**Program Structures & Algorithms**

**Spring 2022**

**Assignment No. 4**

Name: Yuan Huang

(NUID): 002117054

* **Task**

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.

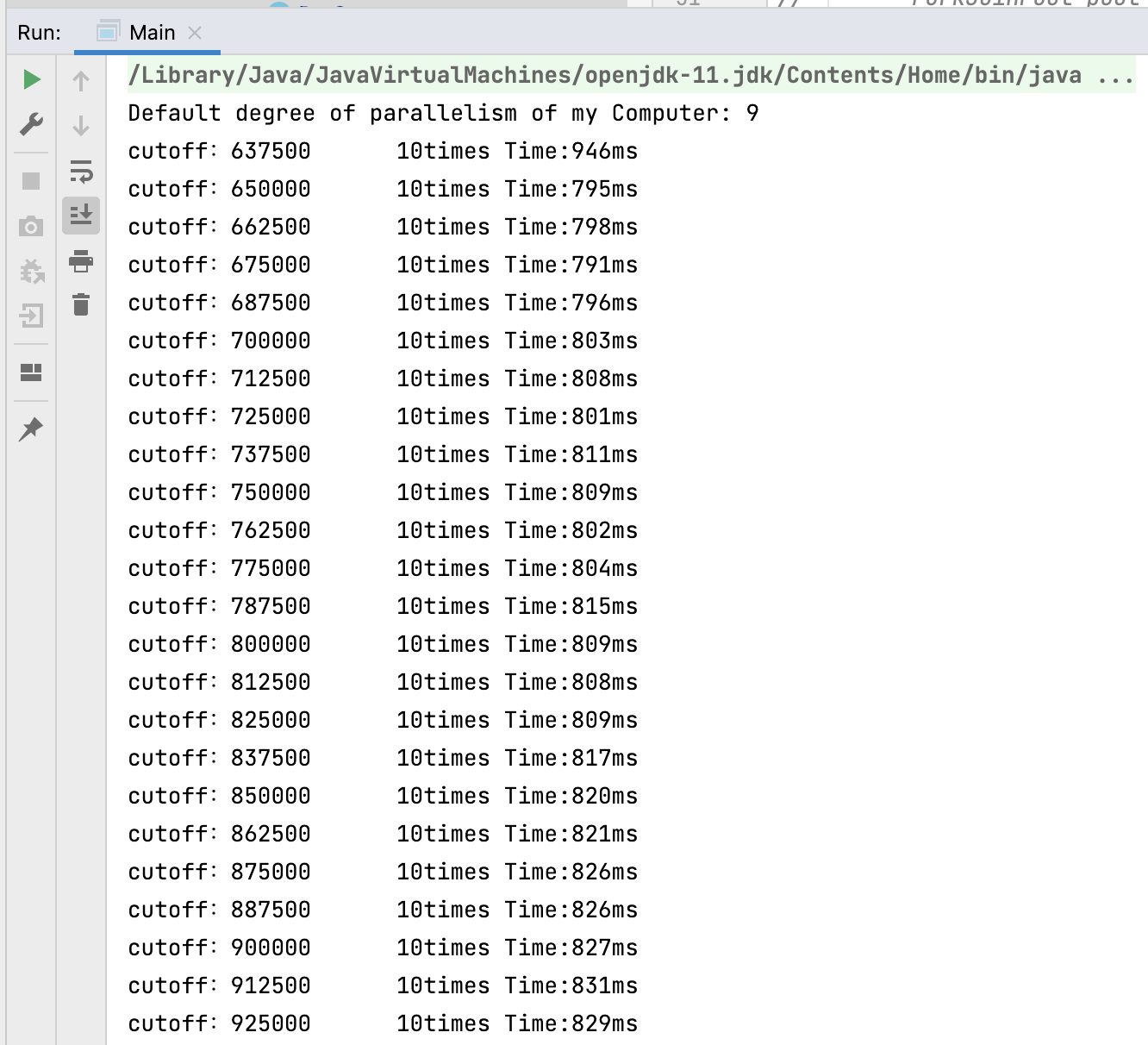
For varying the number of threads available, you might want to consult the following resources:

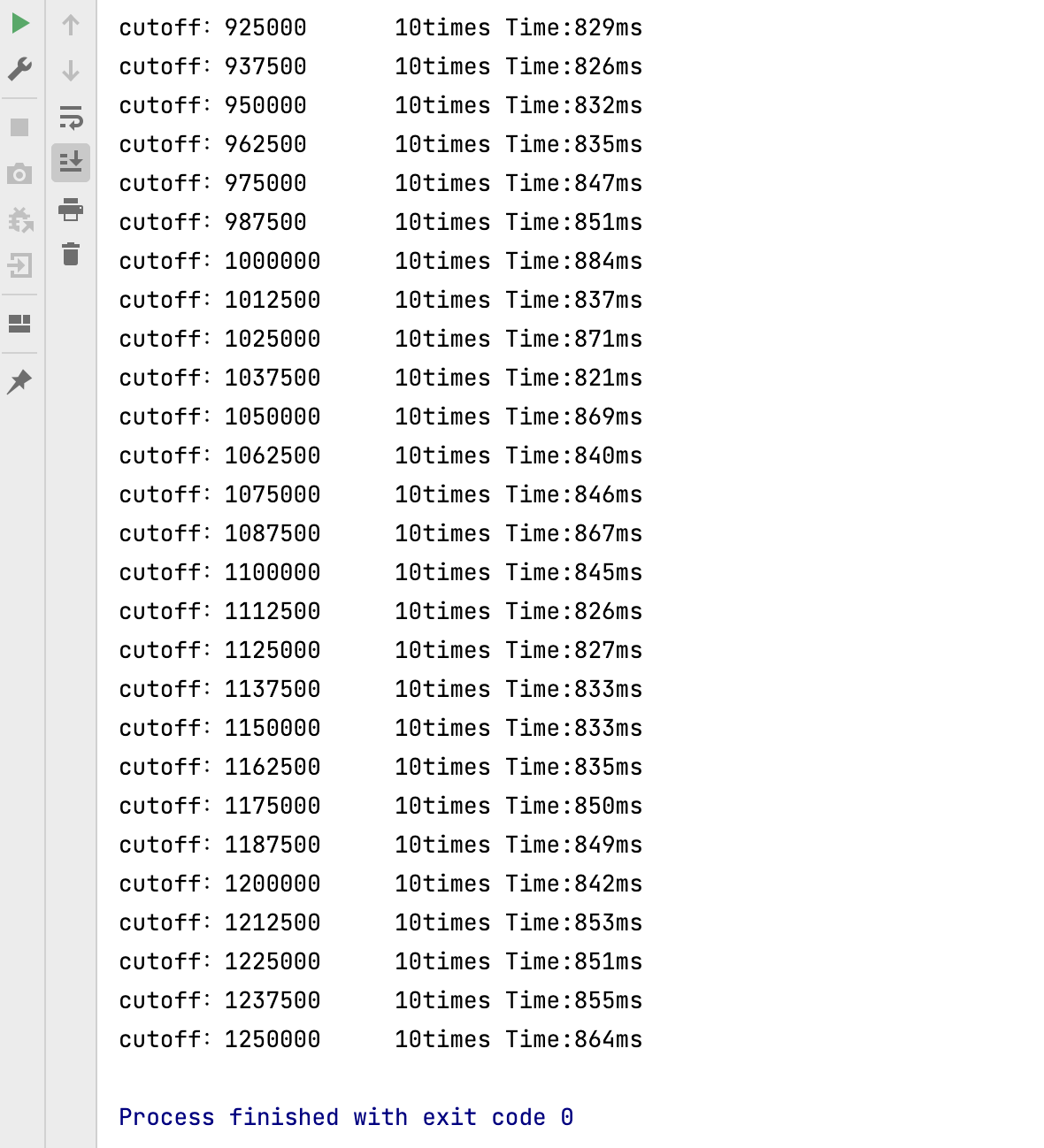
[https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-pool](https://www.callicoder.com/java-8-completablefuture-tutorial/" \l "a-note-about-executor-and-thread-pool) <https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable>

* **Output screenshot**

The test array here has a size of 2,500,000, which is large enough to show the benefits of parallel sorting.

In the screenshot, I set the thread number to 2.





We can see from the screenshot that: the cutoff is ranging from 637500 to 1250000, which I think has a relationship with the size of the input array. So in the result form, I will use the ratio of cutoff and array size as the criteria(cutoff / array size). And the Time is the sum of 10 times of operations. In the result form I will use the average time.

* **Relationship Conclusion**

As for the cutoff, the best cutoff I find is relate to the input array size. When the input array size is large enough to show the benefits of parallel sort, the idea cutoff will be the Array Size \* 0.265.

**CutOff = Array Size \* 0.265**

As for the separate threads, the idea threads run on my computer(default degree of parallelism is 9, which means it has 10 CPU cores) is **16 threads**. It is related to the calculation complexity of the assignment and the core numbers of the machine.

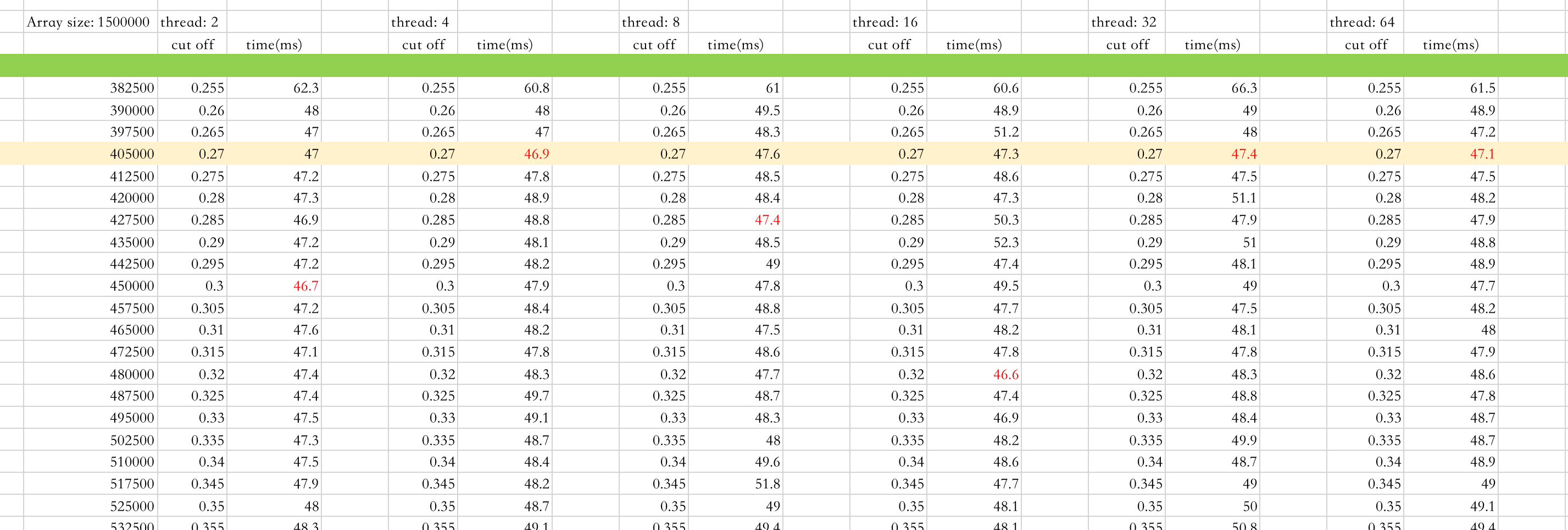
* **Evidence / Graph**

In order to get my conclusion, I tested array size with **1,500,000** and **2,000,000** and **2,500,000**. And thread number from **2** to **64**. Every output is the average of **10 times** operations.

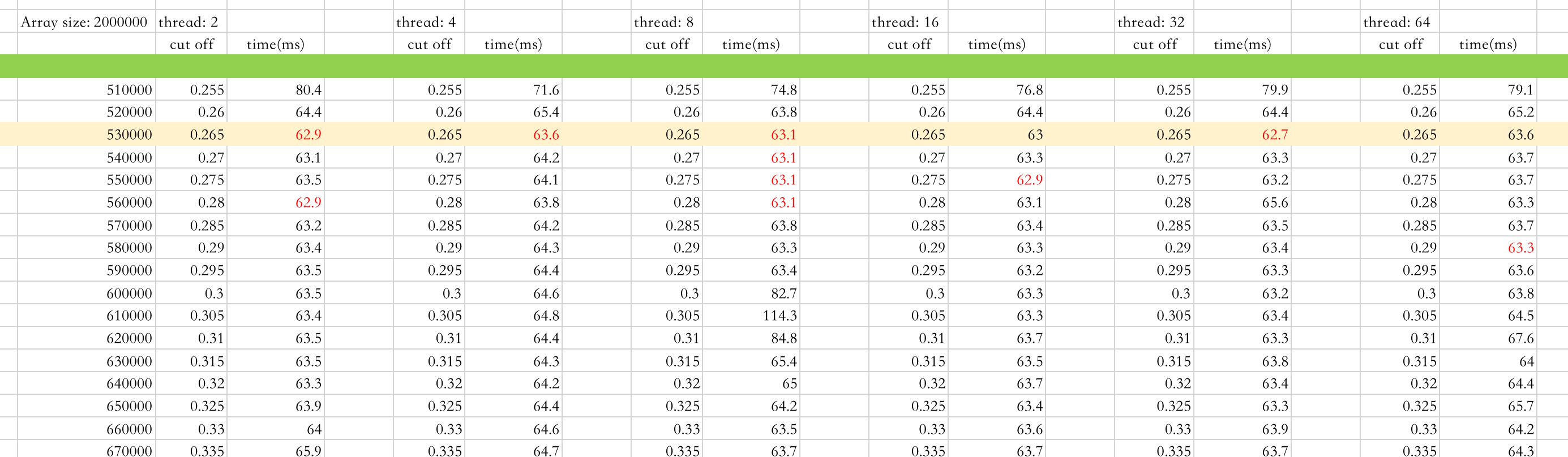
For the cutoff:

The cutoff number, or the cutoff ratio is ranging from 25.5% to 50%.

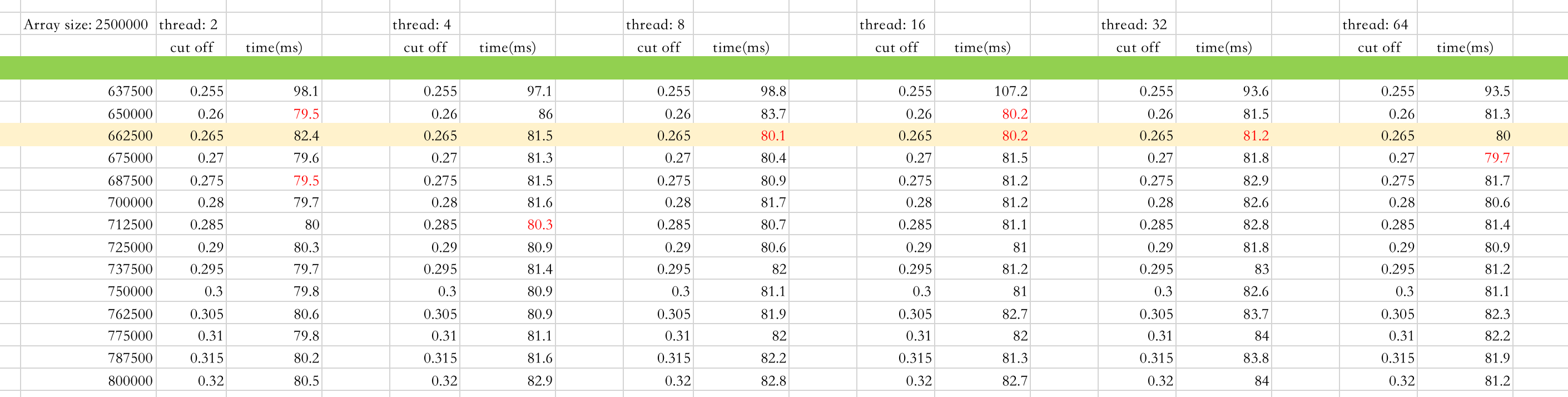
When the array size is 1,500,000:



When the array size is 2,000,000:

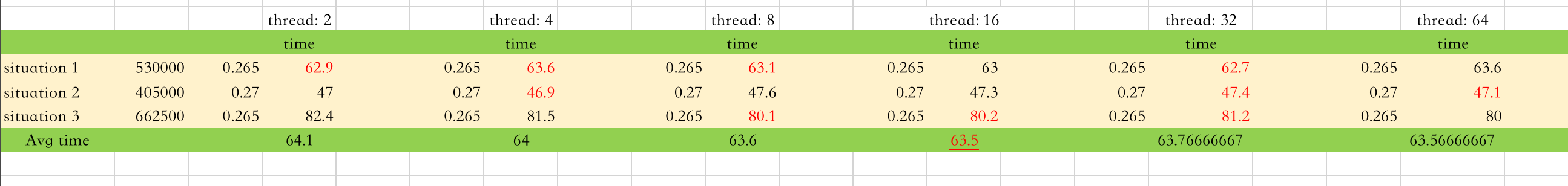


When the array size is 2,500,000:



We can tell from the graphs above that: when the ratio (curoff / size) is close to 0.265, on average the time consuming is the best.

For the threads:



I pick those best cases in different arrays sizes and calculate the average time consuming of them in different condition of threads. It turned out that in my circumstance, the best separate threads number is **16**, which has the lowest time consuming.