Program Structures and Algorithms

Spring 2023 (Section 3)

Assignment 5 – Parallel Sorting

Name: Abhinav Choudhary

NUID: 002780326

Task

- To implement parallel sort by updating merge sort algorithm where each partition is sorted in parallel. If number of elements or array size (n) is less than cut off value then array is to be sorted using Arrays.sort();
- To run the parallel sort algorithm at varying cut off points to locate the most ideal cut off position.
- To run the parallel sort algorithm at varying parallelization threads (t) to find the most ideal thread count. The thread increments in powers of 2 (i.e., 2^p), where p increments in intervals of 2 (i.e., p = p+2).
- To find the most appropriate point where both cut off and threads meet to provide the most ideal location for parallelization.
- The array should be of sufficient size and array elements should be randomized with values between 0 and 10000000.

Relationship Conclusion

By running the program for increasing number of array sizes (n) with increasing cut off (c) and thread count (t), following details were observed:

1. Within a certain array size (n) and thread count (t), the most ideal cut off (c) was (in most cases) at or around

   c = $\frac{n}{10}$
2. Within a certain array size (n) and cut off (c), the most ideal thread count (t) was:
   - For relatively smaller sized arrays (i.e., 2000000 and 4000000) at

     t = 2^8 or 256
   - For relatively larger sized arrays (i.e., 8000000 and 16000000) at

     t = 2^{10} or 1024
3. The biggest increase in performance in regards to thread count (or parallelization) was going from 2^2 (or 4) to 2^4 (or 16).

Thus, we can say that for parallel sort algorithm the most ideal cut off (c) is at $\frac{n}{10}$ and most ideal number of threads (t) are 2^{10}. As cut off increases, there is a massive increase at the beginning which eventually becomes stable and again starts to grow and becomes stable at a relatively higher value. Thread count, on the other hand performs worst at lowest value and improves until it reaches the threshold of 2^{10} after which there is either no massive increase in performance or a decrease in performance.

Evidence to support conclusion

To find the relationship between array size (n), cut off (c), and thread count (t), multiple runs were conducted in which array size was doubled going from 2000000 to 16000000, cut off location was calculated with the following formula: (array.size / 200) * j and thread count's power was doubled going from 2^2 to 2^{12}.

Following data was gathered as a result of this experiment:

(check Assignment5_ParSort.xlsx for better data readability)

Array Size = 2000000			Array Size = 4000000			Array Size = 8000000			Array Size = 16000000		
Parallelism	**Cut off**	**Time (in ms)**	**Parallelism**	**Cut off**	**Time (in ms)**	**Parallelism**	**Cut off**	**Time (in ms)**	**Parallelism**	**Cut off**	**Time (in ms)**
4	10000	1060	4	20000	1748	4	40000	3161	4	80000	6114
	110000	619		220000	1204		440000	2537		880000	5446
	210000	628		420000	1304		840000	2972		1680000	6317
	310000	667		620000	1382		1240000	3324		2480000	6592
	410000	680		820000	1384		1640000	3311		3280000	6689
	510000	651		1020000	1343		2040000	3169		4080000	6213
	610000	653		1220000	1347		2440000	3168		4880000	6062
	710000	642		1420000	1322		2840000	3145		5680000	6424
	810000	647		1620000	1358		3240000	2947		6480000	6176
	910000	664		1820000	1321		3640000	3014		7280000	6128
	1010000	876		2020000	1815		4040000	4145		8080000	8743
	1110000	874		2220000	1827		4440000	4387		8880000	8819
	1210000	870		2420000	1818		4840000	4281		9680000	8529
	1310000	881		2620000	1820		5240000	4264		10480000	8337
	1410000	878		2820000	1815		5640000	4335		11280000	8336
	1510000	882		3020000	1824		6040000	4369		12080000	8440
	1610000	878		3220000	1816		6440000	4502		12880000	8713
	1710000	872		3420000	1818		6840000	4323		13680000	8739
	1810000	882		3620000	1827		7240000	4219		14480000	8761
	1910000	883		3820000	1822		7640000	3764		15280000	8366
	2010000	1391		4020000	2935		8040000	6061		16080000	12693

16	10000	594	16	20000	1312	16	40000	2584	16	80000	5719
	110000	502		220000	1026		440000	2138		880000	4936
	210000	512		420000	1039		840000	2238		1680000	4978
	310000	552		620000	1119		1240000	2406		2480000	5341
	410000	548		820000	1095		1640000	2443		3280000	5479
	510000	637		1020000	1317		2040000	2884		4080000	6454
	610000	640		1220000	1317		2440000	2833		4880000	6263
	710000	634		1420000	1303		2840000	2716		5680000	6662
	810000	634		1620000	1342		3240000	2987		6480000	6483
	910000	635		1820000	1305		3640000	3064		7280000	6333
	1010000	888		2020000	1823		4040000	4163		8080000	8960
	1110000	885		2220000	1818		4440000	4150		8880000	8953
	1210000	883		2420000	1822		4840000	4341		9680000	9070
	1310000	903		2620000	1823		5240000	3962		10480000	8824
	1410000	885		2820000	1826		5640000	4608		11280000	8672
	1510000	879		3020000	1812		6040000	4614		12080000	8554
	1610000	878		3220000	1826		6440000	4066		12880000	8973
	1710000	884		3420000	1840		6840000	3796		13680000	9346
	1810000	881		3620000	1818		7240000	3755		14480000	9221
	1910000	877		3820000	1811		7640000	3733		15280000	8626
	2010000	1397		4020000	2934		8040000	6128		16080000	14618

64	10000	651	64	20000	1208	64	40000	2565	64	80000	5962
	110000	507		220000	1001		440000	2228		880000	4712
	210000	497		420000	996		840000	2196		1680000	4852
	310000	545		620000	1119		1240000	3069		2480000	5274
	410000	545		820000	1091		1640000	3043		3280000	5173
	510000	630		1020000	1313		2040000	2859		4080000	6184
	610000	631		1220000	1317		2440000	4187		4880000	6594
	710000	650		1420000	1303		2840000	2791		5680000	6189
	810000	636		1620000	1308		3240000	2752		6480000	6368
	910000	660		1820000	1316		3640000	2718		7280000	6555
	1010000	882		2020000	1834		4040000	3851		8080000	8464
	1110000	883		2220000	1826		4440000	3869		8880000	8411
	1210000	870		2420000	1826		4840000	3881		9680000	8590
	1310000	877		2620000	1825		5240000	3903		10480000	8654
	1410000	878		2820000	1822		5640000	4574		11280000	8765
	1510000	877		3020000	1811		6040000	4527		12080000	8634
	1610000	889		3220000	1826		6440000	4216		12880000	8164
	1710000	880		3420000	1818		6840000	3960		13680000	8233
	1810000	877		3620000	1817		7240000	3877		14480000	8331
	1910000	878		3820000	1846		7640000	3831		15280000	8436
	2010000	1400		4020000	2916		8040000	6207		16080000	14811

256	10000	576	256	20000	1063	256	40000	2351	256	80000	5360
	110000	492		220000	995		440000	2201		880000	4741
	210000	489		420000	1008		840000	2138		1680000	4897
	310000	542		620000	1093		1240000	2426		2480000	5454
	410000	535		820000	1094		1640000	2493		3280000	5307
	510000	651		1020000	1309		2040000	2850		4080000	6386
	610000	636		1220000	1333		2440000	2877		4880000	6474
	710000	630		1420000	1316		2840000	2842		5680000	6181
	810000	630		1620000	1317		3240000	2707		6480000	6419
	910000	627		1820000	1294		3640000	2682		7280000	6501
	1010000	876		2020000	1829		4040000	3808		8080000	8876
	1110000	879		2220000	1819		4440000	3822		8880000	8730
	1210000	868		2420000	1830		4840000	3889		9680000	8696
	1310000	869		2620000	1828		5240000	3911		10480000	9107
	1410000	875		2820000	1840		5640000	3910		11280000	9191
	1510000	887		3020000	1821		6040000	3915		12080000	8861
	1610000	880		3220000	1825		6440000	3902		12880000	8844
	1710000	884		3420000	1811		6840000	3910		13680000	8879
	1810000	881		3620000	1817		7240000	3772		14480000	8759
	1910000	865		3820000	1817		7640000	3763		15280000	8863
	2010000	1401		4020000	2934		8040000	6146		16080000	14253

1024	10000	657	1024	20000	1082	1024	40000	2335	1024	80000	5237
	110000	502		220000	1001		440000	2093		880000	4627
	210000	491		420000	1007		840000	2184		1680000	4613
	310000	547		620000	1093		1240000	2320		2480000	5235
	410000	540		820000	1095		1640000	2451		3280000	5110
	510000	629		1020000	1317		2040000	2847		4080000	6360
	610000	636		1220000	1309		2440000	2882		4880000	6254
	710000	640		1420000	1324		2840000	2864		5680000	5960
	810000	635		1620000	1327		3240000	2694		6480000	6506
	910000	636		1820000	1306		3640000	2673		7280000	6121
	1010000	883		2020000	1806		4040000	3810		8080000	8407
	1110000	889		2220000	1822		4440000	3774		8880000	8564
	1210000	884		2420000	1818		4840000	3869		9680000	8851
	1310000	872		2620000	1828		5240000	3899		10480000	8711
	1410000	879		2820000	1820		5640000	3893		11280000	8682
	1510000	896		3020000	1811		6040000	3886		12080000	8721
	1610000	1025		3220000	1826		6440000	3881		12880000	8743
	1710000	969		3420000	1828		6840000	3920		13680000	8598
	1810000	995		3620000	1819		7240000	3802		14480000	9257
	1910000	1046		3820000	1822		7640000	3793		15280000	8953
	2010000	1680		4020000	2905		8040000	6152		16080000	14874
4096	10000	990	4096	20000	1215	4096	40000	2464	4096	80000	5456
	110000	547		220000	1016		440000	2162		880000	4959
	210000	517		420000	999		840000	2152		1680000	4806
	310000	542		620000	1095		1240000	2485		2480000	5335
	410000	554		820000	1101		1640000	2449		3280000	5157
	510000	637		1020000	1315		2040000	2848		4080000	6302
	610000	647		1220000	1317		2440000	2669		4880000	6127
	710000	641		1420000	1302		2840000	2679		5680000	6401
	810000	637		1620000	1319		3240000	2700		6480000	6528
	910000	634		1820000	1309		3640000	2824		7280000	6124
	1010000	882		2020000	1813		4040000	3911		8080000	8744
	1110000	881		2220000	1820		4440000	3902		8880000	8707
	1210000	879		2420000	1809		4840000	3884		9680000	8752
	1310000	885		2620000	1808		5240000	3887		10480000	8517
	1410000	888		2820000	1821		5640000	3763		11280000	8554
	1510000	877		3020000	1812		6040000	3764		12080000	8313
	1610000	874		3220000	1811		6440000	3769		12880000	8765
	1710000	883		3420000	1812		6840000	3785		13680000	8732
	1810000	873		3620000	1833		7240000	3762		14480000	8483
	1910000	880		3820000	1829		7640000	3784		15280000	8383
	2010000	1388		4020000	2936		8040000	6101		16080000	14404

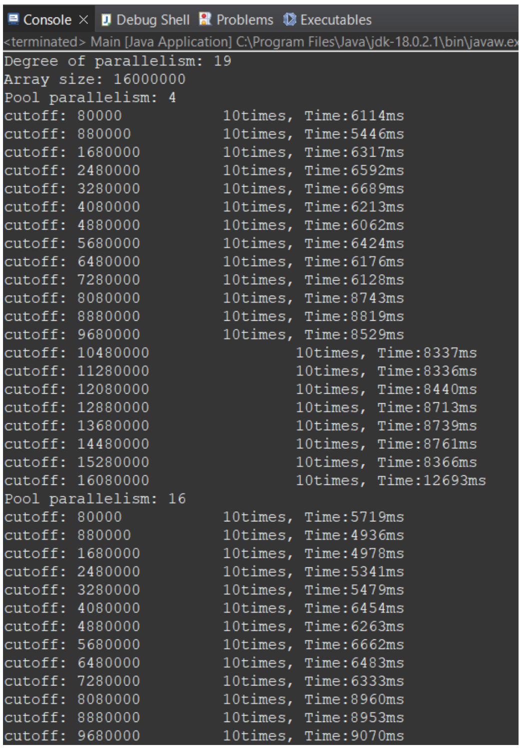
In the above excel data, you can see that cells highlighted light green show the best scenario where cut off and thread count both are at their most optimal location. Light yellow show the best cut off for a given array size and thread count.

One observation or pattern that can be observed is that for a particular cut off, thread count starts at a relatively high time value and then improves or time value gets lower until it reaches thread count of 256 or 1024 (depending on size of array), after which the time values again start to climb. Indicating that biggest performance improvement happens at 16 (2^4) and stays in somewhat of a best-case scenario until it reaches thread count of 256 and 1024 (2^8 and 2^{10}) that is the most ideal location, anything above or below that can produce worst results.

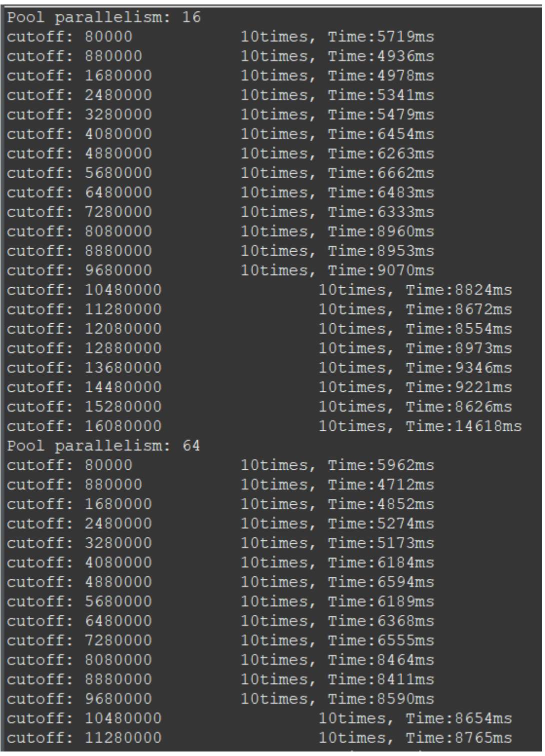
Following are the screenshots from the Main class:

(Note: Due to multiple runs and large number of results, providing screenshot only for array size 16000000)

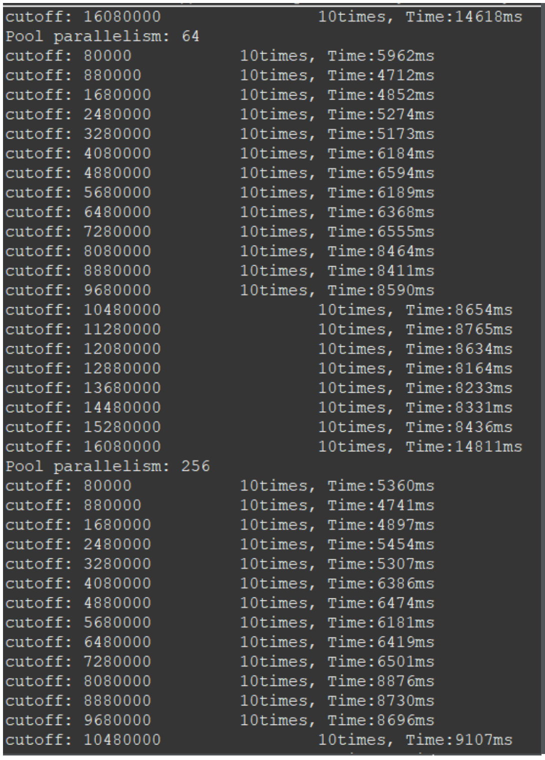
Array Size = 16000000



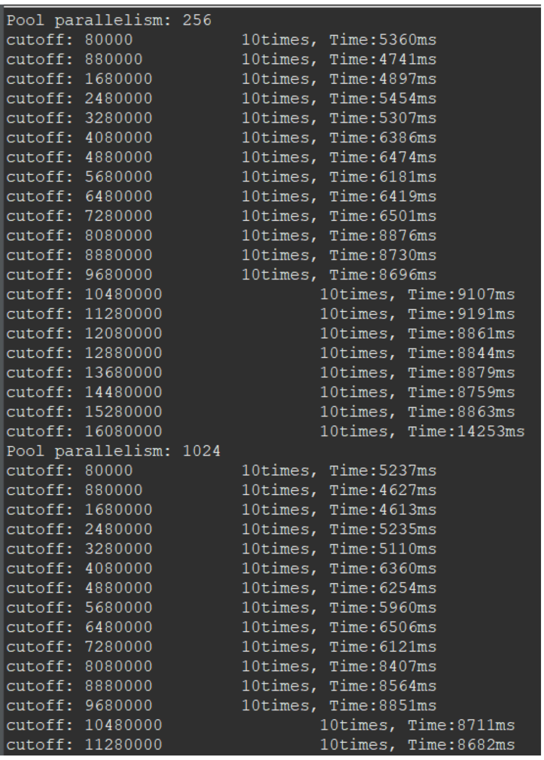
```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.ex
Degree of parallelism: 19
Array size: 16000000
Pool parallelism: 4
cutoff: 80000           10times, Time:6114ms
cutoff: 880000          10times, Time:5446ms
cutoff: 1680000         10times, Time:6317ms
cutoff: 2480000         10times, Time:6592ms
cutoff: 3280000         10times, Time:6689ms
cutoff: 4080000         10times, Time:6213ms
cutoff: 4880000         10times, Time:6062ms
cutoff: 5680000         10times, Time:6424ms
cutoff: 6480000         10times, Time:6176ms
cutoff: 7280000         10times, Time:6128ms
cutoff: 8080000         10times, Time:8743ms
cutoff: 8880000         10times, Time:8819ms
cutoff: 9680000         10times, Time:8529ms
cutoff: 10480000                10times, Time:8337ms
cutoff: 11280000                10times, Time:8336ms
cutoff: 12080000                10times, Time:8440ms
cutoff: 12880000                10times, Time:8713ms
cutoff: 13680000                10times, Time:8739ms
cutoff: 14480000                10times, Time:8761ms
cutoff: 15280000                10times, Time:8366ms
cutoff: 16080000                10times, Time:12693ms
Pool parallelism: 16
cutoff: 80000           10times, Time:5719ms
cutoff: 880000          10times, Time:4936ms
cutoff: 1680000         10times, Time:4978ms
cutoff: 2480000         10times, Time:5341ms
cutoff: 3280000         10times, Time:5479ms
cutoff: 4080000         10times, Time:6454ms
cutoff: 4880000         10times, Time:6263ms
cutoff: 5680000         10times, Time:6662ms
cutoff: 6480000         10times, Time:6483ms
cutoff: 7280000         10times, Time:6333ms
cutoff: 8080000         10times, Time:8960ms
cutoff: 8880000         10times, Time:8953ms
cutoff: 9680000         10times, Time:9070ms
```



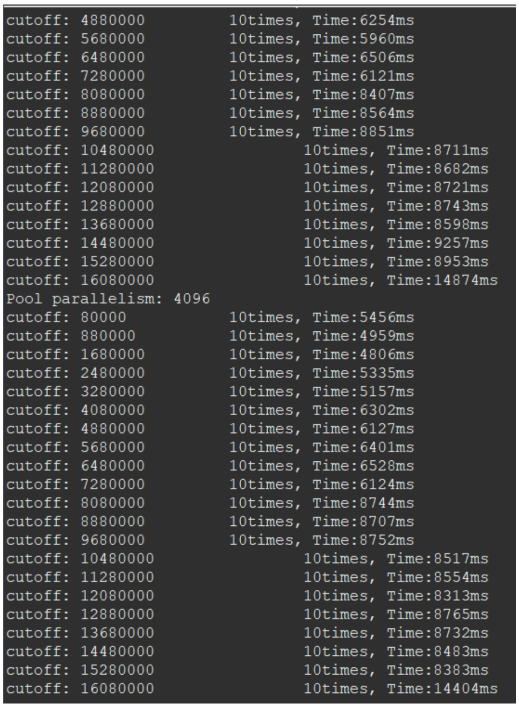
```
Pool parallelism: 16
cutoff: 80000           10times, Time:5719ms
cutoff: 880000          10times, Time:4936ms
cutoff: 1680000         10times, Time:4978ms
cutoff: 2480000         10times, Time:5341ms
cutoff: 3280000         10times, Time:5479ms
cutoff: 4080000         10times, Time:6454ms
cutoff: 4880000         10times, Time:6263ms
cutoff: 5680000         10times, Time:6662ms
cutoff: 6480000         10times, Time:6483ms
cutoff: 7280000         10times, Time:6333ms
cutoff: 8080000         10times, Time:8960ms
cutoff: 8880000         10times, Time:8953ms
cutoff: 9680000         10times, Time:9070ms
cutoff: 10480000                10times, Time:8824ms
cutoff: 11280000                10times, Time:8672ms
cutoff: 12080000                10times, Time:8554ms
cutoff: 12880000                10times, Time:8973ms
cutoff: 13680000                10times, Time:9346ms
cutoff: 14480000                10times, Time:9221ms
cutoff: 15280000                10times, Time:8626ms
cutoff: 16080000                10times, Time:14618ms
Pool parallelism: 64
cutoff: 80000           10times, Time:5962ms
cutoff: 880000          10times, Time:4712ms
cutoff: 1680000         10times, Time:4852ms
cutoff: 2480000         10times, Time:5274ms
cutoff: 3280000         10times, Time:5173ms
cutoff: 4080000         10times, Time:6184ms
cutoff: 4880000         10times, Time:6594ms
cutoff: 5680000         10times, Time:6189ms
cutoff: 6480000         10times, Time:6368ms
cutoff: 7280000         10times, Time:6555ms
cutoff: 8080000         10times, Time:8464ms
cutoff: 8880000         10times, Time:8411ms
cutoff: 9680000         10times, Time:8590ms
cutoff: 10480000                10times, Time:8654ms
cutoff: 11280000                10times, Time:8765ms
```



```
cutoff: 16080000                10times, Time:14618ms
Pool parallelism: 64
cutoff: 80000           10times, Time:5962ms
cutoff: 880000          10times, Time:4712ms
cutoff: 1680000         10times, Time:4852ms
cutoff: 2480000         10times, Time:5274ms
cutoff: 3280000         10times, Time:5173ms
cutoff: 4080000         10times, Time:6184ms
cutoff: 4880000         10times, Time:6594ms
cutoff: 5680000         10times, Time:6189ms
cutoff: 6480000         10times, Time:6368ms
cutoff: 7280000         10times, Time:6555ms
cutoff: 8080000         10times, Time:8464ms
cutoff: 8880000         10times, Time:8411ms
cutoff: 9680000         10times, Time:8590ms
cutoff: 10480000                10times, Time:8654ms
cutoff: 11280000                10times, Time:8765ms
cutoff: 12080000                10times, Time:8634ms
cutoff: 12880000                10times, Time:8164ms
cutoff: 13680000                10times, Time:8233ms
cutoff: 14480000                10times, Time:8331ms
cutoff: 15280000                10times, Time:8436ms
cutoff: 16080000                10times, Time:14811ms
Pool parallelism: 256
cutoff: 80000           10times, Time:5360ms
cutoff: 880000          10times, Time:4741ms
cutoff: 1680000         10times, Time:4897ms
cutoff: 2480000         10times, Time:5454ms
cutoff: 3280000         10times, Time:5307ms
cutoff: 4080000         10times, Time:6386ms
cutoff: 4880000         10times, Time:6474ms
cutoff: 5680000         10times, Time:6181ms
cutoff: 6480000         10times, Time:6419ms
cutoff: 7280000         10times, Time:6501ms
cutoff: 8080000         10times, Time:8876ms
cutoff: 8880000         10times, Time:8730ms
cutoff: 9680000         10times, Time:8696ms
cutoff: 10480000                10times, Time:9107ms
```



```
Pool parallelism: 256
cutoff: 80000           10times, Time:5360ms
cutoff: 880000          10times, Time:4741ms
cutoff: 1680000         10times, Time:4897ms
cutoff: 2480000         10times, Time:5454ms
cutoff: 3280000         10times, Time:5307ms
cutoff: 4080000         10times, Time:6386ms
cutoff: 4880000         10times, Time:6474ms
cutoff: 5680000         10times, Time:6181ms
cutoff: 6480000         10times, Time:6419ms
cutoff: 7280000         10times, Time:6501ms
cutoff: 8080000         10times, Time:8876ms
cutoff: 8880000         10times, Time:8730ms
cutoff: 9680000         10times, Time:8696ms
cutoff: 10480000                10times, Time:9107ms
cutoff: 11280000                10times, Time:9191ms
cutoff: 12080000                10times, Time:8861ms
cutoff: 12880000                10times, Time:8844ms
cutoff: 13680000                10times, Time:8879ms
cutoff: 14480000                10times, Time:8759ms
cutoff: 15280000                10times, Time:8863ms
cutoff: 16080000                10times, Time:14253ms
Pool parallelism: 1024
cutoff: 80000           10times, Time:5237ms
cutoff: 880000          10times, Time:4627ms
cutoff: 1680000         10times, Time:4613ms
cutoff: 2480000         10times, Time:5235ms
cutoff: 3280000         10times, Time:5110ms
cutoff: 4080000         10times, Time:6360ms
cutoff: 4880000         10times, Time:6254ms
cutoff: 5680000         10times, Time:5960ms
cutoff: 6480000         10times, Time:6506ms
cutoff: 7280000         10times, Time:6121ms
cutoff: 8080000         10times, Time:8407ms
cutoff: 8880000         10times, Time:8564ms
cutoff: 9680000         10times, Time:8851ms
cutoff: 10480000                10times, Time:8711ms
cutoff: 11280000                10times, Time:8682ms
```



```
cutoff: 4880000         10times, Time:6254ms
cutoff: 5680000         10times, Time:5960ms
cutoff: 6480000         10times, Time:6506ms
cutoff: 7280000         10times, Time:6121ms
cutoff: 8080000         10times, Time:8407ms
cutoff: 8880000         10times, Time:8564ms
cutoff: 9680000         10times, Time:8851ms
cutoff: 10480000                10times, Time:8711ms
cutoff: 11280000                10times, Time:8682ms
cutoff: 12080000                10times, Time:8721ms
cutoff: 12880000                10times, Time:8743ms
cutoff: 13680000                10times, Time:8598ms
cutoff: 14480000                10times, Time:9257ms
cutoff: 15280000                10times, Time:8953ms
cutoff: 16080000                10times, Time:14874ms
Pool parallelism: 4096
cutoff: 80000           10times, Time:5456ms
cutoff: 880000          10times, Time:4959ms
cutoff: 1680000         10times, Time:4806ms
cutoff: 2480000         10times, Time:5335ms
cutoff: 3280000         10times, Time:5157ms
cutoff: 4080000         10times, Time:6302ms
cutoff: 4880000         10times, Time:6127ms
cutoff: 5680000         10times, Time:6401ms
cutoff: 6480000         10times, Time:6528ms
cutoff: 7280000         10times, Time:6124ms
cutoff: 8080000         10times, Time:8744ms
cutoff: 8880000         10times, Time:8707ms
cutoff: 9680000         10times, Time:8752ms
cutoff: 10480000                10times, Time:8517ms
cutoff: 11280000                10times, Time:8554ms
cutoff: 12080000                10times, Time:8313ms
cutoff: 12880000                10times, Time:8765ms
cutoff: 13680000                10times, Time:8732ms
cutoff: 14480000                10times, Time:8483ms
cutoff: 15280000                10times, Time:8383ms
cutoff: 16080000                10times, Time:14404ms
```

Graphical Representation

Following are the graphs plotted between cut off (c) and time (in ms), with cut off (c) along the x axis and time (in ms) along the y axis.

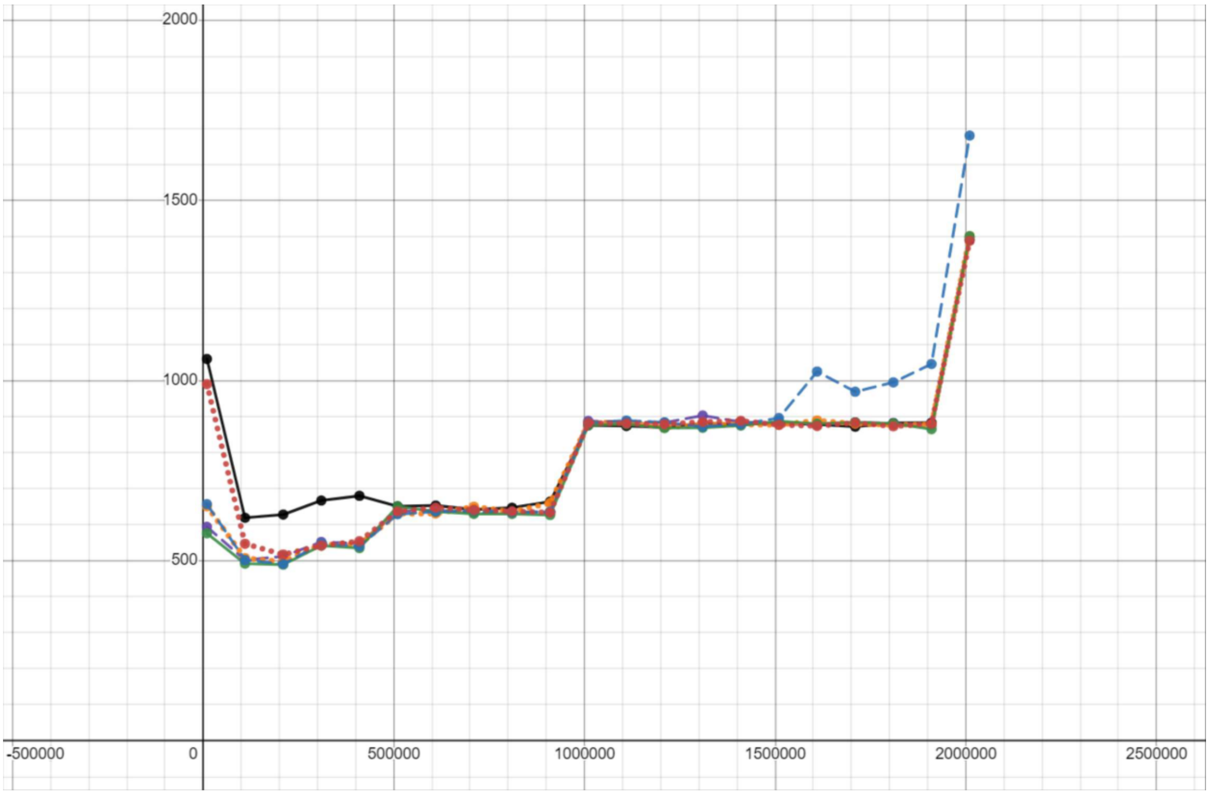
Following is the legend that applies on each graph:

- **Black solid** line denotes thread count or parallelization of **4**
- **Purple dashed** line denotes thread count or parallelization of **16**
- **Orange dotted** line denotes thread count or parallelization of **64**
- **Green solid** line denotes thread count or parallelization of **256**
- **Blue dashed** line denotes thread count or parallelization of **1024**
- **Red dotted** line denotes thread count or parallelization of **4096**

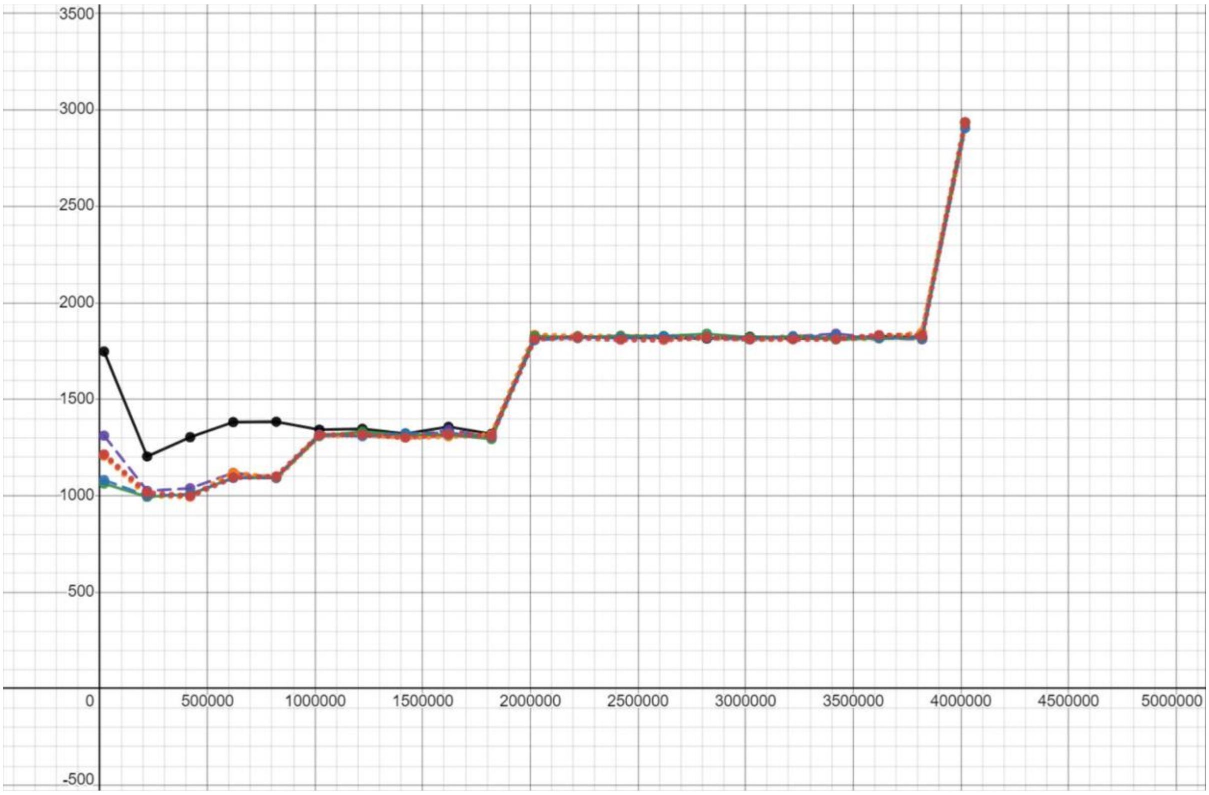
(Check individual graph files for better visualization)

(Desmos Graphing Calculator was used to plot the points and create the graph: Desmos | Graphing Calculator)

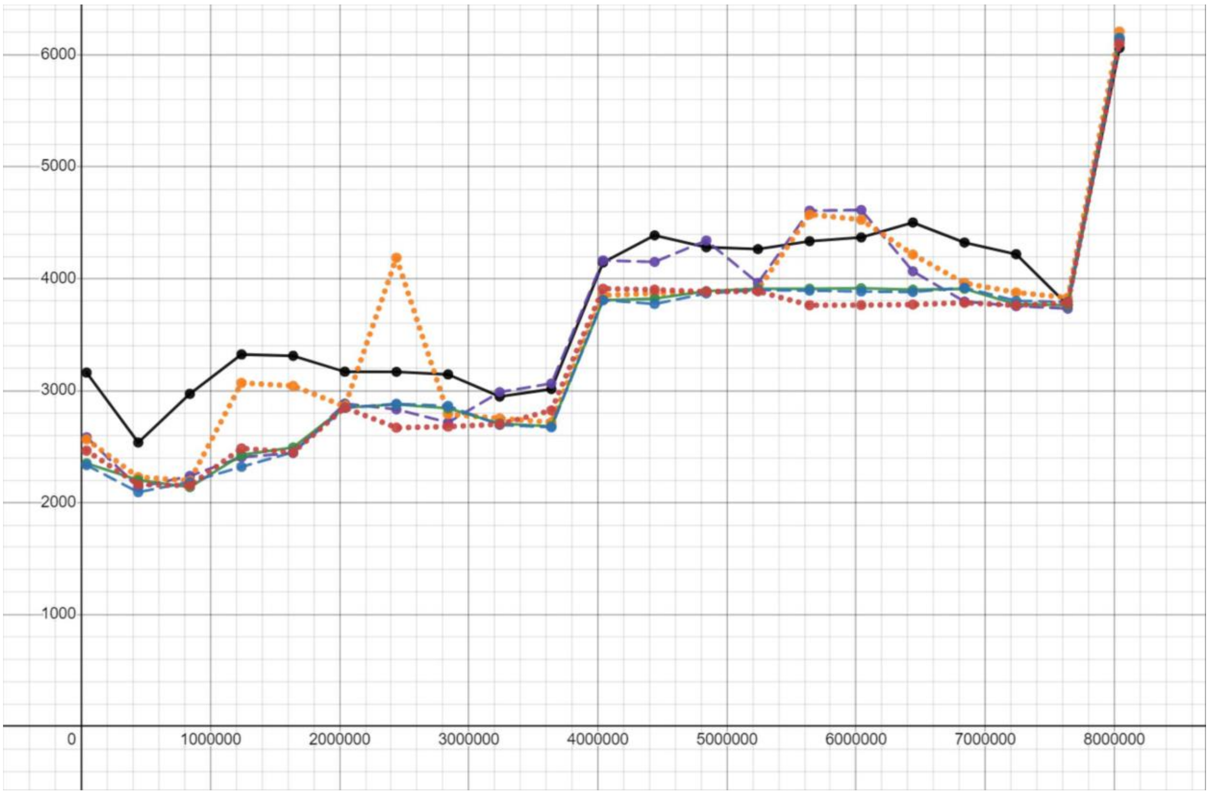
Array Size = 2000000



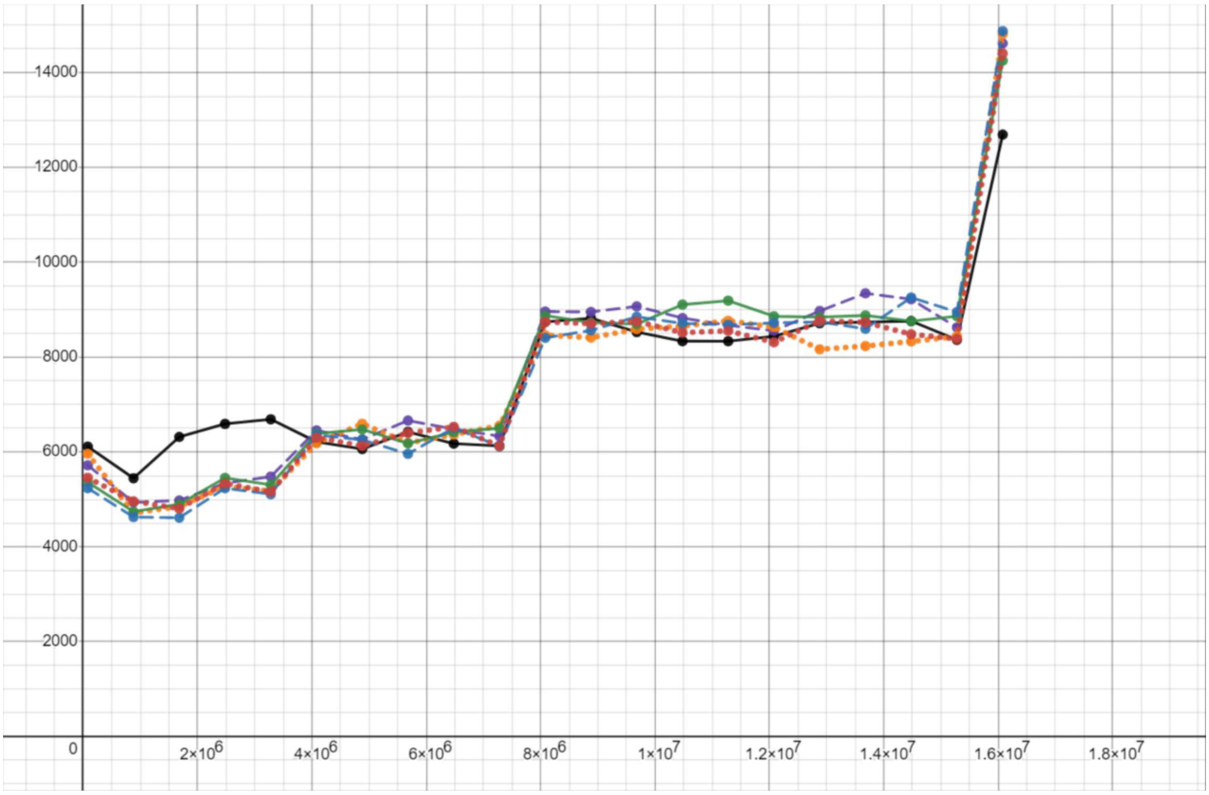
Array Size = 4000000



Array Size = 8000000



Array Size = 16000000



All the graph shows a similar pattern, wherein the graph starts high, then reduces to reach best case in its segment then starts to climb, then levels off for a while, starts to climb again and levels off again for a while until it reaches the threshold (which is cut off higher than array size) after which system sort happens and there is massive spike in time.

This indicates that there is a pattern of growth and levelling off until cut off equals array size. The best cut off point can still be considered at around $\frac{n}{10}$.

These graphs also indicate that there is not a massive improvement as thread count increases, so going above thread count of 1024 or 2^{10} may not bring significantly better results and a simple initial jump going from 2^2 (or 4) to 2^4 (or 16) can bring the most improvement.