INFO6205 Assignment 5 (Parallel Sorting)

NAME: Bohan Feng
NUID: 001564249
Repository: https://github.com/fengb3/INFO6205

Output result

the console output looks like this

```
Array Size: 2000000
    Thread Count: 1
        cutoff: 20000   10 times, time usage: 514
        cutoff: 40000   10 times, time usage: 246
        cutoff: 60000   10 times, time usage: 274
        cutoff: 80000   10 times, time usage: 289
        cutoff: 100000  10 times, time usage: 272
        cutoff: 120000  10 times, time usage: 171
        cutoff: 140000  10 times, time usage: 197
        cutoff: 160000  10 times, time usage: 185
        cutoff: 180000  10 times, time usage: 185
        cutoff: 200000  10 times, time usage: 180
        cutoff: 220000  10 times, time usage: 167
        cutoff: 240000  10 times, time usage: 182
        cutoff: 260000  10 times, time usage: 184
        cutoff: 280000  10 times, time usage: 171
        cutoff: 300000  10 times, time usage: 188
        cutoff: 320000  10 times, time usage: 172
        cutoff: 340000  10 times, time usage: 182
        cutoff: 360000  10 times, time usage: 198
        cutoff: 380000  10 times, time usage: 151
        cutoff: 400000  10 times, time usage: 170
        cutoff: 420000  10 times, time usage: 176
        cutoff: 440000  10 times, time usage: 191
        cutoff: 460000  10 times, time usage: 170
        cutoff: 480000  10 times, time usage: 193
        cutoff: 500000  10 times, time usage: 171
        cutoff: 520000  10 times, time usage: 155
        cutoff: 540000  10 times, time usage: 178
        cutoff: 560000  10 times, time usage: 179
        cutoff: 580000  10 times, time usage: 179
        cutoff: 600000  10 times, time usage: 196
        cutoff: 620000  10 times, time usage: 187
        cutoff: 640000  10 times, time usage: 179
        cutoff: 660000  10 times, time usage: 182
        cutoff: 680000  10 times, time usage: 177
        cutoff: 700000  10 times, time usage: 154
        cutoff: 720000  10 times, time usage: 177
        cutoff: 740000  10 times, time usage: 153
        cutoff: 760000  10 times, time usage: 177
        cutoff: 780000  10 times, time usage: 176
        cutoff: 800000  10 times, time usage: 154
        cutoff: 820000  10 times, time usage: 178
        cutoff: 840000  10 times, time usage: 180
        cutoff: 860000  10 times, time usage: 181
        cutoff: 880000  10 times, time usage: 153
        cutoff: 900000  10 times, time usage: 174
        cutoff: 920000  10 times, time usage: 176
        cutoff: 940000  10 times, time usage: 154
```

```
        cutoff: 960000  10 times, time usage: 152
        cutoff: 980000  10 times, time usage: 178
        cutoff: 1000000 10 times, time usage: 175
    Thread Count: 2
        cutoff: 20000   10 times, time usage: 187
        cutoff: 40000   10 times, time usage: 177
        cutoff: 60000   10 times, time usage: 168
        cutoff: 80000   10 times, time usage: 165
        cutoff: 100000  10 times, time usage: 174
        cutoff: 120000  10 times, time usage: 174
        cutoff: 140000  10 times, time usage: 160
        cutoff: 160000  10 times, time usage: 179
        cutoff: 180000  10 times, time usage: 168
        cutoff: 200000  10 times, time usage: 176
        cutoff: 220000  10 times, time usage: 159
        cutoff: 240000  10 times, time usage: 177
        cutoff: 260000  10 times, time usage: 164
```

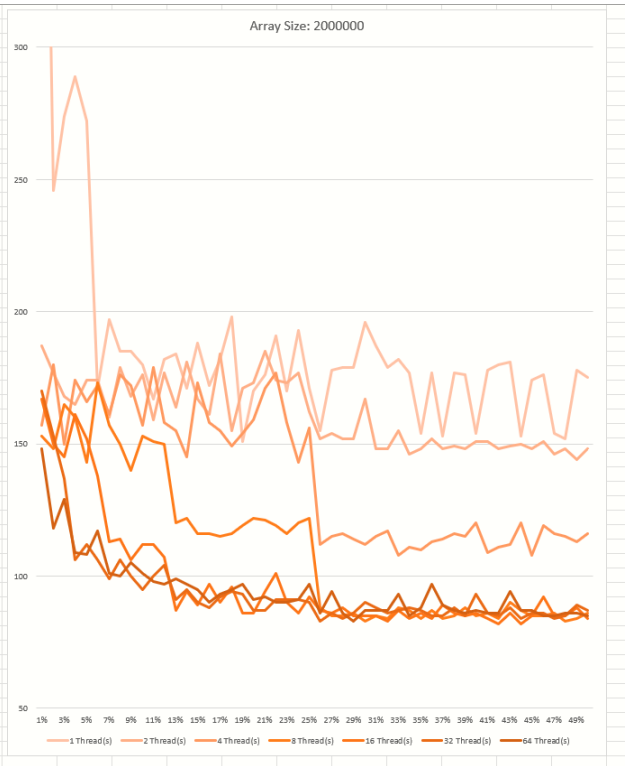
Graph

I try 5 different value for size of array (2 million to 10 million). The cutoff value increasing based on the percentage of array size.

Array Size 2,000,000

Array Size: 2000000

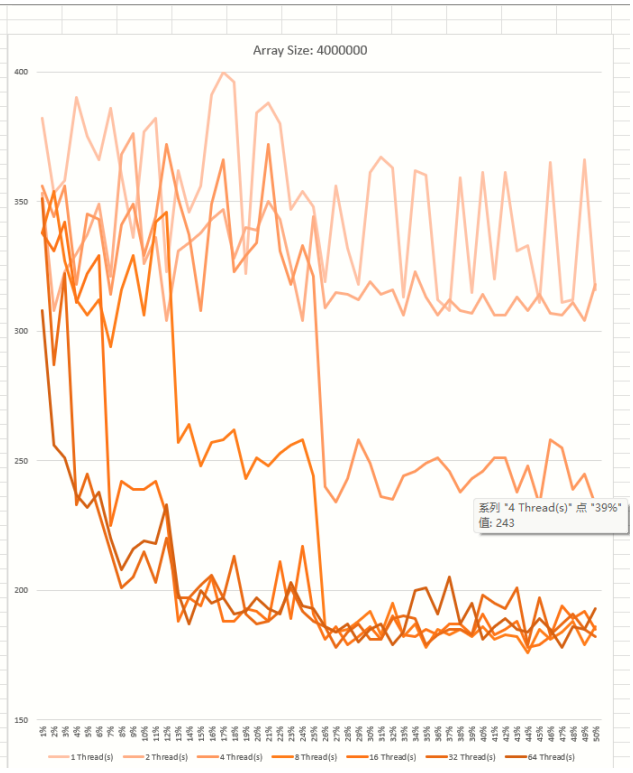
Cutoff	1 Thread(s)	2 Thread(s)	4 Thread(s)	8 Thread(s)	16 Thread(s)	32 Thread(s)	64 Thread(s)	Cutoff/arraySize
20000	514	187	157	153	167	170	148	1%
40000	246	177	180	148	150	153	118	2%
60000	274	168	150	165	145	137	129	3%
80000	289	165	174	160	161	106	109	4%
100000	272	174	166	143	152	112	108	5%
120000	171	174	172	173	138	106	117	6%
140000	197	160	161	157	113	99	101	7%
160000	185	179	176	150	114	106	100	8%
180000	185	168	172	140	106	100	105	9%
200000	180	176	157	153	112	95	101	10%
220000	167	159	179	151	112	100	98	11%
240000	182	177	158	150	107	104	97	12%
260000	184	164	155	120	87	91	99	13%
280000	171	181	145	122	94	95	97	14%
300000	188	167	173	116	89	90	95	15%
320000	172	161	158	116	97	88	90	16%
340000	182	184	155	115	90	92	93	17%
360000	198	155	149	116	96	94	95	18%
380000	151	171	154	119	86	93	97	19%
400000	170	173	159	122	86	87	91	20%
420000	176	185	171	121	94	87	92	21%
440000	191	174	177	119	101	91	90	22%
460000	170	173	158	116	90	91	90	23%
480000	193	177	143	120	86	91	91	24%
500000	171	162	156	122	92	90	97	25%
520000	155	152	112	88	87	83	86	26%
540000	178	154	115	85	86	86	94	27%
560000	179	152	116	85	88	84	86	28%
580000	179	152	114	86	85	86	83	29%
600000	196	167	112	83	85	90	87	30%
620000	187	148	115	85	85	88	87	31%
640000	179	148	117	84	83	86	87	32%
660000	182	155	108	88	87	87	93	33%
680000	177	146	111	87	84	88	85	34%
700000	154	148	110	84	86	87	88	35%
720000	177	152	113	87	84	85	97	36%
740000	153	148	114	84	89	85	89	37%
760000	177	149	116	85	86	88	87	38%
780000	176	148	115	88	85	85	86	39%
800000	154	151	120	85	86	93	87	40%
820000	178	151	109	86	84	86	86	41%
840000	180	148	111	84	82	85	86	42%
860000	181	149	112	90	86	88	94	43%
880000	153	150	120	87	82	84	87	44%
900000	174	148	108	85	85	86	87	45%
920000	176	151	119	85	92	86	85	46%
940000	154	146	116	86	85	84	85	47%
960000	152	148	115	83	85	85	86	48%
980000	178	144	113	84	88	89	86	49%
1000000	175	148	116	86	84	87	85	50%



Array Size 4,000,000

Array Size: 4000000

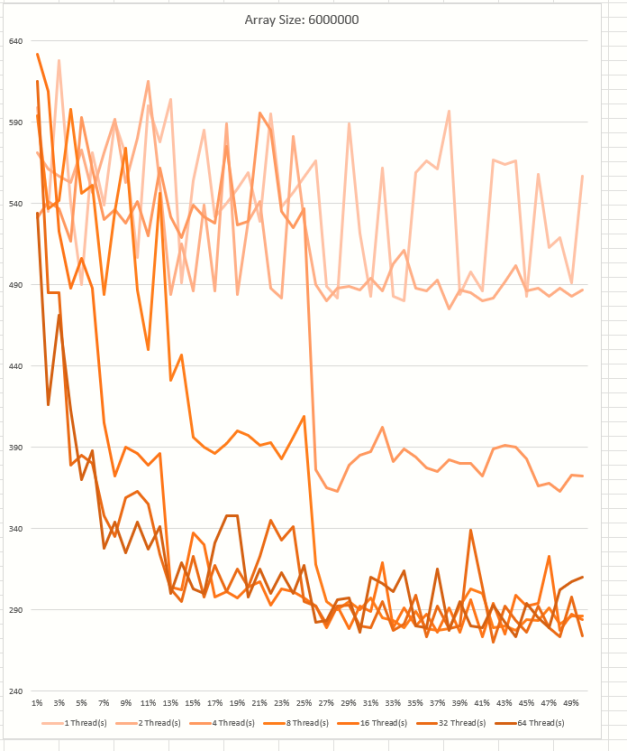
Cutoff	1 Thread(s)	2 Thread(s)	4 Thread(s)	8 Thread(s)	16 Thread(s)	32 Thread(s)	64 Thread(s)	Cutoff/arraySize
40000	382	353	356	338	338	351	308	1%
80000	353	308	344	354	331	287	256	2%
120000	358	323	356	327	342	322	251	3%
160000	390	330	318	312	311	233	237	4%
200000	375	337	345	306	322	245	232	5%
240000	366	349	343	312	329	230	238	6%
280000	386	321	314	294	225	215	220	7%
320000	360	368	341	316	242	201	208	8%
360000	336	376	349	329	239	205	216	9%
400000	377	326	329	306	239	215	219	10%
440000	382	336	344	342	242	203	218	11%
480000	323	304	372	346	229	220	233	12%
520000	362	331	351	257	188	197	199	13%
560000	346	334	337	264	197	197	187	14%
600000	356	338	308	248	194	202	200	15%
640000	391	343	349	257	205	206	195	16%
680000	400	347	366	258	188	197	197	17%
720000	396	328	323	262	188	213	191	18%
760000	322	340	329	243	193	191	192	19%
800000	384	339	334	251	192	187	197	20%
840000	388	350	372	248	188	188	193	21%
880000	380	343	331	253	211	192	191	22%
920000	347	325	318	256	189	201	203	23%
960000	354	304	333	258	217	192	194	24%
1000000	348	344	321	244	191	188	193	25%
1040000	319	309	240	186	181	186	186	26%
1080000	356	315	234	184	186	178	184	27%
1120000	332	314	243	185	179	184	187	28%
1160000	318	312	258	188	182	187	180	29%
1200000	361	319	249	192	186	181	185	30%
1240000	367	314	236	183	181	181	187	31%
1280000	363	316	235	195	190	189	179	32%
1320000	313	306	244	182	183	190	184	33%
1360000	362	323	246	187	182	189	200	34%
1400000	360	313	249	178	185	179	201	35%
1440000	312	306	251	185	183	183	191	36%
1480000	308	312	246	183	187	185	205	37%
1520000	359	308	238	185	187	185	187	38%
1560000	315	307	243	182	183	183	195	39%
1600000	361	314	246	186	191	198	181	40%
1640000	320	306	251	181	183	195	186	41%
1680000	361	306	251	183	185	193	189	42%
1720000	331	313	238	182	188	201	185	43%
1760000	333	308	248	176	178	179	184	44%
1800000	311	314	233	185	179	197	189	45%
1840000	365	307	258	181	182	183	185	46%
1880000	311	306	255	184	194	187	178	47%
1920000	312	311	239	188	189	191	186	48%
1960000	366	304	245	179	192	185	185	49%
2000000	316	318	233	186	185	182	193	50%



Array Size 6,000,000

Array Size: 6000000

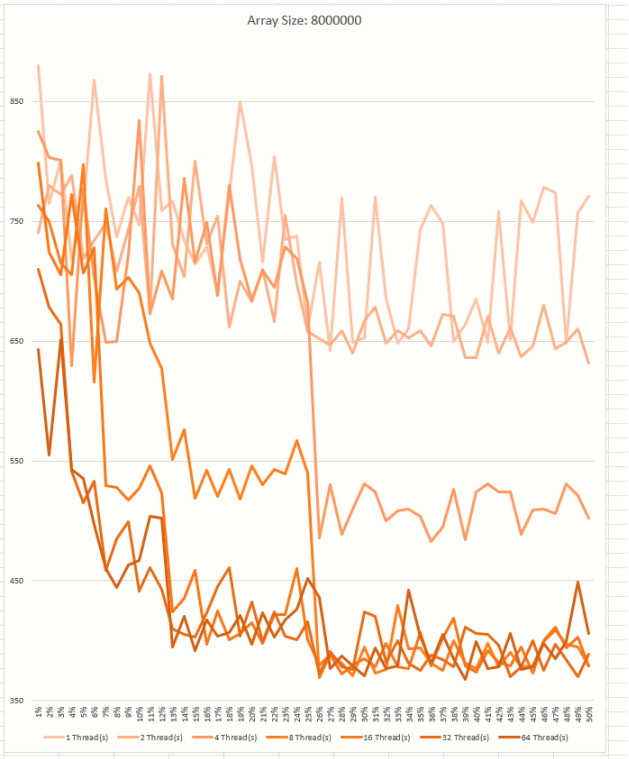
Cutoff	1 Thread(s)	2 Thread(s)	4 Thread(s)	8 Thread(s)	16 Thread(s)	32 Thread(s)	64 Thread(s)	Cutoff/arraySize
60000	599	571	531	594	632	615	534	1%
120000	535	561	541	537	609	485	416	2%
180000	628	557	537	542	523	485	471	3%
240000	540	553	517	598	488	379	413	4%
300000	490	573	593	546	506	385	370	5%
360000	571	547	561	551	488	380	388	6%
420000	539	571	530	484	485	348	328	7%
480000	590	592	536	535	372	335	344	8%
540000	570	553	528	574	390	359	325	9%
600000	507	580	541	487	386	363	344	10%
660000	600	615	520	450	379	355	327	11%
720000	578	552	562	546	386	324	341	12%
780000	604	484	532	431	384	303	300	13%
840000	491	515	519	447	302	295	319	14%
900000	554	486	539	396	337	323	303	15%
960000	585	539	532	390	330	298	300	16%
1020000	532	486	528	386	298	317	331	17%
1080000	540	589	575	392	301	301	348	18%
1140000	549	484	527	400	297	315	348	19%
1200000	559	529	529	397	304	304	298	20%
1260000	529	541	596	391	307	323	315	21%
1320000	595	488	585	393	293	345	300	22%
1380000	538	482	535	383	303	333	313	23%
1440000	547	581	525	396	301	341	300	24%
1500000	556	530	537	409	297	295	317	25%
1560000	566	490	376	318	292	292	282	26%
1620000	489	480	365	295	279	281	283	27%
1680000	482	488	363	290	292	292	296	28%
1740000	589	489	379	295	278	293	297	29%
1800000	522	487	385	290	292	280	276	30%
1860000	483	494	387	297	289	279	310	31%
1920000	562	486	402	285	319	295	306	32%
1980000	483	503	381	283	279	277	301	33%
2040000	480	511	389	279	291	281	314	34%
2100000	559	488	384	289	280	299	280	35%
2160000	566	486	377	278	287	273	279	36%
2220000	561	493	375	277	276	292	315	37%
2280000	597	475	382	278	291	278	277	38%
2340000	484	487	380	292	276	280	295	39%
2400000	498	485	380	303	296	339	280	40%
2460000	486	480	372	300	273	304	279	41%
2520000	567	482	389	279	294	270	293	42%
2580000	564	492	391	280	275	292	282	43%
2640000	566	502	390	277	299	283	273	44%
2700000	483	486	383	284	292	276	294	45%
2760000	558	488	366	283	294	292	285	46%
2820000	513	483	368	291	323	279	279	47%
2880000	519	488	363	281	277	273	302	48%
2940000	491	483	373	286	287	298	307	49%
3000000	557	487	372	286	284	274	310	50%



Array Size 8,000,000

Array Size: 8000000

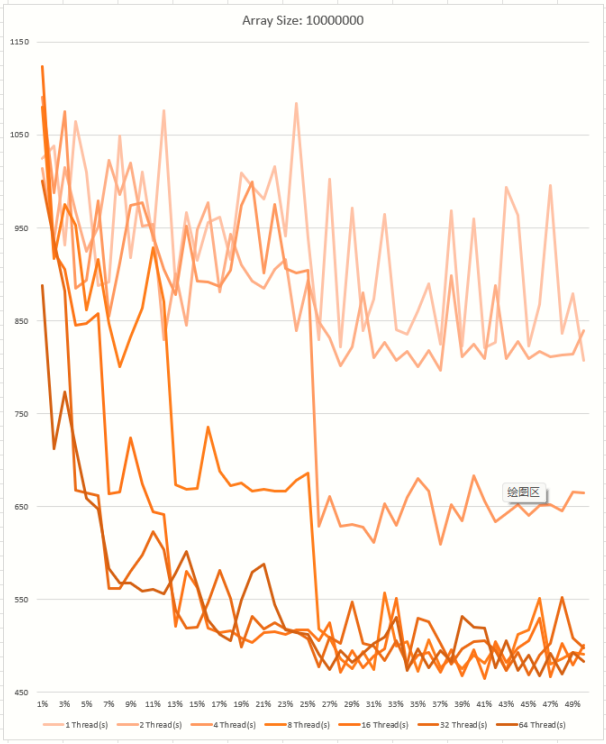
Cutoff	1 Thread(s)	2 Thread(s)	4 Thread(s)	8 Thread(s)	16 Thread(s)	32 Thread(s)	64 Thread(s)	Cutoff/arraySize
80000	880	741	825	763	799	710	643	1%
160000	765	780	803	750	724	678	555	2%
240000	801	772	801	715	705	664	651	3%
320000	718	788	629	705	772	541	543	4%
400000	760	719	777	797	707	515	535	5%
480000	868	734	703	616	728	533	497	6%
560000	785	748	649	760	529	459	460	7%
640000	737	708	650	693	528	485	444	8%
720000	770	743	722	703	517	499	463	9%
800000	747	779	834	690	527	441	467	10%
880000	873	673	673	648	546	461	504	11%
960000	759	871	708	627	523	443	502	12%
1040000	767	731	685	551	424	410	395	13%
1120000	734	704	786	576	435	405	420	14%
1200000	714	800	716	519	459	403	392	15%
1280000	729	731	749	542	397	423	417	16%
1360000	690	754	688	520	425	445	404	17%
1440000	773	662	780	543	401	461	407	18%
1520000	850	700	718	518	406	404	421	19%
1600000	796	683	684	546	415	432	397	20%
1680000	716	710	709	530	398	399	423	21%
1760000	804	666	695	543	422	424	403	22%
1840000	735	755	729	539	422	404	417	23%
1920000	738	698	719	567	460	401	426	24%
2000000	660	658	682	540	401	416	452	25%
2080000	716	652	486	369	380	372	436	26%
2160000	642	647	530	389	388	391	377	27%
2240000	769	659	489	381	372	378	387	28%
2320000	649	640	510	371	380	376	379	29%
2400000	653	667	531	395	385	424	371	30%
2480000	770	678	524	373	378	420	394	31%
2560000	686	648	500	376	398	382	377	32%
2640000	648	659	508	429	378	400	379	33%
2720000	660	653	510	393	377	381	442	34%
2800000	743	659	504	394	407	375	404	35%
2880000	763	646	483	381	379	388	381	36%
2960000	748	672	495	375	402	384	405	37%
3040000	650	671	526	400	419	378	385	38%
3120000	664	636	484	381	379	411	368	39%
3200000	685	636	524	377	374	406	399	40%
3280000	649	671	531	398	392	405	377	41%
3360000	758	640	524	378	382	396	378	42%
3440000	651	661	524	390	379	370	406	43%
3520000	767	637	489	378	395	378	376	44%
3600000	749	646	509	379	373	400	378	45%
3680000	778	680	510	399	400	375	398	46%
3760000	774	644	506	409	411	397	385	47%
3840000	648	649	531	397	394	384	399	48%
3920000	757	660	521	395	403	370	449	49%
4000000	771	632	502	379	379	389	406	50%



Array Size 10,000,000

Array Size: 10000000

Cutoff	1 Thread(s)	2 Thread(s)	4 Thread(s)	8 Thread(s)	16 Thread(s)	32 Thread(s)	64 Thread(s)	Cutoff/arraySize
100000	1024	1014	1090	1080	1123	1000	888	1%
200000	1038	937	988	917	924	936	712	2%
300000	931	1015	1075	975	905	882	773	3%
400000	1064	967	885	953	845	668	714	4%
500000	1010	925	894	862	847	665	659	5%
600000	888	951	979	916	858	662	647	6%
700000	892	1023	855	847	664	562	583	7%
800000	1049	986	912	800	666	562	568	8%
900000	918	1020	974	832	724	580	568	9%
1000000	1010	952	977	863	674	598	559	10%
1100000	936	954	941	928	644	623	561	11%
1200000	1076	830	905	870	641	604	556	12%
1300000	887	900	878	673	521	539	578	13%
1400000	966	845	952	669	580	519	602	14%
1500000	915	948	893	670	563	520	565	15%
1600000	956	977	892	735	519	546	528	16%
1700000	961	881	887	688	514	581	512	17%
1800000	916	943	904	672	516	551	506	18%
1900000	1009	910	974	675	509	499	549	19%
2000000	994	893	999	667	504	532	579	20%
2100000	981	885	901	669	514	518	588	21%
2200000	1016	905	975	667	515	525	544	22%
2300000	941	916	906	667	512	518	517	23%
2400000	1084	839	901	678	517	515	514	24%
2500000	939	894	904	686	517	508	512	25%
2600000	830	849	629	518	506	478	491	26%
2700000	1002	831	661	509	525	510	475	27%
2800000	822	801	629	486	472	503	495	28%
2900000	971	822	631	476	495	547	482	29%
3000000	839	880	628	494	477	503	492	30%
3100000	873	810	611	475	489	500	503	31%
3200000	964	827	653	557	497	484	510	32%
3300000	840	807	630	500	551	506	531	33%
3400000	835	817	660	505	476	480	474	34%
3500000	861	800	680	473	490	530	497	35%
3600000	890	818	667	507	493	526	477	36%
3700000	825	797	609	477	472	503	495	37%
3800000	968	898	652	487	496	480	483	38%
3900000	823	811	635	476	468	497	532	39%
4000000	959	825	683	490	496	505	520	40%
4100000	821	809	656	481	465	506	519	41%
4200000	827	888	634	500	505	495	477	42%
4300000	993	809	642	474	482	474	506	43%
4400000	963	828	652	512	498	493	474	44%
4500000	823	809	640	517	506	469	490	45%
4600000	867	817	651	551	530	490	468	46%
4700000	995	811	652	480	467	503	492	47%
4800000	836	813	645	486	503	552	470	48%
4900000	879	814	666	493	479	509	493	49%
5000000	807	839	665	491	501	498	483	50%



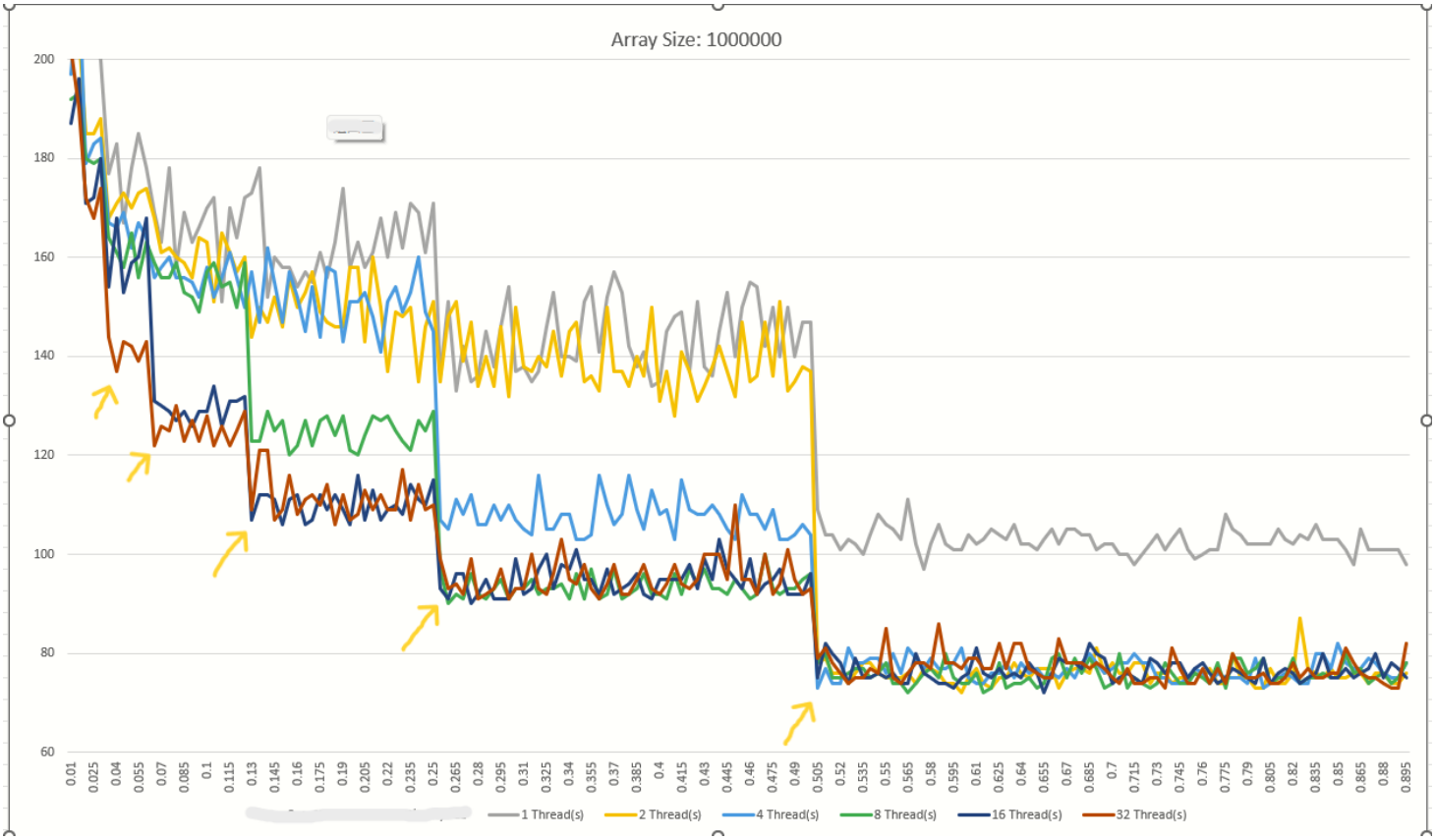
Observation

In the chart above, we can see the cutoff value is above 25% of arraysize, the parallel sorting proformence better than cutoff value smaller than 25% of arraysize.

For number of threads, it seems that the more threads we have the faster the algorthm sort the array. For threads number more than 16, there is no obvious difference.

More Observation

Then I try to extent the percentage of array size as cutoff value (1% - 90%) to see how cutoff value effect sorting speed.



In some intervals of cutoff values, the sorting time tends to be stable.

(50% - 100%], (25% - 50], (12.5% - 25%],(6.25%,-12.5%] ...

in generally, this interval is (*$1/(n+1)^2, (1/n)$] where $n = 2, 3, 4...etc$

The larger n, the lower the efficiency of sorting. When the cutoff value is greater than 50% of the array size, the sorting efficiency is the highest and tends to be stable.

Multithreading is always more efficient than single threading. When the number of threads is greater than or equal to 2, the increase of threads will bring more obvious improvement when the cutoff value is lower. As the cutoff value increases, the improvement brought by increasing the number of threads is not obvious

Conclusion

Since we are using a merge sort strategy, this strategy requires dividing the array into halfs. The number of dividing operation depends on the cutoff value. This results in increasing the dividing operation to n times when the cutoff is less than $1/n^2$, where $n >= 2$. This is why the sorting time is different between the previously mentioned intervals

When the cutoff value is smaller, in an other word when there are more partitions. Using more threads results in significant performance gains. However, whenever two adjacent partition are sorted and merged, some threads will be idle and not participate in the rest of the sorting. So when the cutoff value is too small, the sorting efficiency is low.

Since our algorithm will split the array into two parts at least once. when the cutoff value is large, there will be two threads participating in the sorting, and the remaining threads will be idle, which will result in when the cutoff is greater than 50% of the array size, no matter how many threads we set, the algorithm efficiency are not improved.

For parallel sorting, the best strategy is to **set the cutoff value greater than 50% of the array size, and use two threads.**

possible improvement

make those idle threads participate in soring rest of the array.