**Shuo Wang (001533020)**

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 5**

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

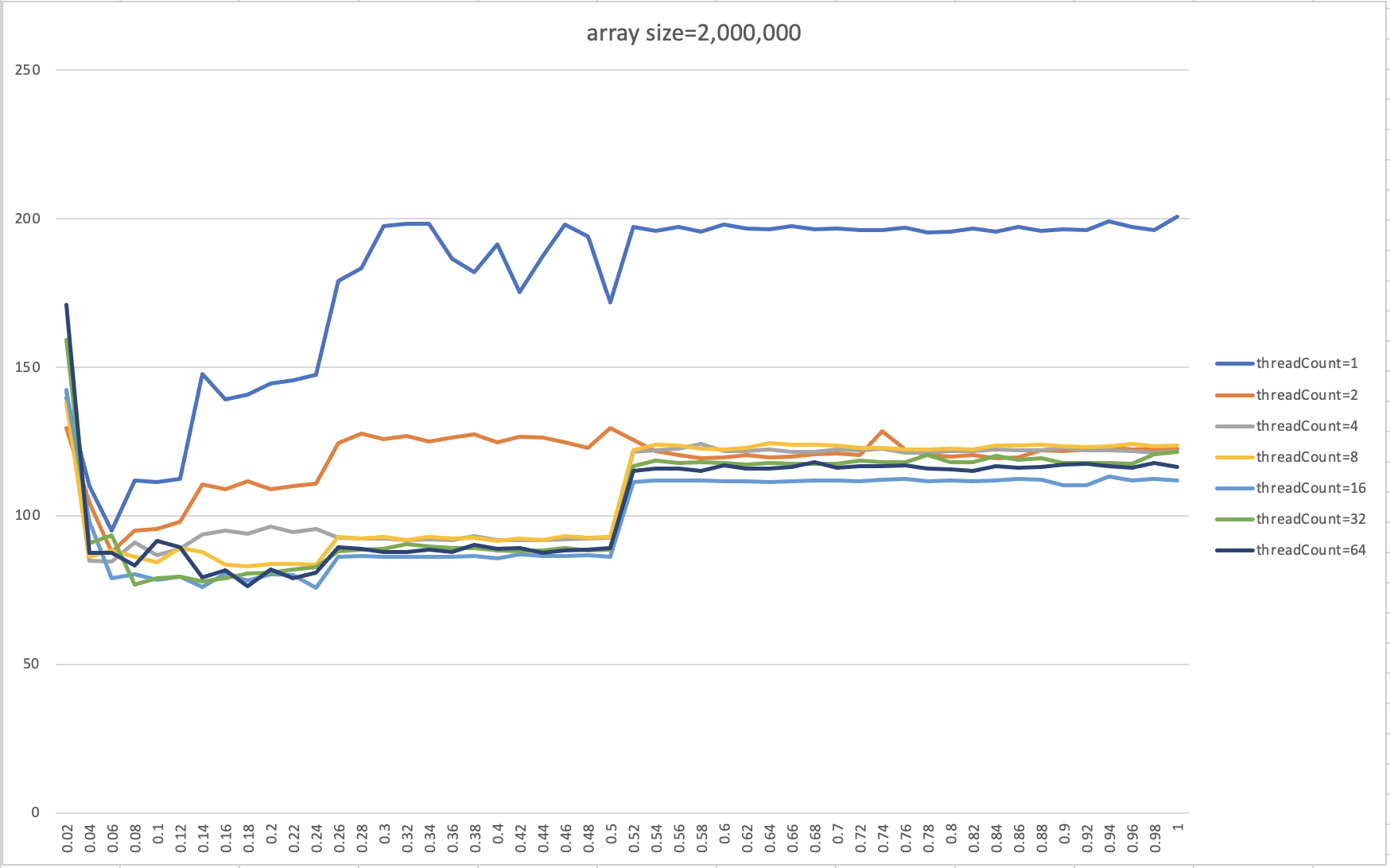
1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. Recursion depth or 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).
3. An appropriate combination of these.

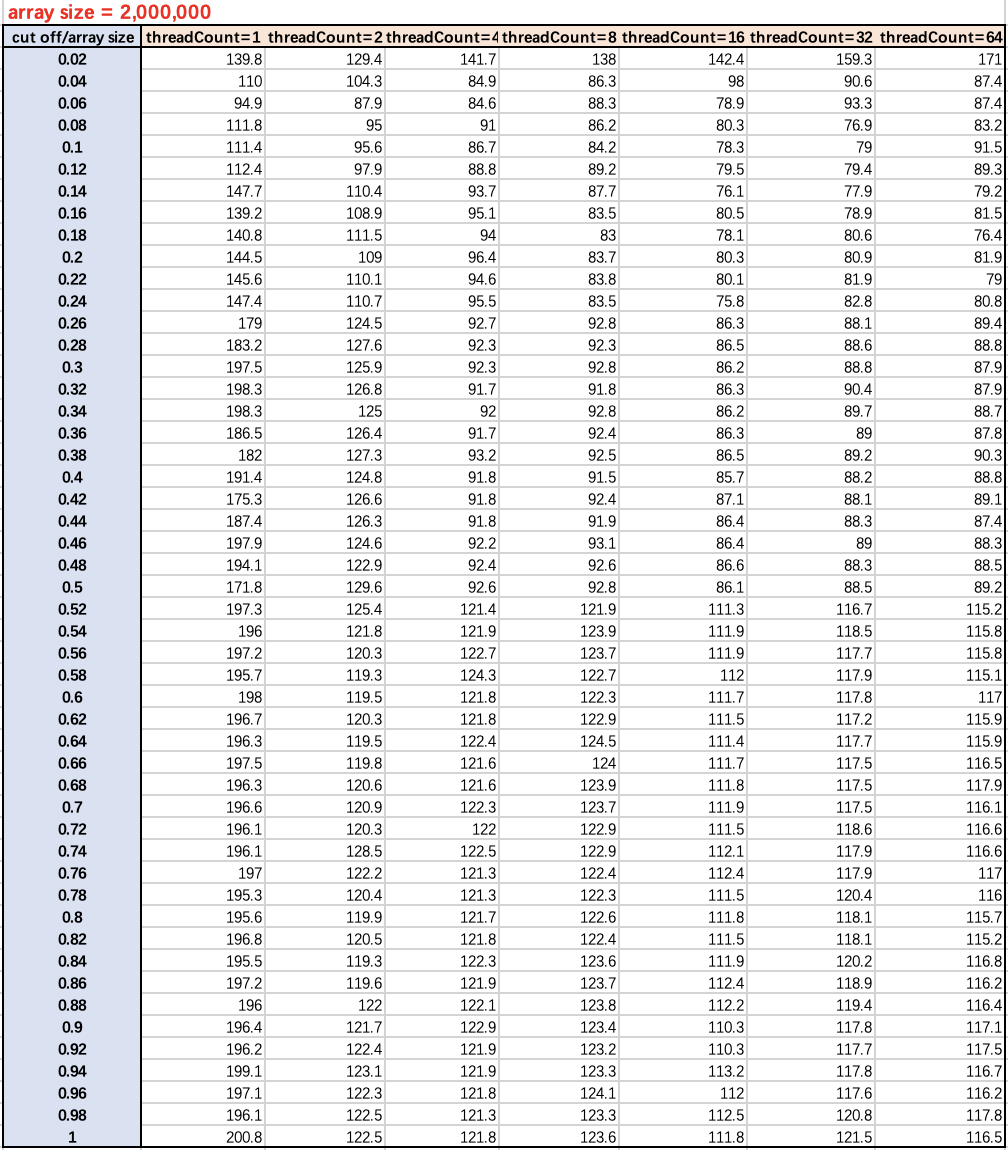
* **Relationship Conclusion:**

1. ThreadCount = 16 is a good choice in my computer for Parallel Sorting.
2. Cut off percent between 0.08~0.22 is a suitable range for Parallel Sorting when we choose a reasonable treadCount number (such as 8, 16 or 32).
3. Sorting time is proportional to the array size.

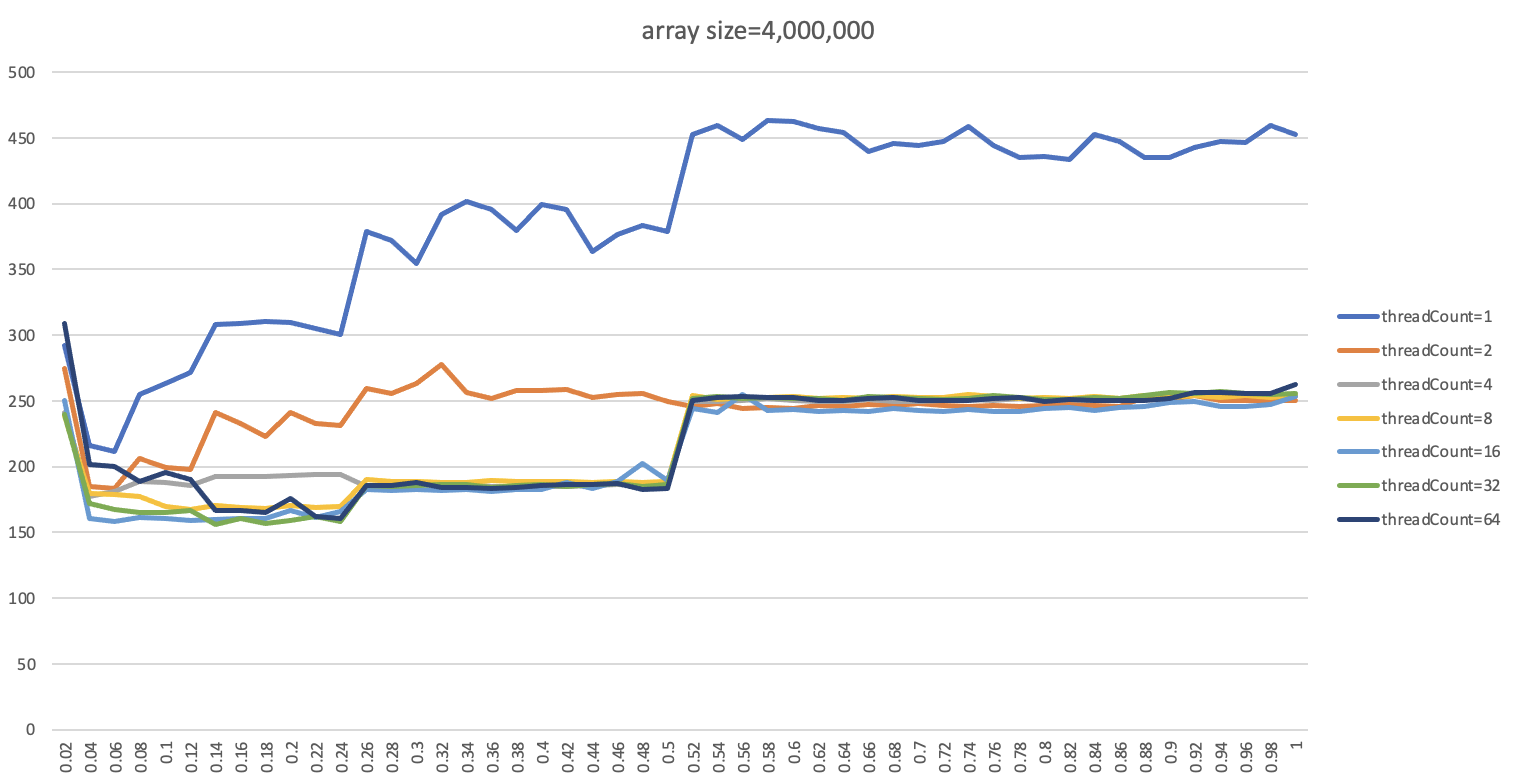
* **Evidence to support the conclusion:**

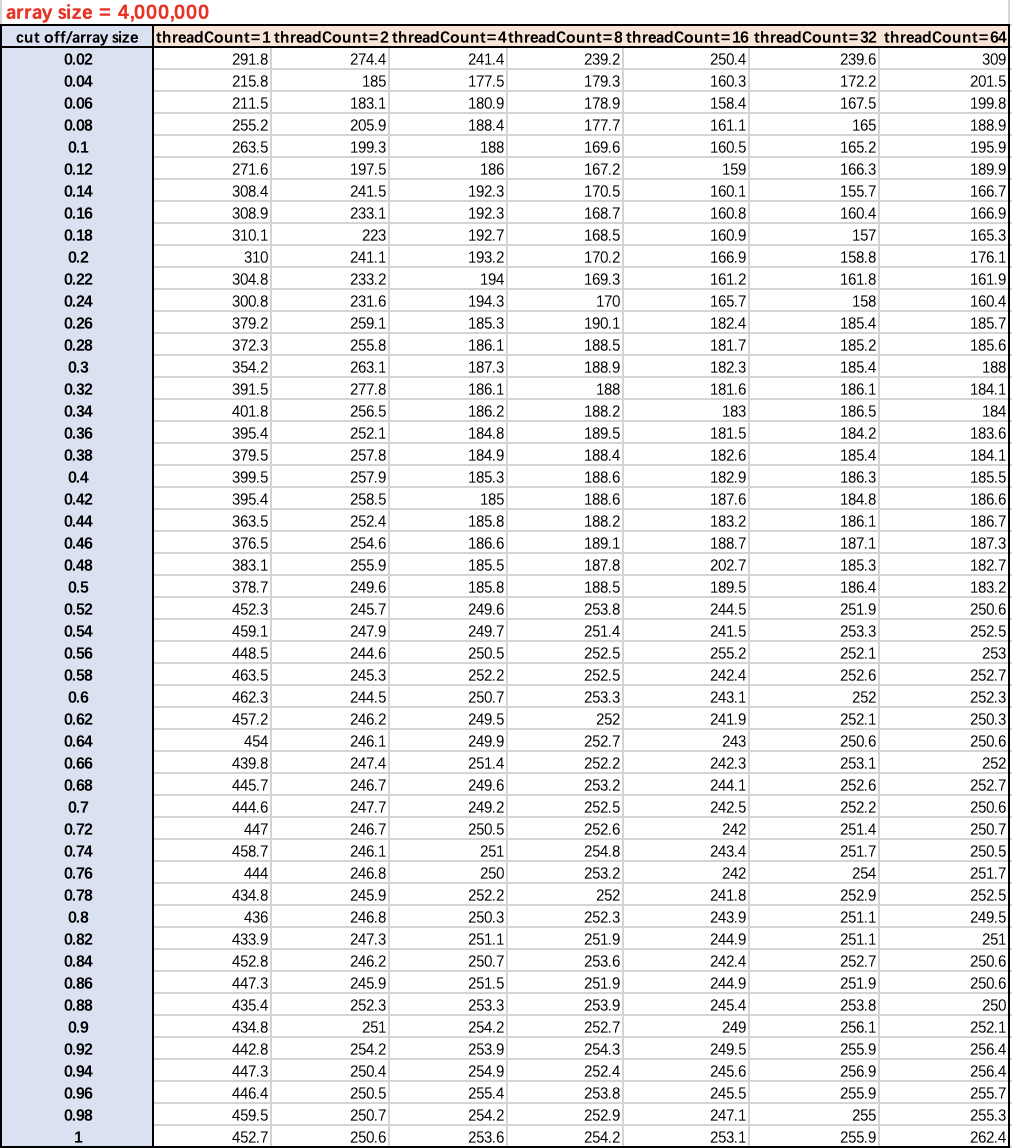
1. First, I chose a **fixed array size = 2,000,000**. And I tried to use **threadCount = 1,2,4,8,16,32,64** and **different cut off** in my experiment. I found that when threadCount = 1, the sorting is the slowest one, and when **threadCount = 16**, the performance is the best. When threadCount is small, the better cut off is about 0.06 percent of array size. And when threadCount is in a suitable range such as 4~64, the best cut off is between **0.08~0.22** percent of array size. And when the cut off percent larger than 0.5, the performance has a huge increase, and stable between 0.5~1.

****

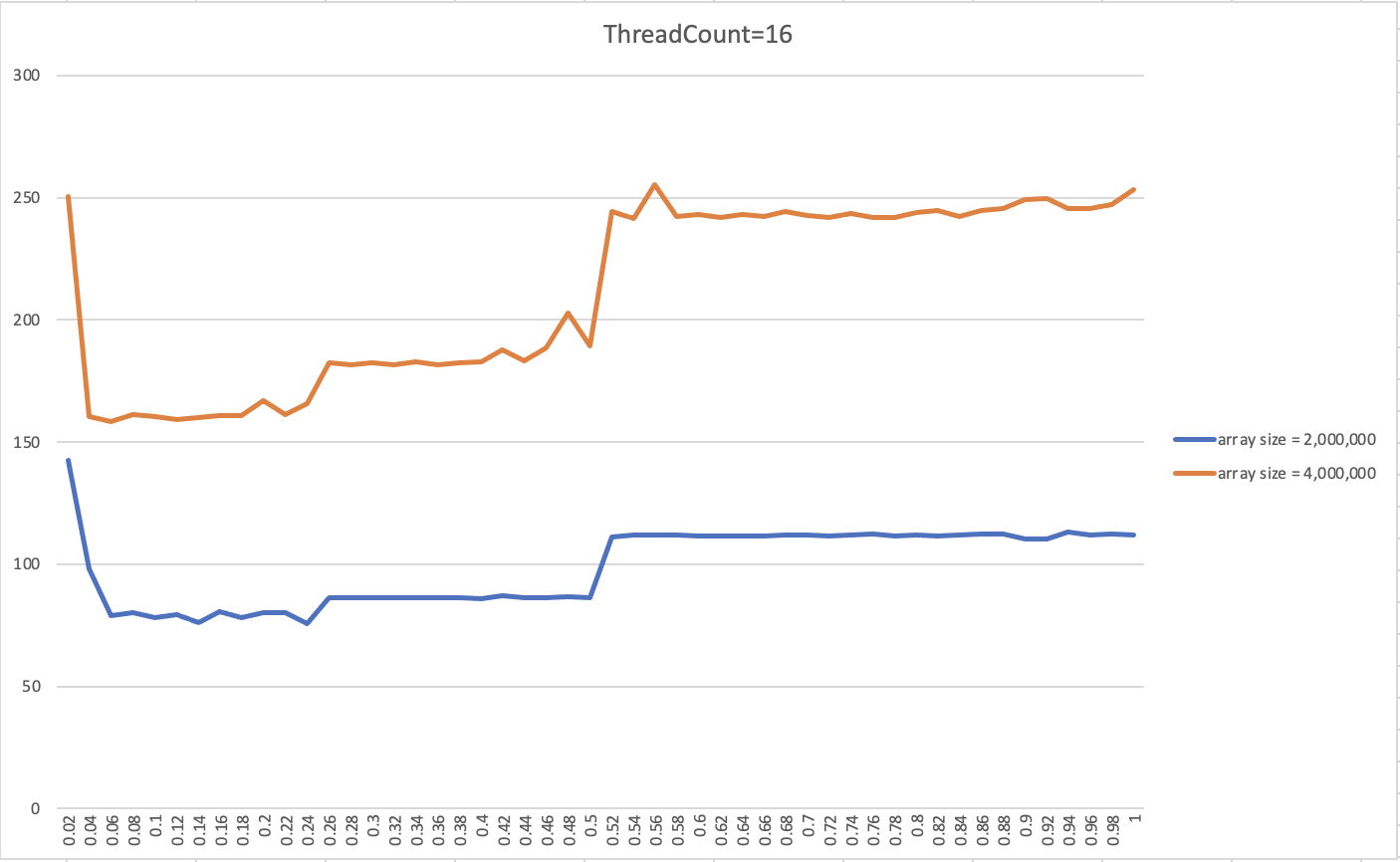


1. Second, I chose a **fixed array size = 4,000,000**. And I tried to use the same set of data: **threadCount = 1,2,4,8,16,32,64** and **different cut off** in my experiment. I found that when threadCount = 1, the sorting is the slowest one, and when **threadCount = 16/32**, the performance are the best two (they are very close). When threadCount is small, the better cut off is about 0.06 percent of array size. And when threadCount is in a suitable range such as 4~64, the best cut of is between **0.08~0.22** percent of array size. And when the cut off percent larger than 0.5, the performance has a huge increase, and stable between 0.5~1.

****

****

1. Third, I chose a **fixed threadCount = 16**, because when threadCount = 16 the performance is good whatever I chose array size = 2,000,000 or 4,000,000. I found sorting time of array size = 2,000,000 and 4,000,000 changed along with percent of array size are synchronize, both of them are short when cut off percent between 0.06~0.24, and have a huge increase when cut off percent = 0.5, and stable when cut off percent > 0.5. The sorting time of array size = 4,000,000 is about 2 times bigger than array size = 2,000,000. So I guess that the sorting time is proportional to the array size.

****

* **One of Screenshots:**

