Random Walk Assignment

By Shardul Negi

(001898181)

Problem Statement

Please see the presentation on Assignment on Parallel Sorting under Course Materials/Course Documents/Exams. etc. Don't worry about the fact that it talks about "Assignment 5."
This assignment is optional. If your assignment grades are disappointing (or you have some missing) then you should do this assignment. Otherwise, do it if you have time and you think you'd enjoy it.

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.

1. 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.

2. Recursion depth or 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 depth of $\lg t$ is reached).
3. An appropriate combination of these.

Implementation Details

In the current assignment, approximation is already given by Professor. The report pertains to the usage and the derived examinations of my understanding of the code and implementation of the same.

I am using Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70Ghz 8GB RAM.

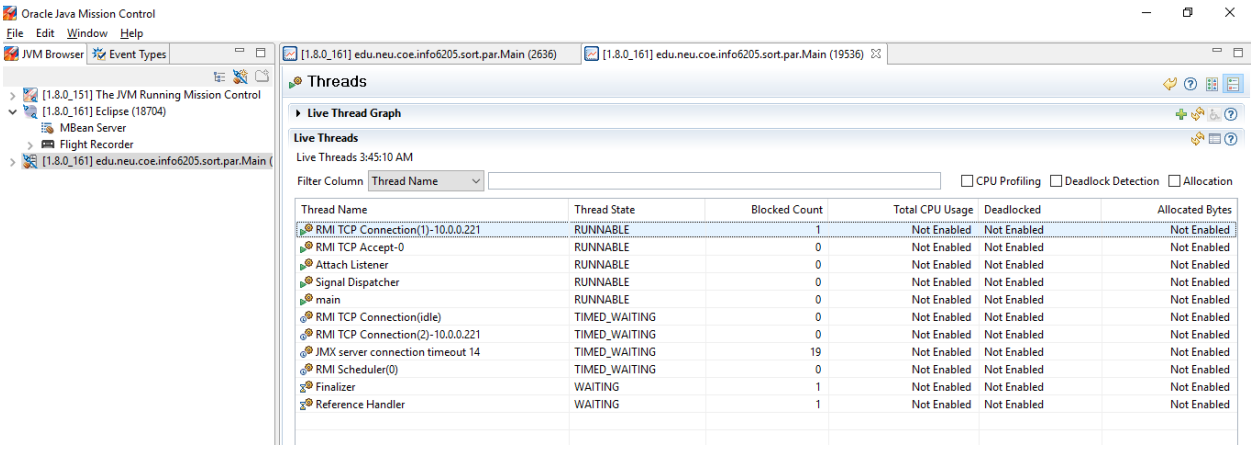
a) **Understanding of Code:** The main method assigns a cutoff to the ParSort class which has been initially assigned the value of 1000. Now, this value is increased in the iteration of 50 to 1000 and the results are further iterated 10 time to get a mean of the time taken by the process to run using currentTimeMills() function.
The loop calls a function sort to sort the randomly generated numbers using parallel sort method. The mechanism is defined by the cutoff by the expression
If size is < cutoff , which decides to use the system sort if the value is less.
Since, the cutoff is given the method assigns two aync CompletableFuture<T> which is used for threading purposes. It triggers actions depending on the completion of the actions.
In the code, the parsort function is called triggering an Aync call to the Java environment which triggers a Thread in the system for execution of the functions. A recursive call will divide the machine's time in order and divide the thread call between the various array values which need to be sorted.
Here, partsort1, will precede partsort2 and later join when completed joining the two arrays. Parsesort calls the sort method and rechecks for the cutoff value to sort which will continue forming the trigger values till the base is not encountered. We return by using **supplyAsync** which Returns a new CompletableFuture that is asynchronously completed by a task running in the given executor after it runs the given action. Hence, making use of the parallel sort we can implement and enhance the sorting of the array.

b) **Changing the Cutoff value**: Part 1 of the assignment specifies the reason of ideal value of cutoff for best computation and behavior of the cutoff value. On increasing values of the cutoff(demonstrated in future report slide) we can infer and determine the cutoff for the same.

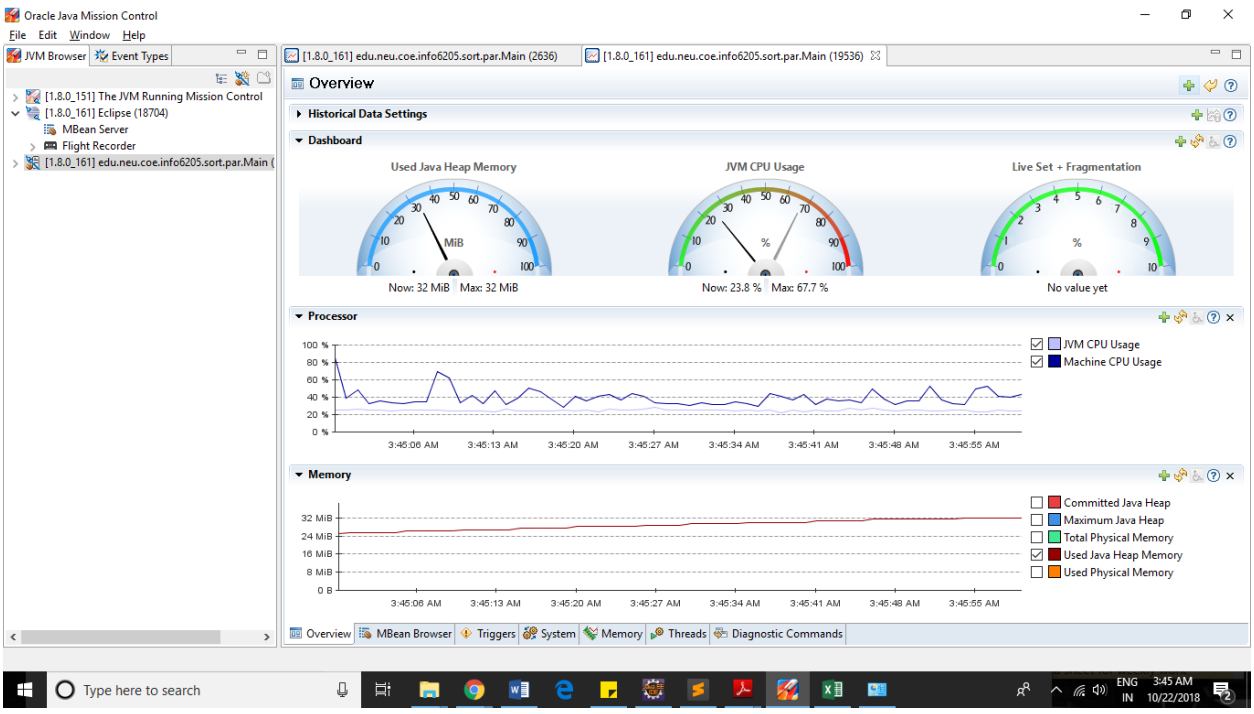
Experiments

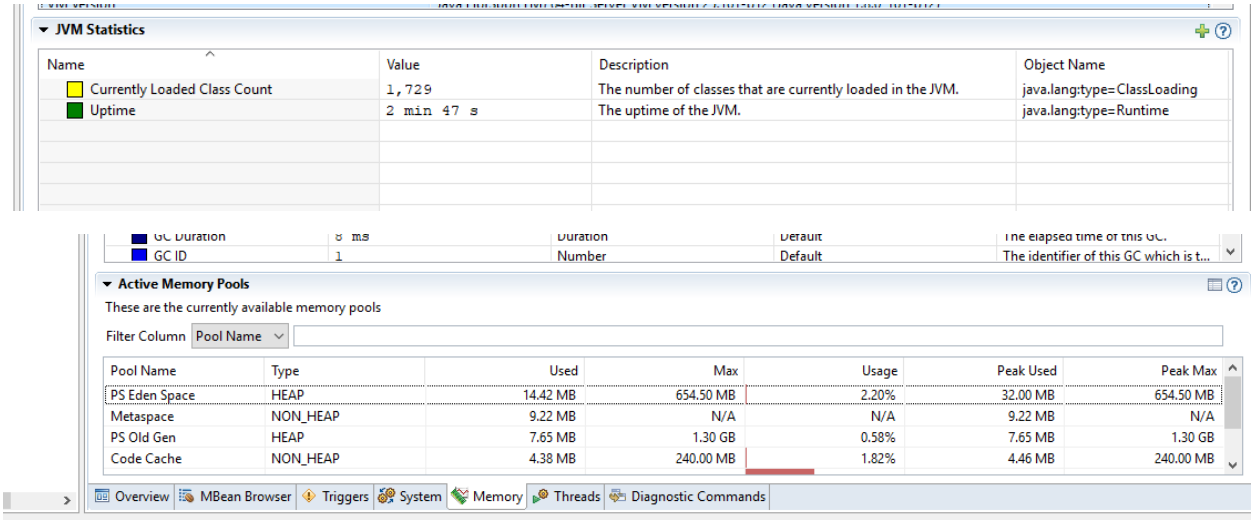
Cutoff: 65536*(j+1) :

Valuee from the JMC shows 4 runnable threads which toggle between 5 to 9 threads for the program shifting between the impementations.

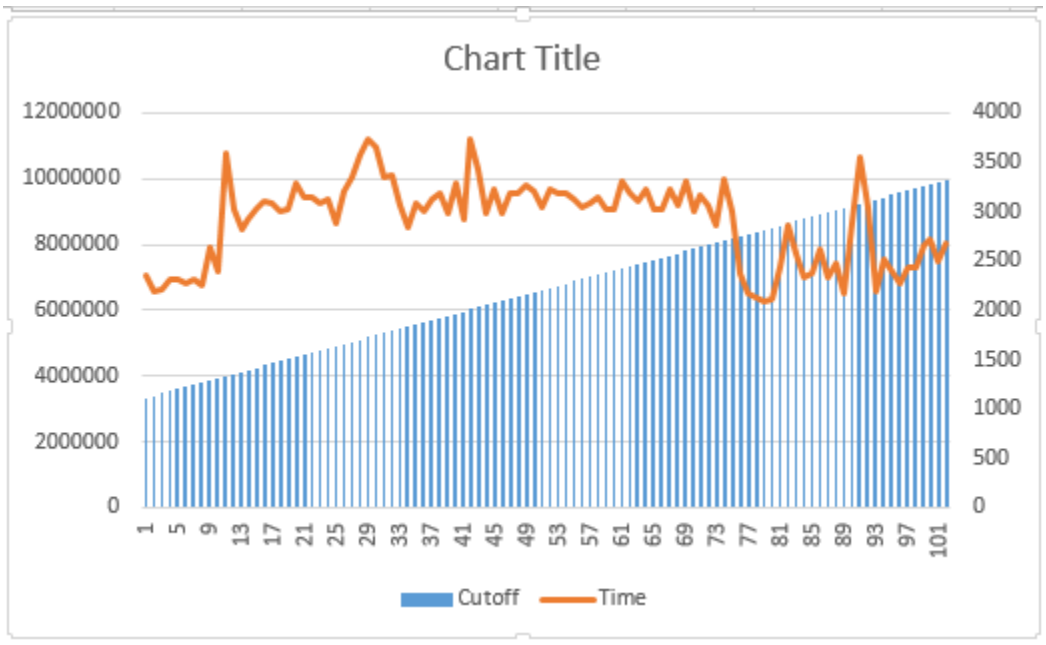


Memory usage demonstrates the heap memory, processor usage.

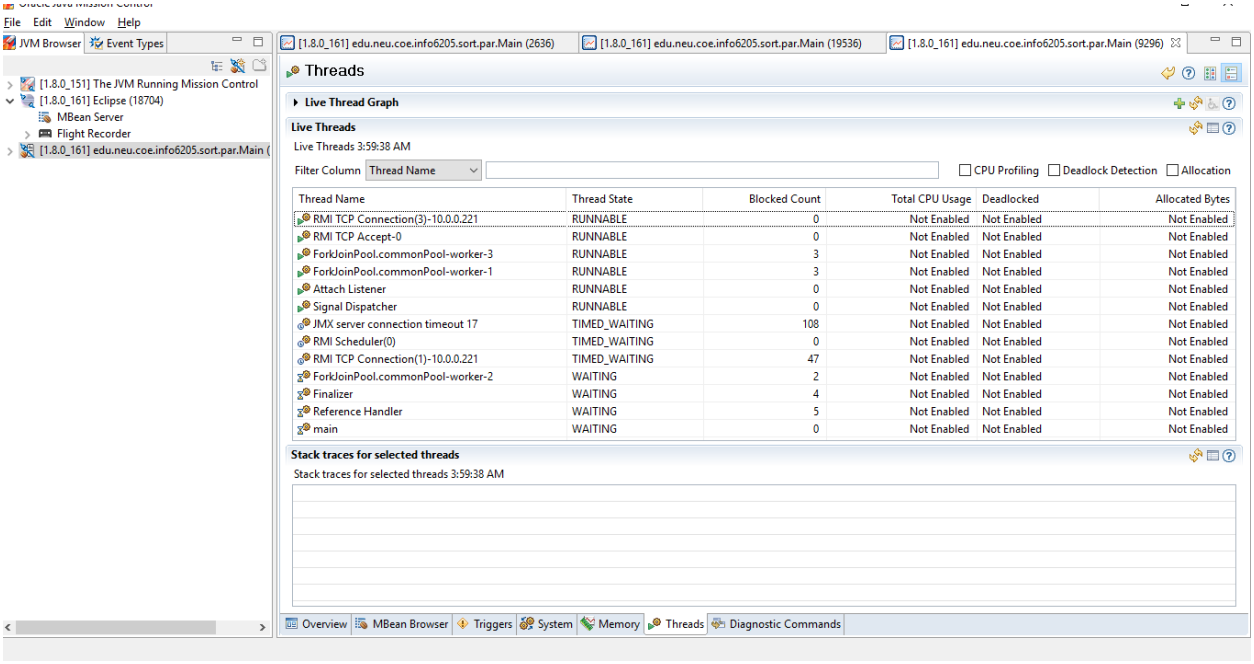


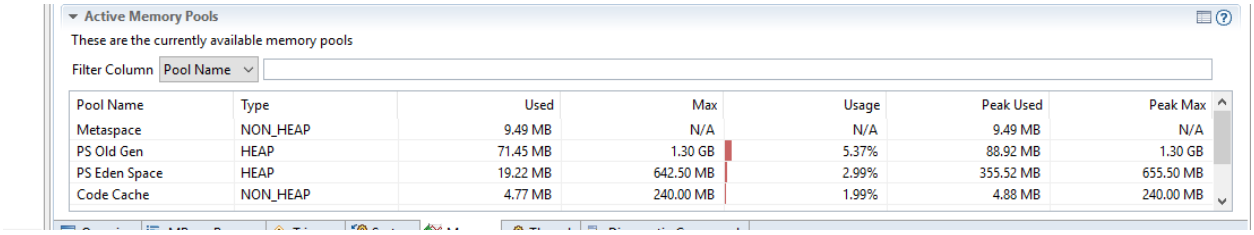


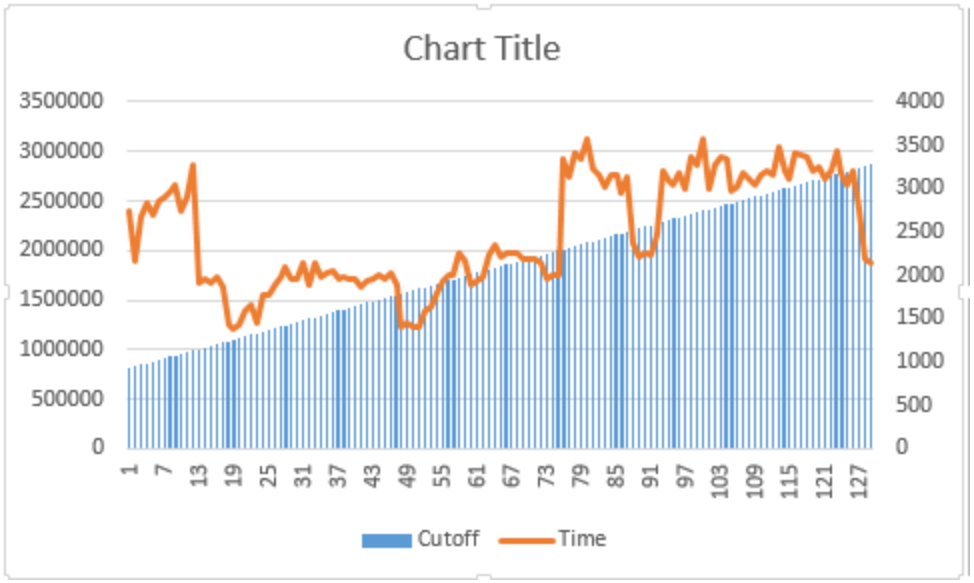
Line chart demonstrating the variation of cutoff and time in milliseconds.



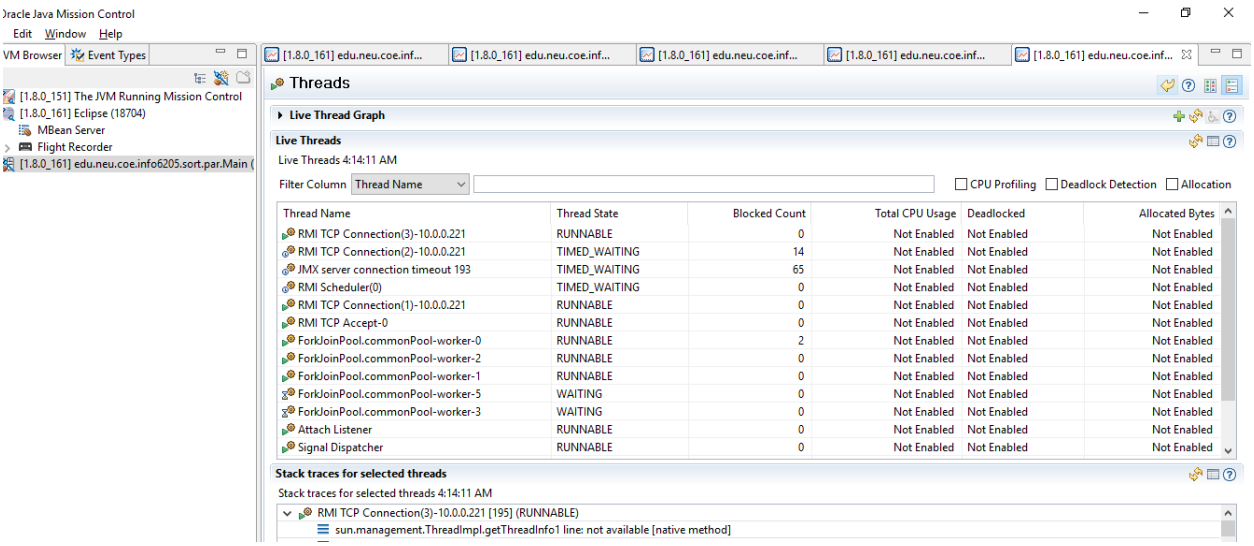
2: cutoff: 16000(j+1)


File Edit Window Help
JVM Browser
Event Types
[1.8.0_151] The JVM Running Mission Control
[1.8.0_161] Eclipse (18704)
MBean Server
Flight Recorder
[1.8.0_161] edu.neu.coe.info6205.sort.par.Main (
[1.8.0_161] edu.neu.coe.info6205.sort.par.Main (2636)
[1.8.0_161] edu.neu.coe.info6205.sort.par.Main (19536)
[1.8.0_161] edu.neu.coe.info6205.sort.par.Main (9296)
Threads
Live Thread Graph
Live Threads
Live Threads 3:59:38 AM
Filter Column
Thread Name
CPU Profiling
Deadlock Detection
Allocation
Thread Name
Thread State
Blocked Count
Total CPU Usage
Deadlocked
Allocated Bytes
RMI TCP Connection(3)-10.0.0.221 RUNNABLE 0 Not Enabled Not Enabled Not Enabled
RMI TCP Accept-0 RUNNABLE 0 Not Enabled Not Enabled Not Enabled
ForkJoinPool.commonPool-worker-3 RUNNABLE 3 Not Enabled Not Enabled Not Enabled
ForkJoinPool.commonPool-worker-1 RUNNABLE 3 Not Enabled Not Enabled Not Enabled
Attach Listener RUNNABLE 0 Not Enabled Not Enabled Not Enabled
Signal Dispatcher RUNNABLE 0 Not Enabled Not Enabled Not Enabled
JMX server connection timeout 17 TIMED_WAITING 108 Not Enabled Not Enabled Not Enabled
RMI Scheduler(0) TIMED_WAITING 0 Not Enabled Not Enabled Not Enabled
RMI TCP Connection(1)-10.0.0.221 TIMED_WAITING 47 Not Enabled Not Enabled Not Enabled
ForkJoinPool.commonPool-worker-2 WAITING 2 Not Enabled Not Enabled Not Enabled
Finalizer WAITING 4 Not Enabled Not Enabled Not Enabled
Reference Handler WAITING 5 Not Enabled Not Enabled Not Enabled
main WAITING 0 Not Enabled Not Enabled Not Enabled
Stack traces for selected threads
Stack traces for selected threads 3:59:38 AM
Overview
MBean Browser
Triggers
System
Memory
Threads
Diagnostic Commands

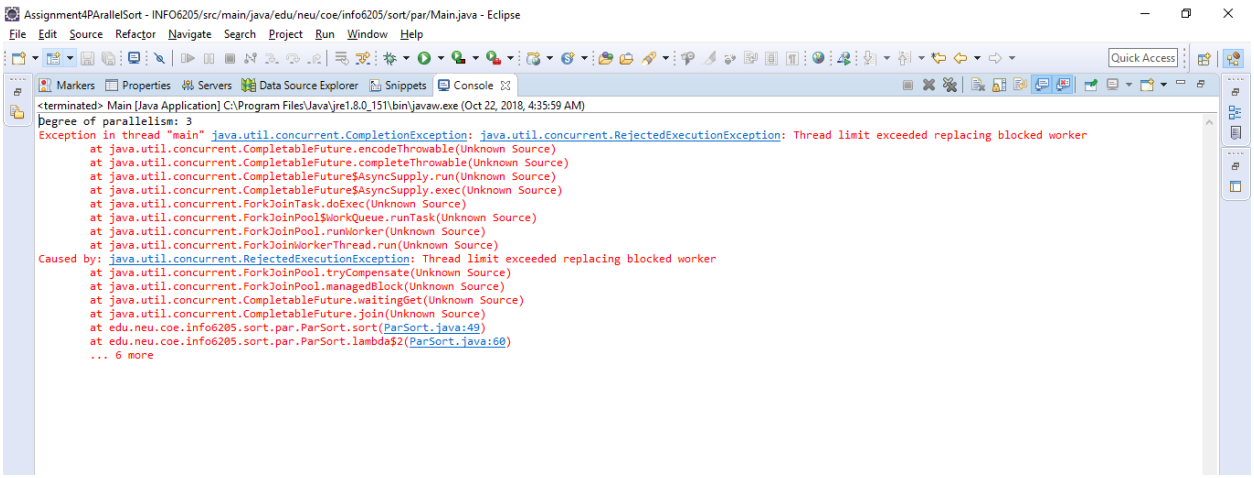

Active Memory Pools
These are the currently available memory pools
Filter Column
Pool Name
Pool Name Type Used Max Usage Peak Used Peak Max
Metaspace NON_HEAP 9.49 MB N/A N/A 9.49 MB N/A
PS Old Gen HEAP 71.45 MB 1.30 GB 5.37% 88.92 MB 1.30 GB
PS Eden Space HEAP 19.22 MB 642.50 MB 2.99% 355.52 MB 655.50 MB
Code Cache NON_HEAP 4.77 MB 240.00 MB 1.99% 4.88 MB 240.00 MB



Cutoff : 8000*(j+1)



Lastly, with 2, it throws an error



Demonstrating the overflow of the Aysnc function usage.

Conclusions

While seeing the Fork-Join Pool pattern from the JMC for mapping the CPU usage we see an ideal cutoff make from 8000 to 80000 which gives an ideal mean time of 1560 ms for all the experiments which have been saved in the project as well. The conversion of the methods can be demonstrated in this by running various values of cutoff from 10 to 100000.

The ideal cutoff range varies with processor speed but on my machine it comes for values of 8000 to 80000 cutoff to sort the array.