Zhijie Li (001563872)

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 5**

* **Task (List down the tasks performed in the Assignment)**

**•** Implement a parallel sorting algorithm such that each partition of the array is sorted in parallel

**•** Find relation between the efficacy of this method of parallelizing sort and number of threads or cutoff

* **Relationship Conclusion:**

• When the ratio of cutoff/Elements is around 0.36, the efficacy comes at best

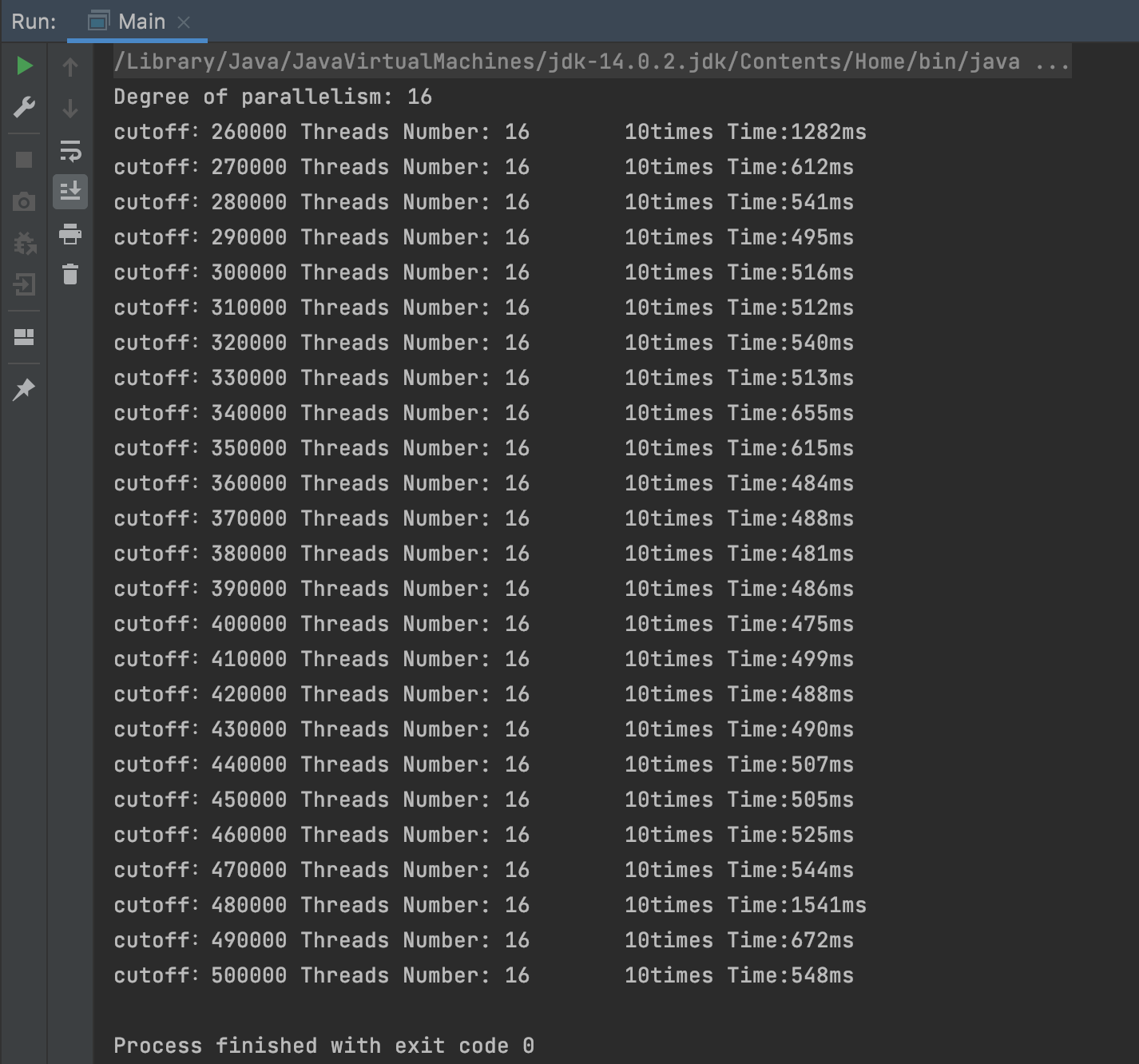
• When the recursion depth is around 4 (which is the concurrency threads of my cpu), the time consumed is shortest.

* **Evidence to support the conclusion:**

As I adjust array sizes and cutoff, and define array sizes respectively as 500,000, 1,000,000, 2,000,000. However, as the memory limit of my macbook, I set the element number as 1,000,000 (which I think is big enough) and to distinguish cutoffs and recursion depth.

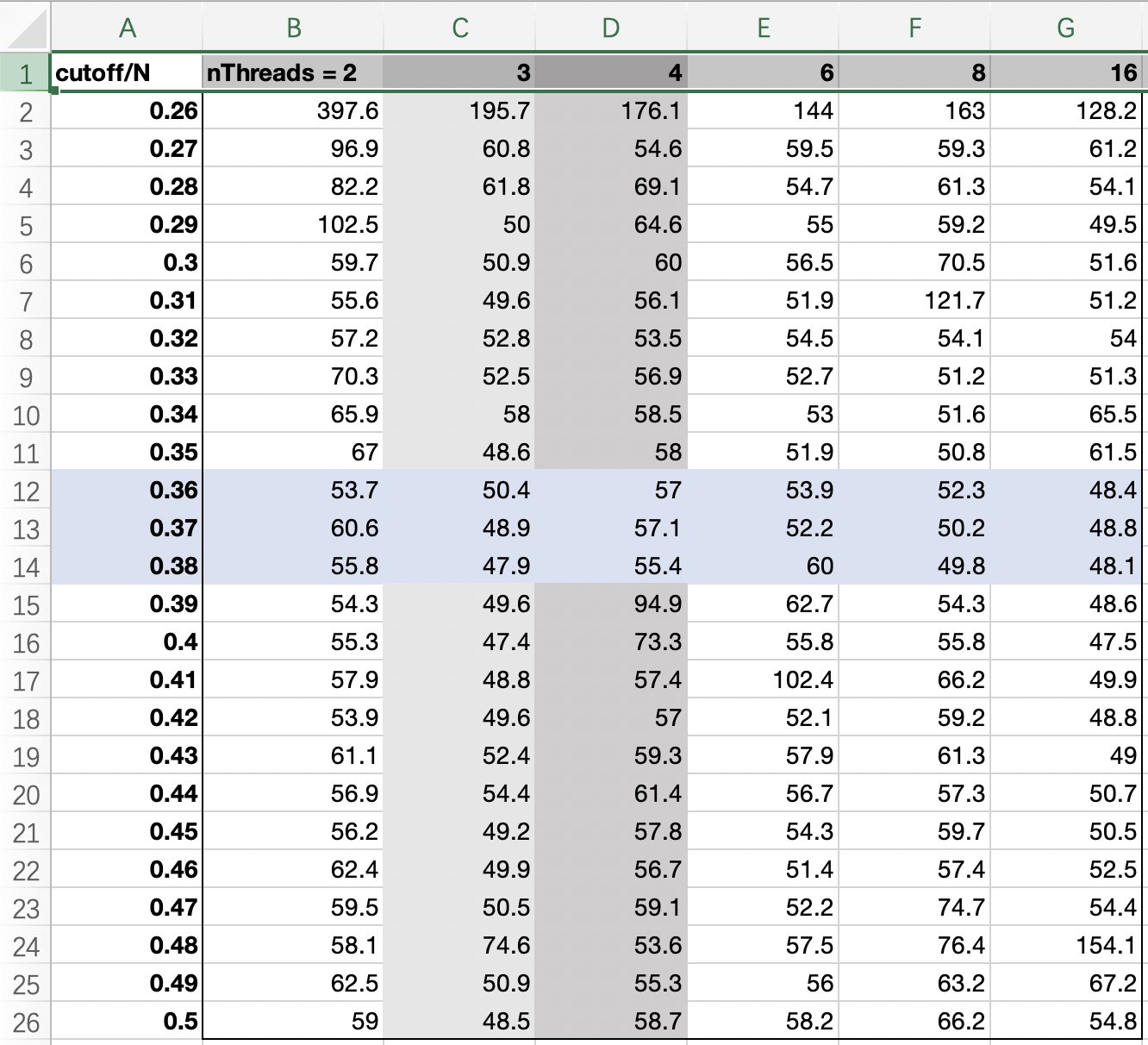
As I set cutoff between 25000 and 50000, the time-list lines go down and as cutoff/N ratio is around 0.36 (below in blue row), the time-to-cutoff generally has its first flat. Generally my macbook takes 3 threads at best, and with nearly 1/3 as cutoff by elements. It means it concurrency deals with 3 partitions and each sort.

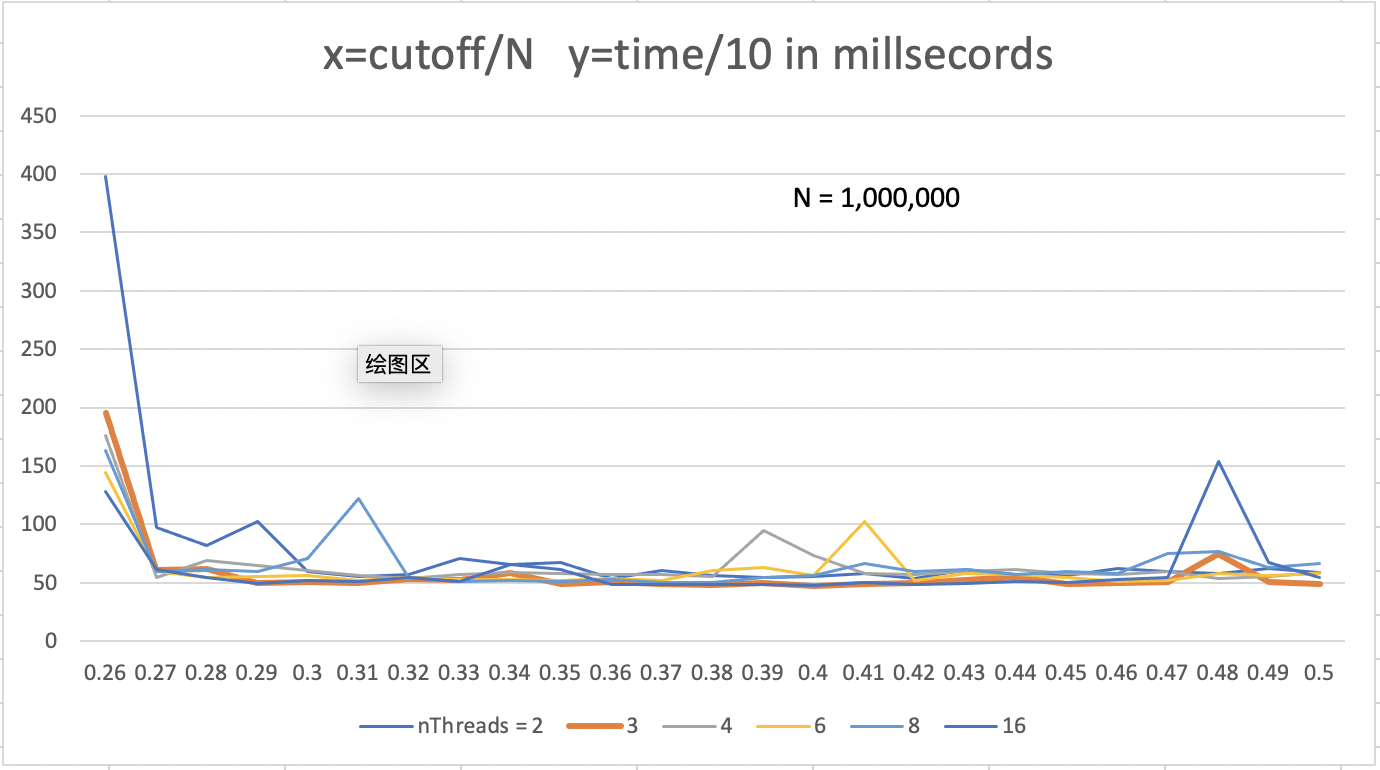
On the other hand, number of threads is set as 2, 4, 8, 16. And it grows most flat when threads number is around 4 (below the grey line, and actually I got my best efficacy with 3 threads), and my macbook is 4 concurrency threads. Because merge sort takes extra memory, when the subarray below cutoff is sorted, to merge them need to transfer between core, or thread. So too many threads may however, not benefit the efficacy.

1. **Output (Snapshot of Code output in the terminal)**

*The case of 16 threads and size as 1,000,000*

1. **Graphical Representation(Observations from experiments should be tabulated and analyzed by plotting graphs(usually in excel) to arrive on the relationship conclusion)**

****



*The grey line (4 threads) and red line (3 threads) grow most flat*