Program Structures and Algorithms
Assignment 5 (Parallel Sorting)
Spring 2024

NAME: Suchita Arvind Dabir
NUID: 002957879
GITHUB LINK: https://github.com/suchitadabir/INFO6205

Tasks:

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.

Solution:

In my analysis, I double the array size starting from 1M upto 5 runs.

Hence, **ArraySizes = [1M, 2M, 4M, 8M, 16M]**

For the thread count, I also double the thread count with a start value of 2 and upto one level above the max number of threads supported on my laptop:

```
sysctl hw.physicalcpu hw.logicalcpu
hw.physicalcpu: 8
hw.logicalcpu: 8
```

Hence, **NumThreads = [2,4,8,16]**

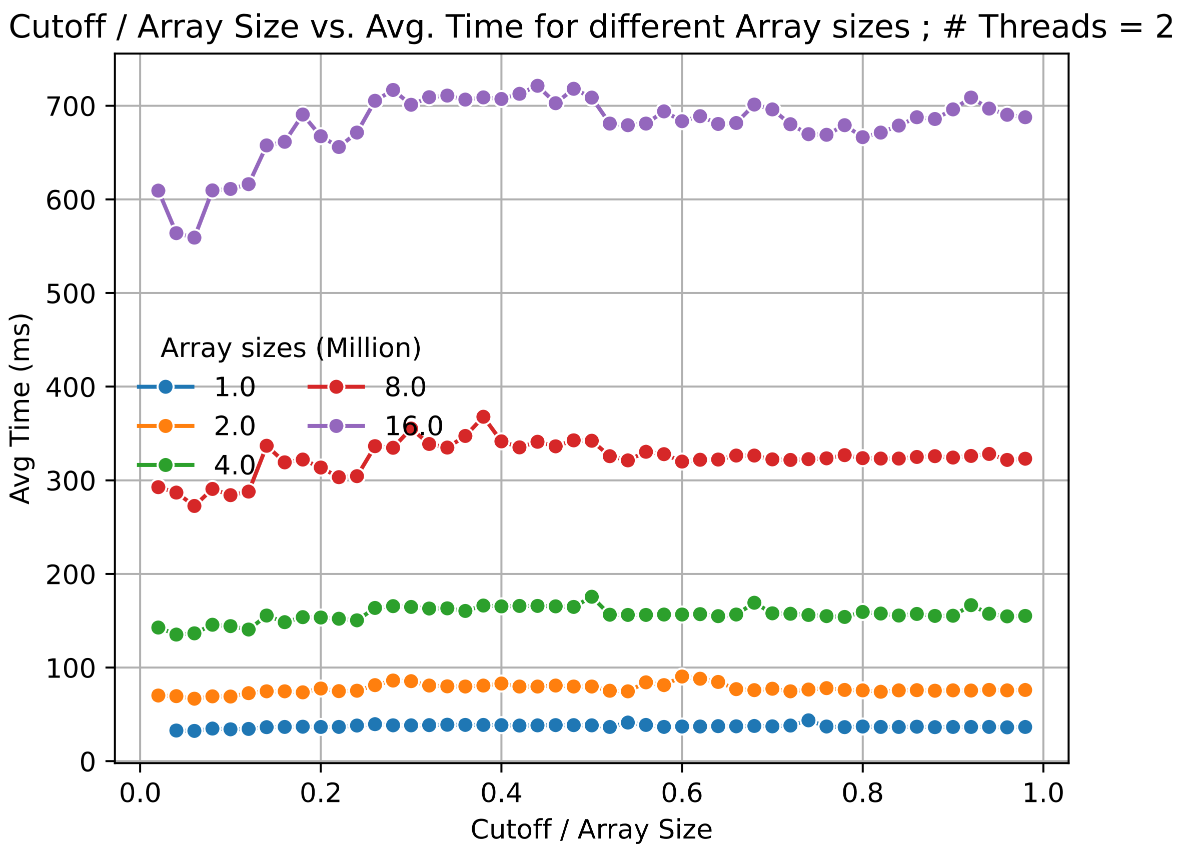
I analyze the effect of parallel sorting with 50 different cutoff values. These values are chosen based on the input array sizes such that the ratio of cutoff to array size varies from 0 to 1 in 0.02 increments (j).

Cutoff = ArraySize X (j)

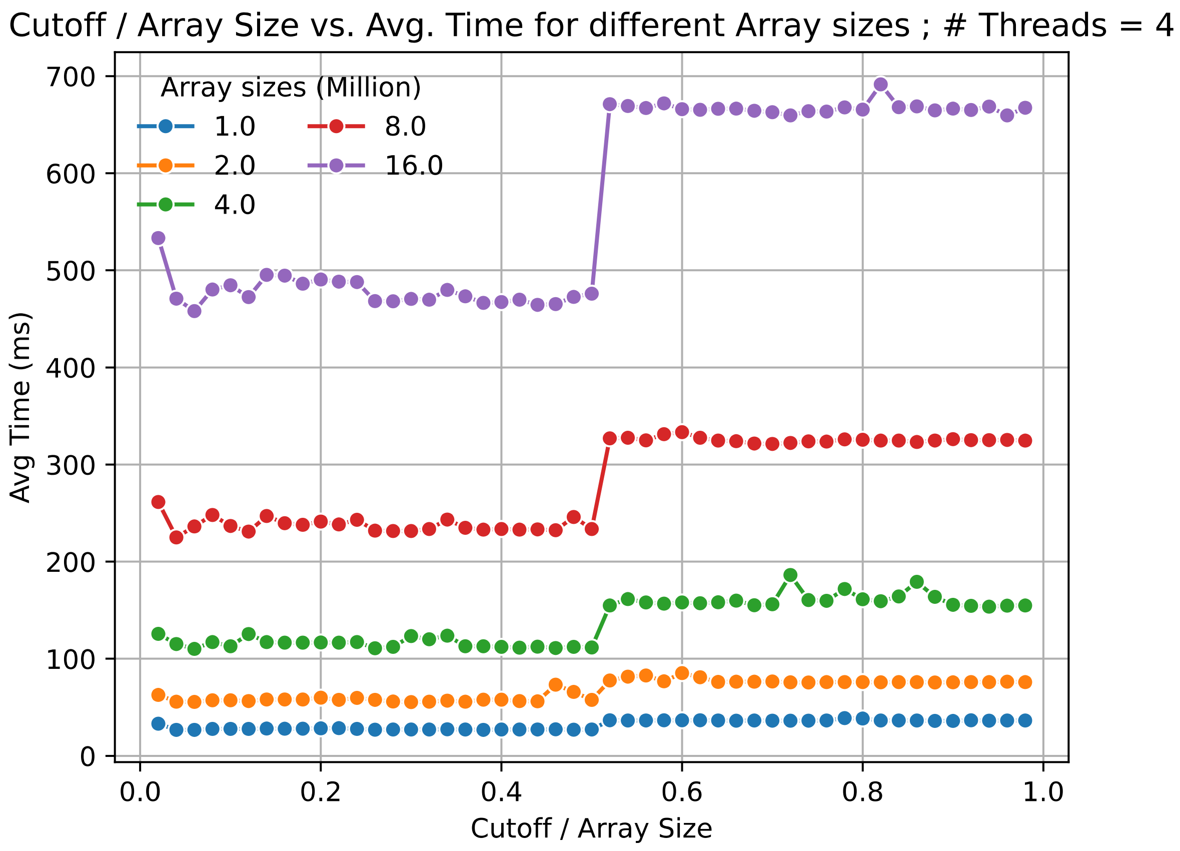
I plot the (Cutoff / ArraySize) value X-axis and avg Time (ms) on Y-axis.

First, I keep the parallelization level constant to one single value and plot the time taken by parallel sort for different array sizes as below:

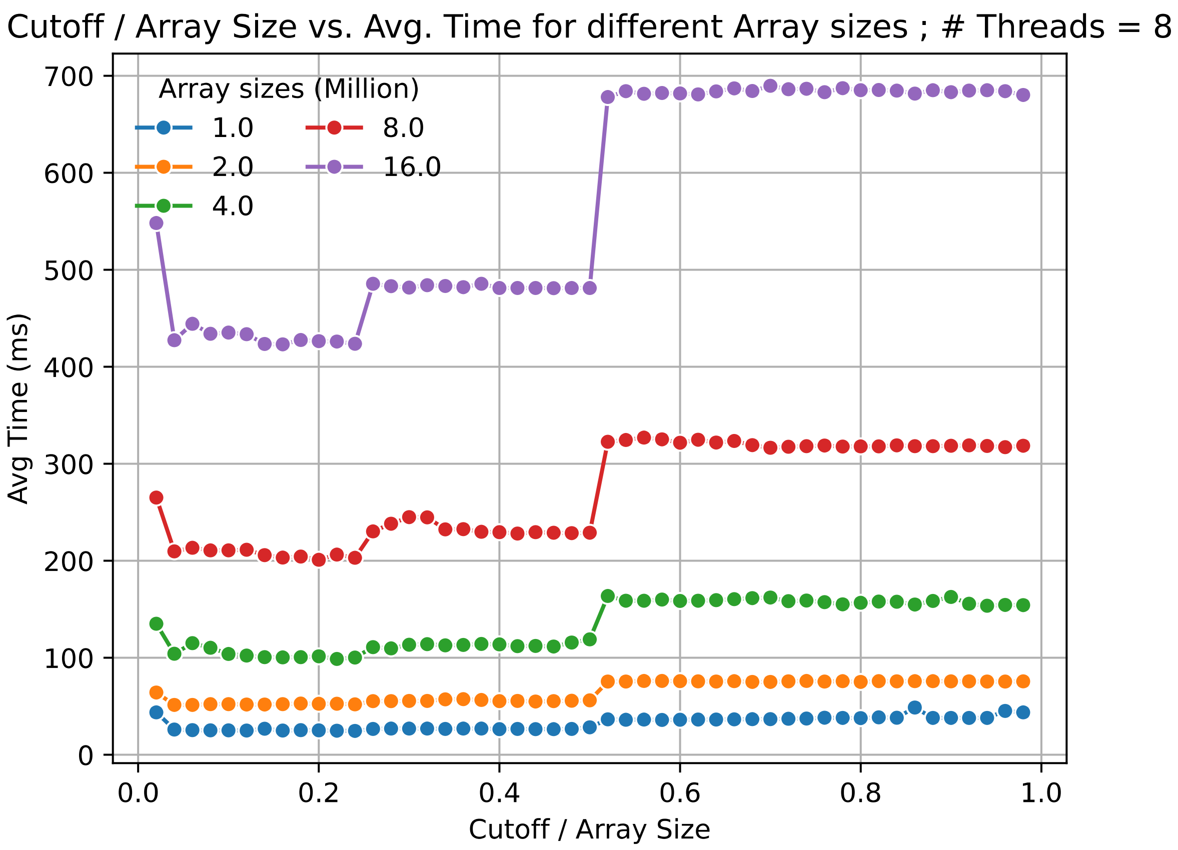
Parallelization Level = 2:



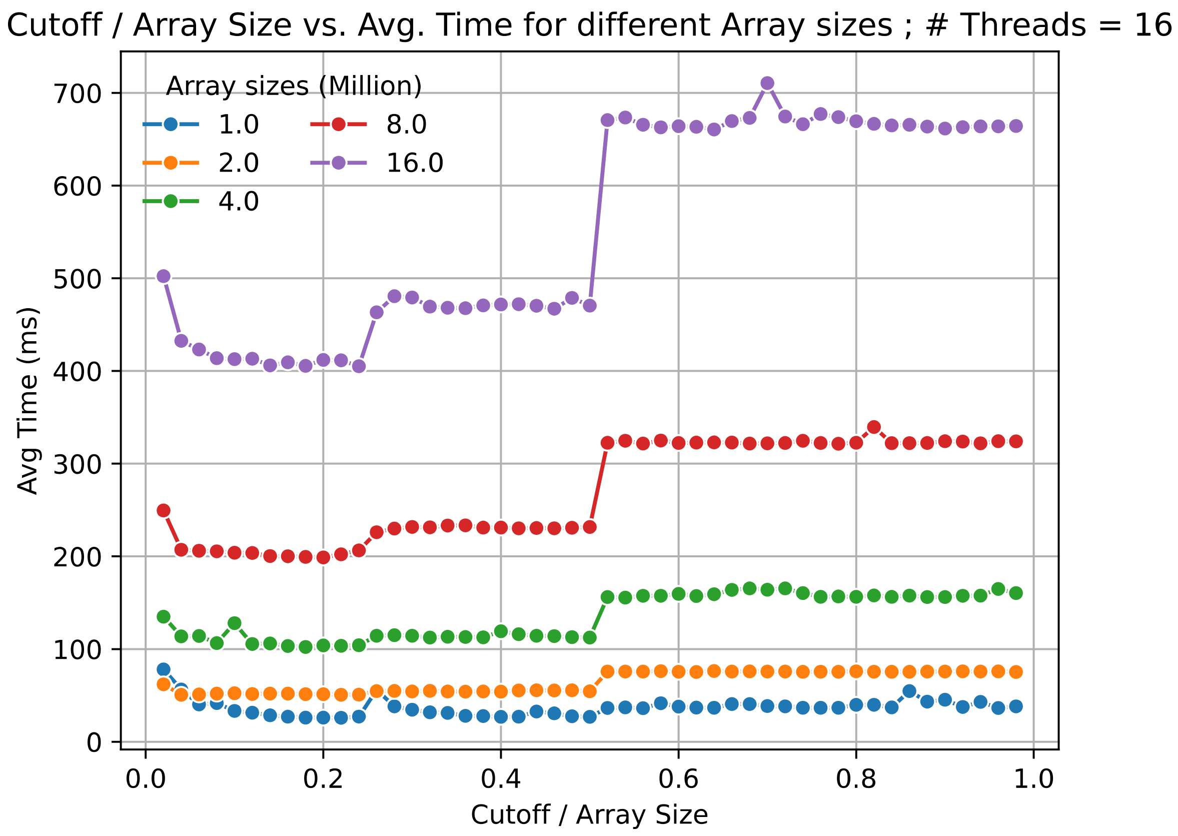
Parallelization Level = 4:



Parallelization Level = 8:



Parallelization Level = 16:

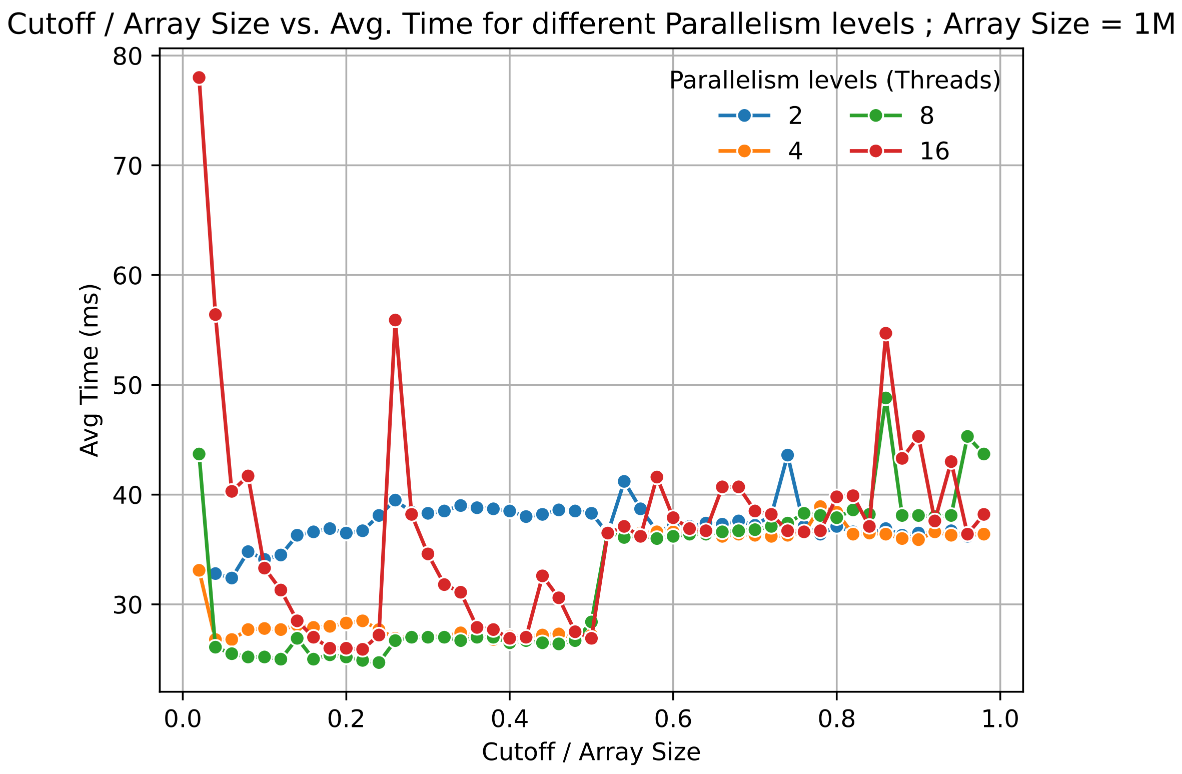


Above four plots show that as the size of input array increases, the time taken to sort the array also increases. Moreover, for X-axis values (Cutoff / Array size) > 0.5, the time taken increases sharply for all array input sizes. This increase in time after X value > 0.5, is observed prominently for arrays with considerably higher input sizes. That's why, we see negligible increase in sort time for 1M array size after X > 0.5 as compared to 16M array size.

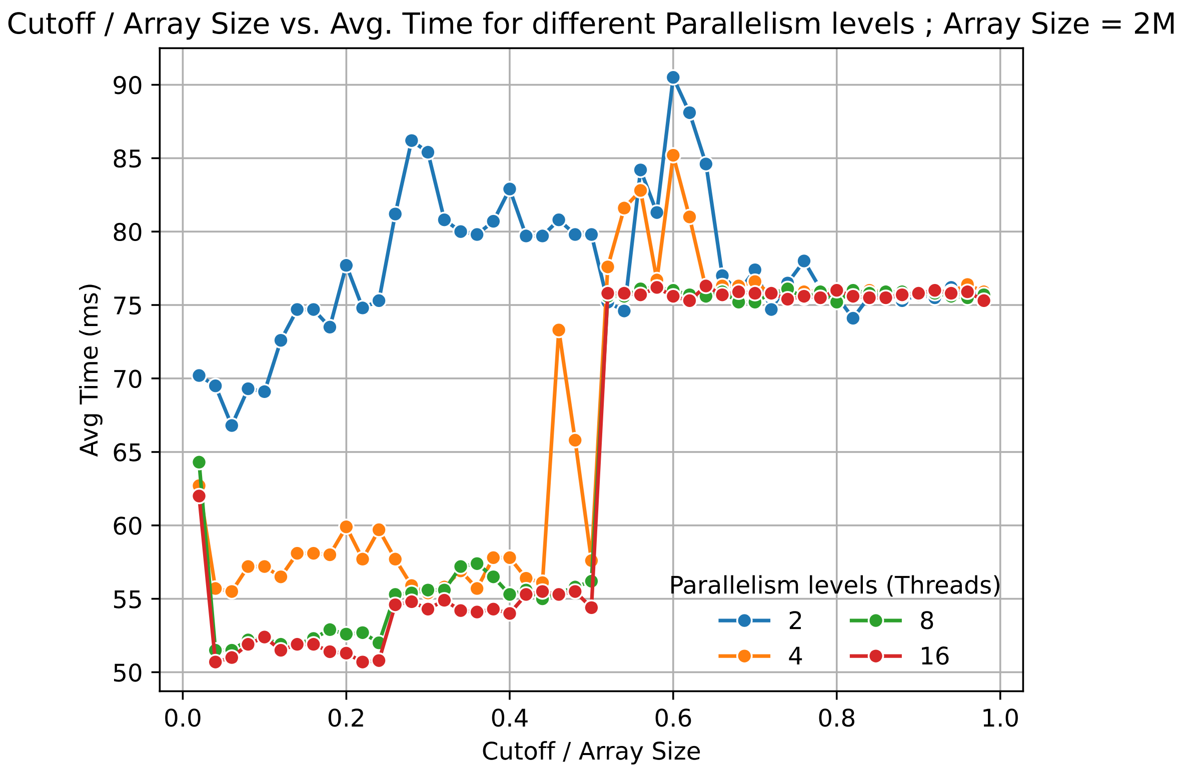
This behavior is consistent across all number of threads except when number of threads are equal to 2. When # threads = 2, the sorting time remains on average constant across all values of (Cutoff / Array size). Therefore, we can conclude that the ideal value for this (Cutoff / Array size) is 0.5 or less.

Next, I keep the array size constant to one single value and plot the time taken by parallel sort across parallelization levels as below:

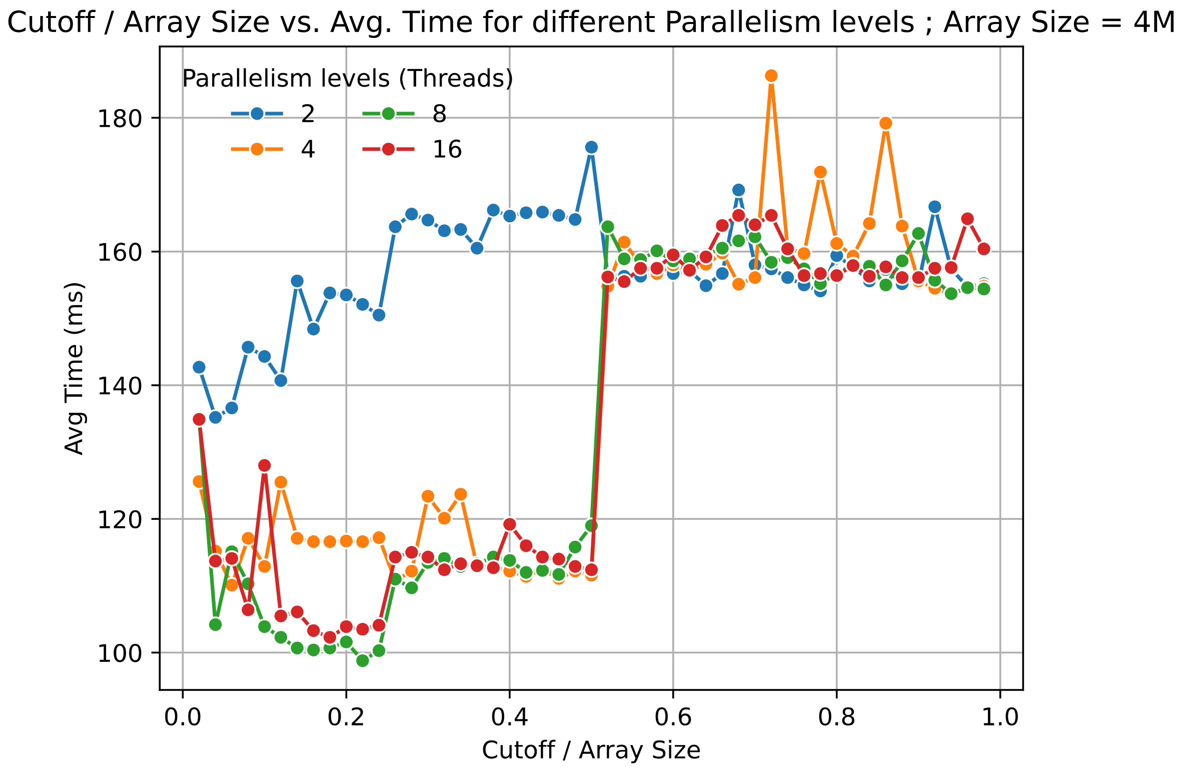
Array Size = 1M:



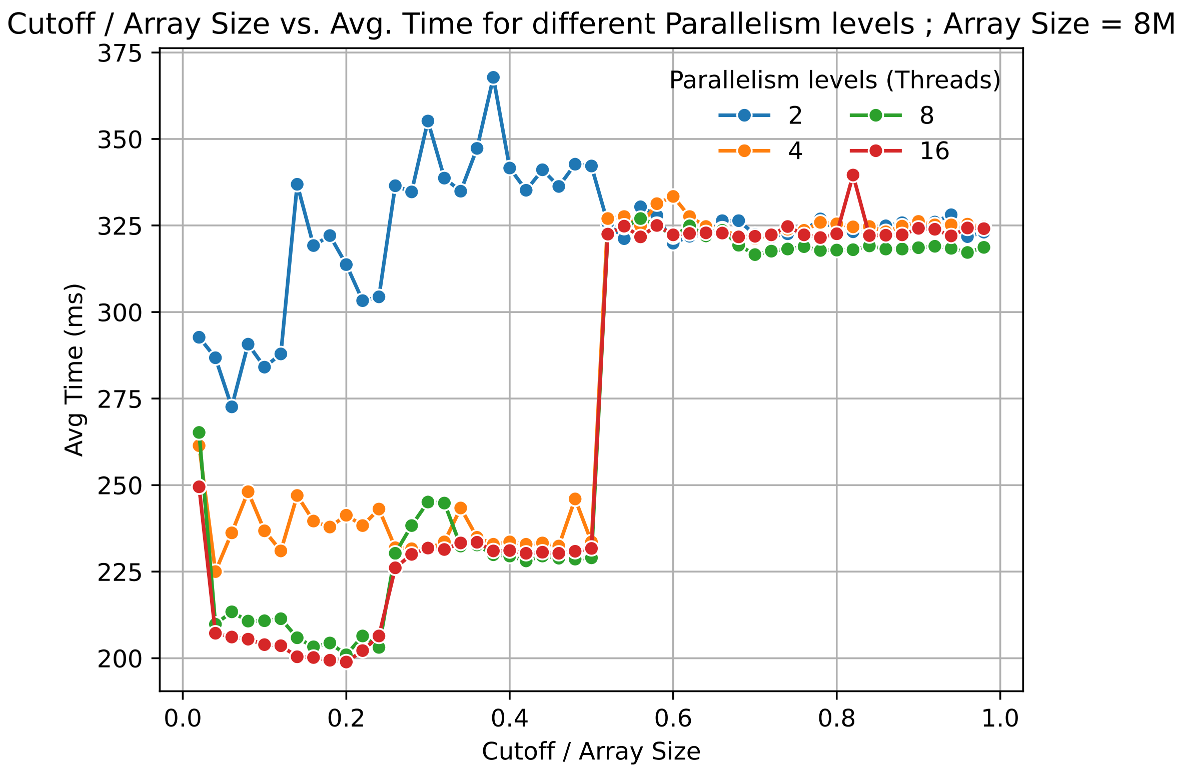
Array Size = 2M:



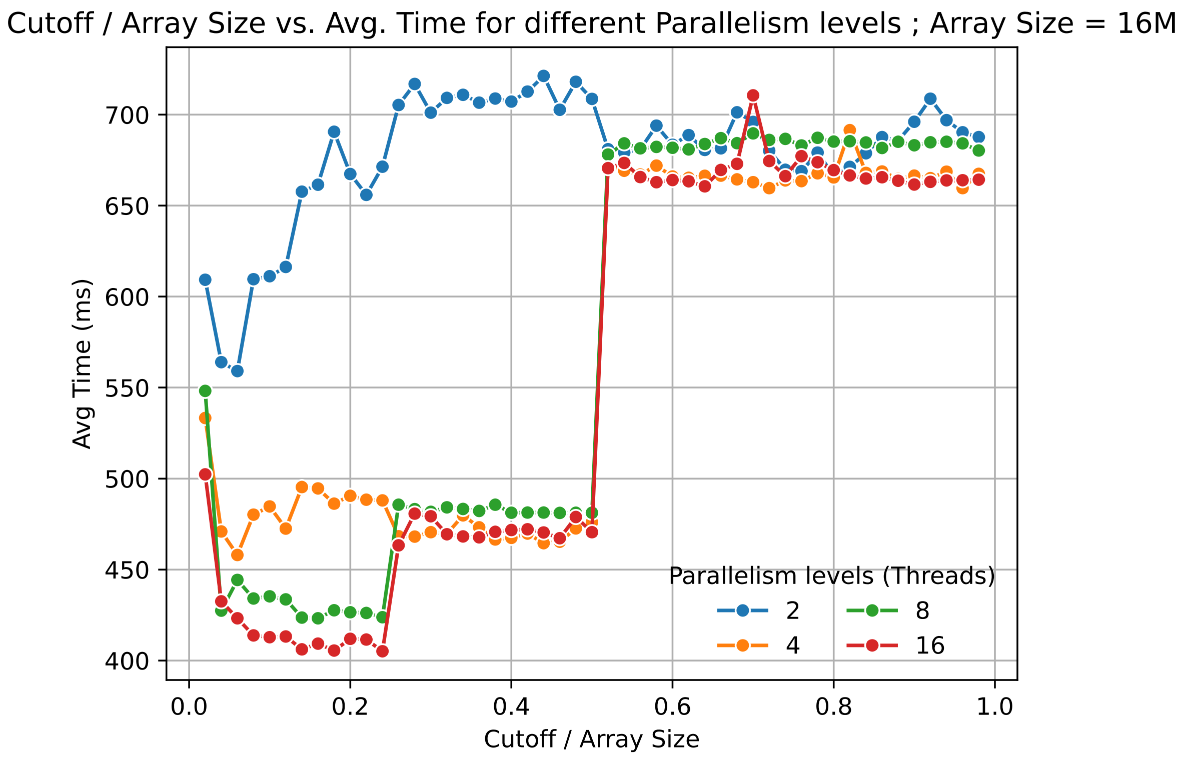
Array Size = 4M:



Array Size = 8M:



Array Size = 16M:



From above four plots, we can conclude that as the level of parallelization increases, the time taken to sort the array decreases. Since, my system has 8 logical processors, increasing parallelization beyond this value does not yield any improvement.

After crossing the (Cutoff / Array Size) value of 0.5, all the sorting time remains constant irrespective of the parallelization levels being undertaken. This again proves that ideal value for (Cutoff / Array Size) is 0.5 or less.

Note: Since, the CSV data generated after running these experiments is huge, I have not added it into this document. Instead, it has been checked in to the GitHub repository and plots are added based on the same data..