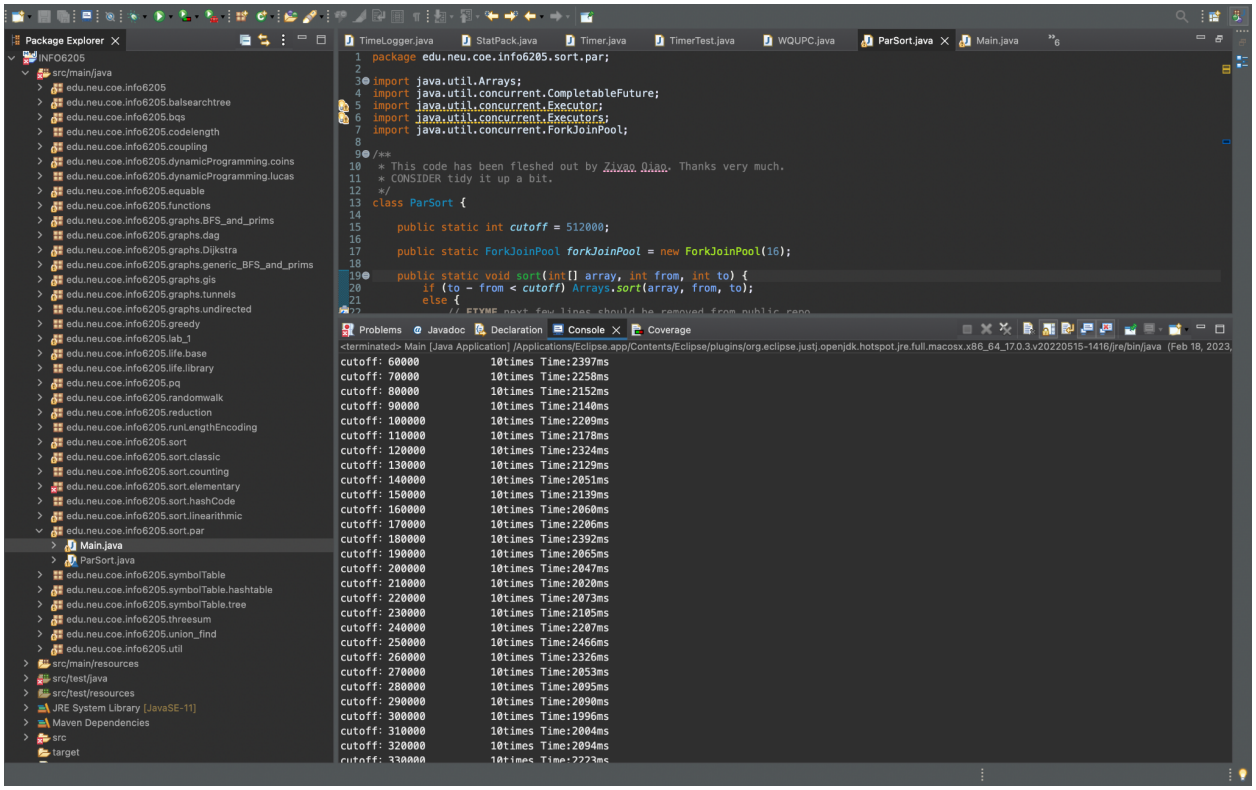
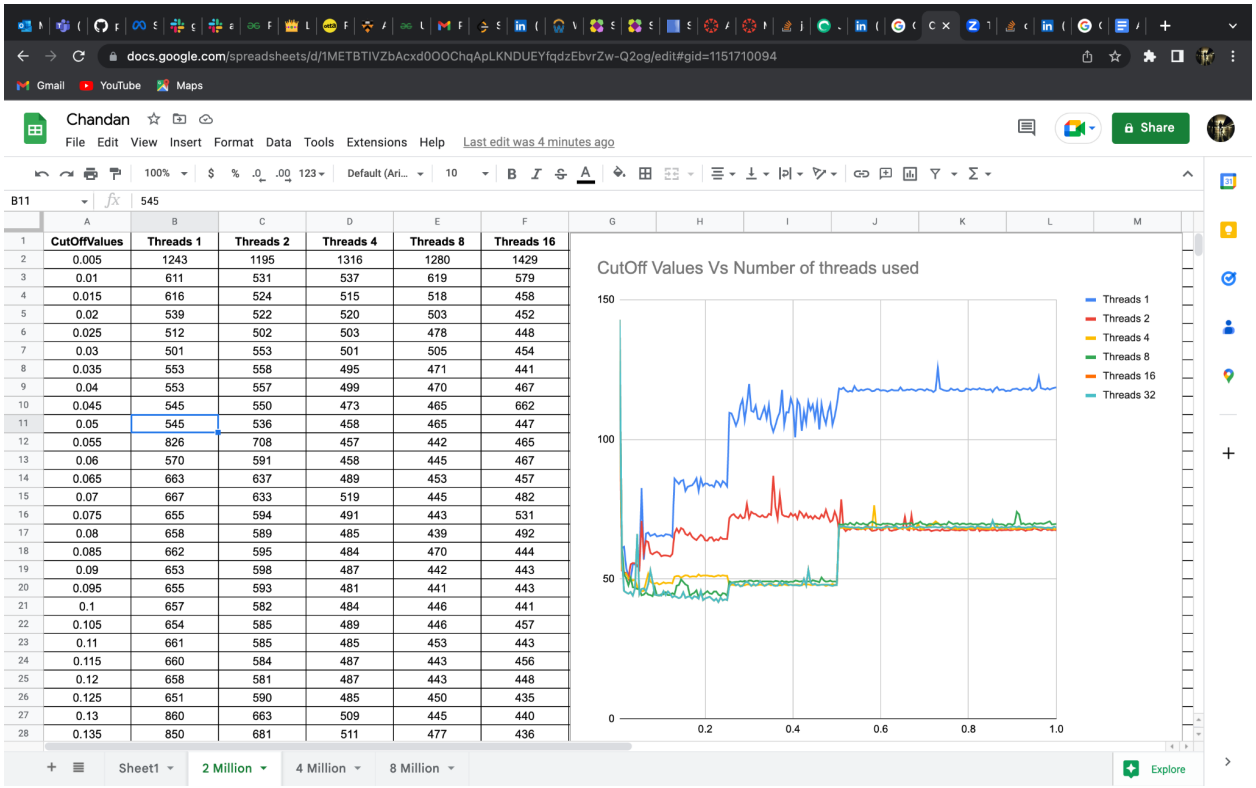
Assignment 5

Please see the presentation on Assignment on Parallel Sorting under the Exams. etc. module.
Your task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.
A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (t) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of lg t is reached).
An appropriate combination of these.
There is a Main class and the ParSort class in the sort.par package of the INFO6205 repository. The Main class can be used as is but the ParSort class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].
Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.
You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.
For varying the number of threads available, you might want to consult the following resources:
https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-poolLinks to an external site.
https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completableLinks to an external site.
Good luck and enjoy.

Answer ;

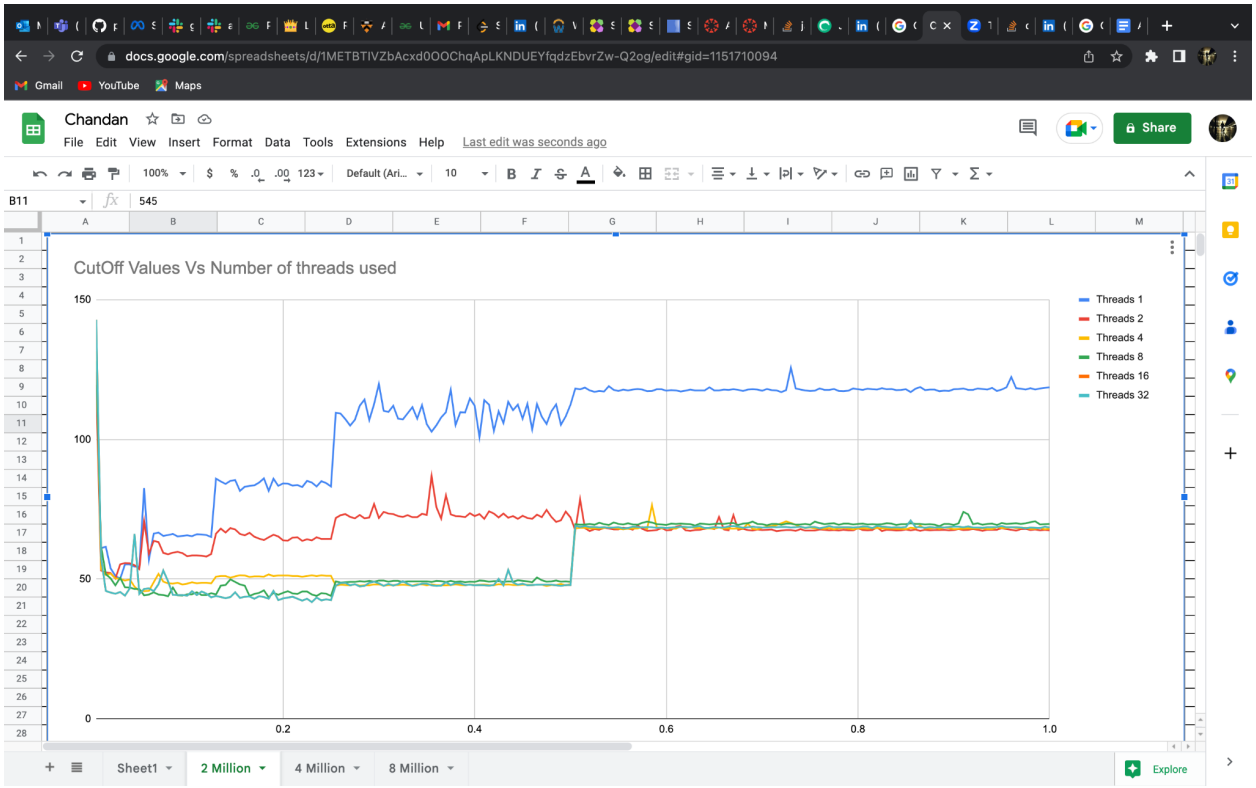


This is the sample data which was taken from the console and the csv file which was generated from the code



CutOffValues	Threads 1	Threads 2	Threads 4	Threads 8	Threads 16
0.005	1243	1195	1316	1280	1429
0.01	611	531	537	619	579
0.015	616	524	515	518	458
0.02	539	522	520	503	452
0.025	512	502	503	478	448
0.03	501	553	501	505	454
0.035	553	558	495	471	441
0.04	553	557	499	470	467
0.045	545	550	473	465	662
0.05	545	536	458	465	447
0.055	826	708	457	442	465
0.06	570	591	458	445	467
0.065	663	637	489	453	457
0.07	667	633	519	445	482
0.075	655	594	491	443	531
0.08	658	589	485	439	492
0.085	662	595	484	470	444
0.09	653	598	487	442	443
0.095	655	593	481	441	443
0.1	657	582	484	446	441
0.105	654	585	489	446	457
0.11	661	585	485	453	443
0.115	660	584	487	443	456
0.12	658	581	487	443	448
0.125	651	590	485	450	435
0.13	860	663	509	445	440
0.135	850	681	511	477	436

This is the sample data that are taken the cutoff for each trial is considered as 10% for the total size of the array for example the trial which is taking 200000 as the array size then the cut off value is around 20000 so that we can easily identify the values.

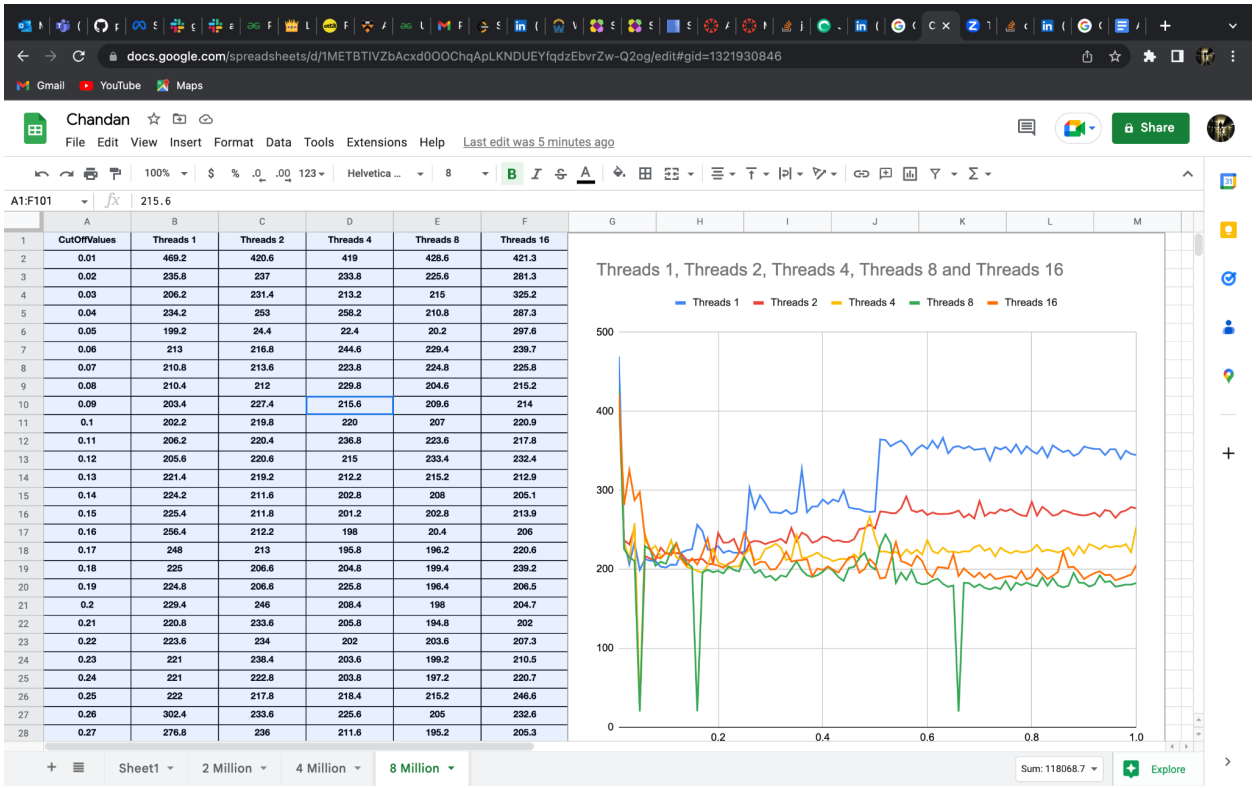


Form the above graph for a array size of 2million we have taken the cut off as 10000 which is equal to 5% for the total array.

From the above graph we can see that after increasing the number of threads we can see there is no improvement in the performance and therefore the value remains almost constant and the time taken by thread 4 and thread taken almost similar time to execute.

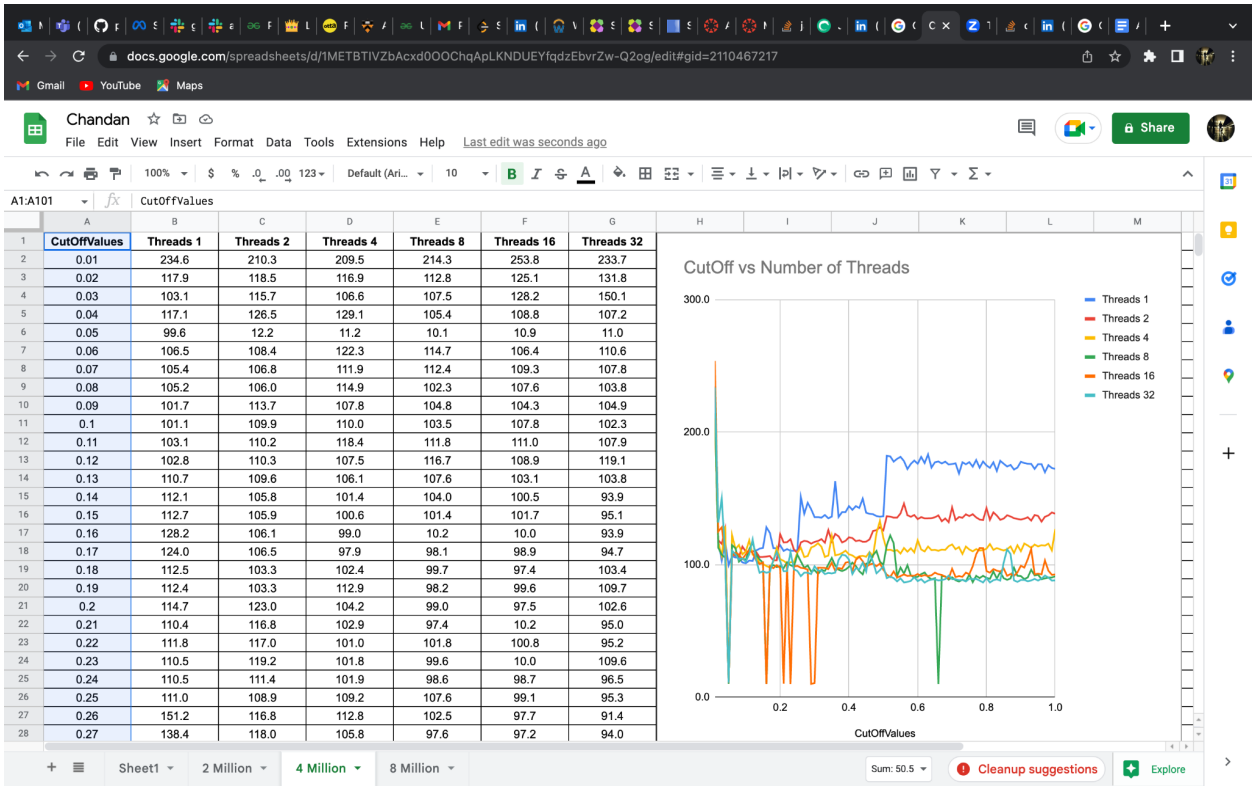
From the above graph the time taken by the single thread takes more time and compared to the other threads where the initial cutoff values are small when the number of threads keeps on increasing the time taken will decrease only to a particular point and later on the graph when the size of the array is large and the number of threads will not make any difference for larger arrays usually the array size when it crosses the cut off value which is around 50%, of the total array size and the number of thread which is idea will be close to 8 threads where after the 8 threads the performance of 16 and 32 threads are almost the same.

These are the data :



CutOffValues	Threads 1	Threads 2	Threads 4	Threads 8	Threads 16
0.01	469.2	420.6	419	428.6	421.3
0.02	235.8	237	233.8	225.6	281.3
0.03	206.2	231.4	213.2	215	325.2
0.04	234.2	253	258.2	210.8	287.3
0.05	199.2	24.4	22.4	20.2	297.6
0.06	213	216.8	244.6	229.4	239.7
0.07	210.8	213.6	223.8	224.8	225.8
0.08	210.4	212	229.8	204.6	215.2
0.09	203.4	227.4	215.6	209.6	214
0.1	202.2	219.8	220	207	220.9
0.11	206.2	220.4	236.8	223.6	217.8
0.12	205.6	220.6	215	233.4	232.4
0.13	221.4	219.2	212.2	215.2	212.9
0.14	224.2	211.6	202.8	208	205.1
0.15	225.4	211.8	201.2	202.8	213.9
0.16	256.4	212.2	198	20.4	206
0.17	248	213	195.8	196.2	220.6
0.18	225	206.6	204.8	199.4	239.2
0.19	224.8	206.6	225.8	196.4	206.5
0.2	229.4	246	208.4	198	204.7
0.21	220.8	233.6	205.8	194.8	202
0.22	223.6	234	202	203.6	207.3
0.23	221	238.4	203.6	199.2	210.5
0.24	221	222.8	203.8	197.2	220.7
0.25	222	217.8	218.4	215.2	246.6
0.26	302.4	233.6	225.6	205	232.6
0.27	276.8	236	211.6	195.2	205.3

The above graphs which has a arrays capacity of 8 million.



CutOffValues	Threads 1	Threads 2	Threads 4	Threads 8	Threads 16	Threads 32
0.01	234.6	210.3	209.5	214.3	253.8	233.7
0.02	117.9	118.5	116.9	112.8	125.1	131.8
0.03	103.1	115.7	106.6	107.5	128.2	150.1
0.04	117.1	126.5	129.1	105.4	108.8	107.2
0.05	99.6	12.2	11.2	10.1	10.9	11.0
0.06	106.5	108.4	122.3	114.7	106.4	110.6
0.07	105.4	106.8	111.9	112.4	109.3	107.8
0.08	105.2	106.0	114.9	102.3	107.6	103.8
0.09	101.7	113.7	107.8	104.8	104.3	104.9
0.1	101.1	109.9	110.0	103.5	107.8	102.3
0.11	103.1	110.2	118.4	111.8	111.0	107.9
0.12	102.8	110.3	107.5	116.7	108.9	119.1
0.13	110.7	109.6	106.1	107.6	103.1	103.8
0.14	112.1	105.8	101.4	104.0	100.5	93.9
0.15	112.7	105.9	100.6	101.4	101.7	95.1
0.16	128.2	106.1	99.0	10.2	10.0	93.9
0.17	124.0	106.5	97.9	98.1	98.9	94.7
0.18	112.5	103.3	102.4	99.7	97.4	103.4
0.19	112.4	103.3	112.9	98.2	99.6	109.7
0.2	114.7	123.0	104.2	99.0	97.5	102.6
0.21	110.4	116.8	102.9	97.4	10.2	95.0
0.22	111.8	117.0	101.0	101.8	100.8	95.2
0.23	110.5	119.2	101.8	99.6	10.0	109.6
0.24	110.5	111.4	101.9	98.6	98.7	96.5
0.25	111.0	108.9	109.2	107.6	99.1	95.3
0.26	151.2	116.8	112.8	102.5	97.7	91.4
0.27	138.4	118.0	105.8	97.6	97.2	94.0

This is the graph obtained after changing the array size to 4 million and therefore the from the above graphs we can conclude that for a 4 million the cutoff value is around 0.5 which is 50 percent of the total array size. And the from the number of threads single threads takes more time which is fallowed by 2 threads and 4 threads and after the fourths thread the threads which are qual to and greater than 8 threads takes the same time as the rest of other threads and therefore the there is not significant increase in performance after increasing the number of threads from 8 Therefore the ideal cutoff values range is almost around 50 % and the number of threads which are good and to give maximum performance is 8 threads.