Efficiency

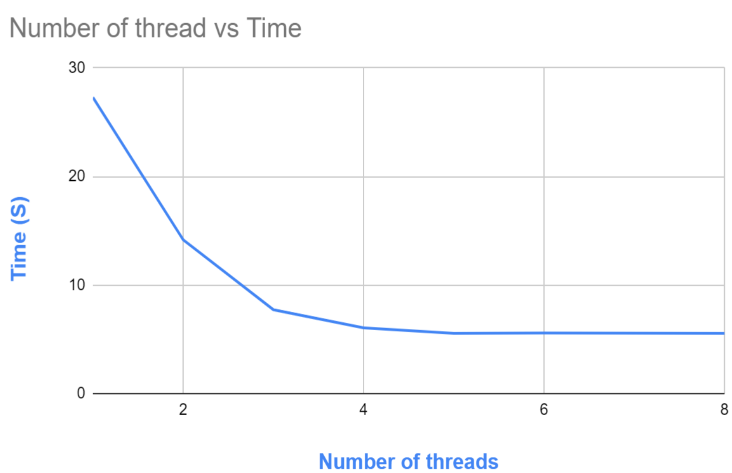
I executed 100 million elements for various threads(1-8) in onyx.boisestate.edu to remain consistent with the system and testing purpose.

The no of cores and system information of the onyx server is as follows:

Model name: Intel(R) Core(TM) i5-8500T CPU @ 2.10GHz
CPU family: 6

The analysis of the report is done in the **spreadsheet file** attached to the repository.

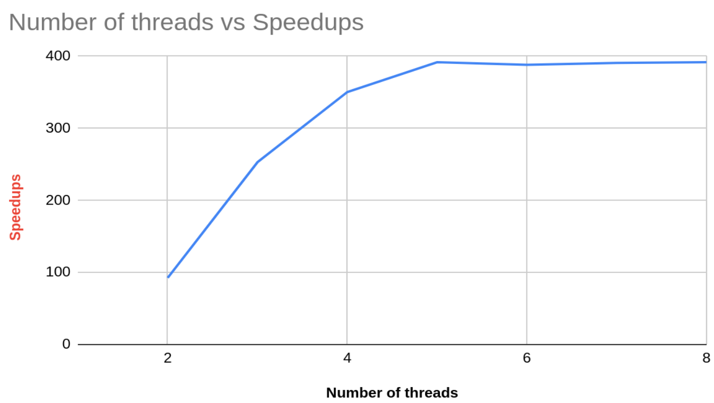
I have plotted the no of threads vs the time taken to execute 100 million elements as shown in figure 1.



Similarly, I have tried to understand the speed-ups while increasing the number of threads in comparison to single-threaded.

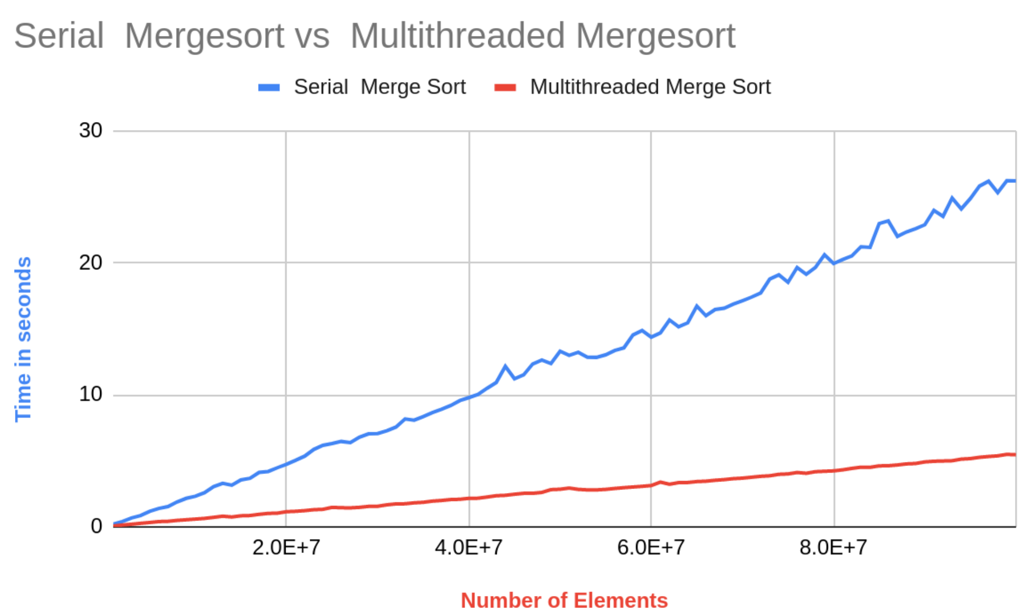
Speed up improvement compared to single thread is given by the formula,

Speed up(n) improvement $= \dfrac{T(single\ thread) - T(n - thread)}{T(n - thread)} X\ 100$



We can see that when I increase the number of threads to 2, it takes less time to execute 100 million elements by around 92.5% speedups improvement. Similarly, when I increase the number of threads to 3, it takes less time to execute by around 253 % speedup improvement in comparison to a single thread. When I increase the thread by 4, ultimately it has a 350% speedup improvement. However, it looks like the speedup improvement remains almost the same from 5 to 8 threads.

Effectiveness



It looks like the multithreaded version of mergesort is always better than serial mergesort. It takes less time in every case to sort the same element in multithreaded in comparison to serial mergesort. Hence it is very effective to use a multithreaded version of merge sort.