	58.8482	155.6196	313.9628	615.1252	1177.524	2434.537	4601.943
	12.0357	24.7355	31.3444	109.6003	147.2254	282.1964	621.3896	1161.919	2297.042	5234.081
	3.0517	22.7625	31.3431	125.6208	141.0331	284.1965	680.6789	1241.059	3476.84	4703.98
	15.6535	15.7294	31.6539	78.6773	113.1049	267.0354	687.5223	1195.44	2640.389	4932.806

Reverse Sorted Integer Array

Input Size (2^x)	16	17	18	19	20	21	22	23	24	25
Actual Input Size	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432
Mean Runtime	6.38173	10.75532	30.25745	43.76788	107.8351	189.3925	407.0716	827.9754	1695.166	3474.793
Standard Deviation	2.314629	2.072645	8.827322	10.91284	14.57988	21.34682	58.85529	109.2918	301.3602	495.7538
Runtime (milliseconds	5.1052	11.6747	19.8822	35.9791	82.6304	170.3144	305.5363	624.144	1313.625	2559.922
	8.0169	10.2918	20.4446	45.1323	89.4919	161.9847	312.0288	650.2402	1309.645	2555.275
	1.6149	10.7479	29.1417	48.3361	114.7186	228.1573	408.4739	927.7303	1300.5	3509.378
	6.0465	12.2328	35.7503	40.2716	109.97	203.2639	471.581	816.8877	1934.102	3683.441

10.3128	8.3672	35.3738	31.6711	114.9843	210.4987	440.3613	816.589	1776.793	3532.971
7.8962	9.8204	33.3428	39.1346	126.3386	180.5938	455.456	911.953	1759.112	3692.356
7.0996	7.6603	33.4936	44.3596	110.1726	173.5972	401.4805	927.0119	1787.907	3833.765
4.8869	15.1437	31.9301	53.2928	94.6599	172.2083	396.8139	894.8384	2217.082	3788.35
5.8808	10.8088	45.9835	67.7192	109.8635	188.9794	408.0148	829.196	1828.157	3805.276
6.9575	10.8056	17.2319	31.7824	125.5216	204.3272	470.9692	881.1634	1724.737	3787.199