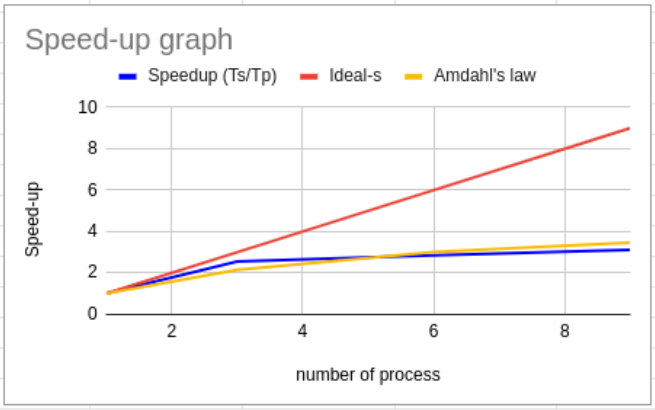
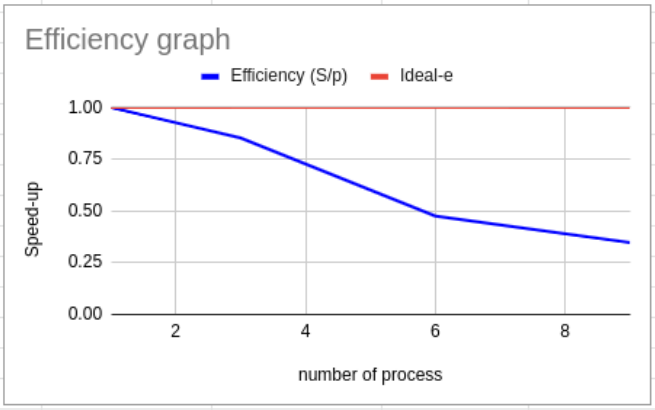
Quick-Sort					sequencial / all			
n process	Ts (sec.)	Tp (sec.)	Speedup (Ts/Tp)	Efficiency (S/p)	estimate f	Amdahl's law	Ideal-s	Ideal-e
1	132.831102	132.831102	1	1	0.2	1	1	1
3	132.831102	51.98308	2.555275717	0.8517585722	0.2	2.142857143	3	1
6	132.831102	46.629646	2.848640584	0.4747734306	0.2	3	6	1
9	132.831102	42.599232	3.118157201	0.3464619112	0.2	3.461538462	9	1





Speedup achieved by a parallel algorithm is defined as the ratio of the time required by the best sequential algorithm to solve a problem, $T(s)$, to the time required by parallel algorithm using p processors to solve the same problem, $T(p)$. For simplicity, $T(s)$ is calculated by running the parallel program on one processor.

The parallel efficiency of a program is the ratio of the speedup factor and the number of processors. These quantities can be used to estimate how much of the computing capacity is actually used to carry out a calculation.

Amdahl's law is often represented as a graph, called Amdahl's graph, which illustrates the theoretical speedup of a program as a function of the number of processors used. The graph has a horizontal axis that represents the number of processors, and a vertical axis that represents the speedup. The speedup is the ratio of the time taken to complete a task on a single processor to the time taken to complete the same task on multiple processors. The Amdahl's graph has a straight line that represents the maximum achievable speedup, which is limited by the sequential portion of the algorithm. The slope of the line represents the degree of parallelism of the algorithm. The steeper the line, the more parallel the algorithm is. The graph also has a curve that represents the actual speedup achieved by the program as a function of the number of processors used. The curve is always below the maximum achievable speedup line due to the overhead associated with parallelization.

In summary, Amdahl's graph is a visual representation of Amdahl's law, which predicts the theoretical speedup of a parallel algorithm as a function of the number of processors used. The graph illustrates the maximum achievable speedup, which is limited by the sequential portion of the algorithm, and the actual speedup achieved by the program as a function of the number of processors used. The shape of the curve depends on the degree of parallelism of the algorithm and the size of the problem. Amdahl's law and Amdahl's graph are important tools in parallel computing that help developers optimize the performance of their parallel programs.

My experiments produced graphs showing speed-up and efficiency. My program's speed is not as excellent as it should be with a rising number of processes taken into account. The run time is only marginally decreased because it is not split by the number of processes. If a program's efficiency is poor, its value will reflect it in the same way as efficiency graphs do. It is considerably different from the ideal, as seen in the image above. The reason for this is because when the number of threads rises, there will be greater computing power since work will be distributed across several threads working simultaneously, but there will also be overhead during communication. As a result, I should determine the best number of threads for quick and efficient aggregate calculation. Spend no effort on unneeded overhead.

Appendix:

Amdahl's Law

- Let *f* be the fraction of operations in a computation that must be performed sequentially. The maximum speedup achievable by a parallel computer with *p* processors is:

$$s = \frac{1}{f + (1 - f)/p}$$

```
~/Doc/k/year32/hpc/pj1_quicksort ./a.out 1 biginput.txt
Initialize array!
set nthread=1
Time = 132.831102 secs
Sorted array (size 10000000)
~/Doc/k/y/hpc/pj1_quicksort
```

```
~/Doc/k/y/hpc/pj1_quicksort gcc pj1.c -fopenmp
~/Documents/kmutt/year32/hpc/pj1_quicksort ./a.out 3 biginput.txt
Initialize array!
set nthread=3
Time = 51.983080 secs
Sorted array (size 10000000)
~/Doc/k/year32/hpc/pj1_quicksort
```

```
~/Doc/k/year32/hpc/pj1_quicksort ./a.out 6 biginput.txt
Initialize array!
set nthread=6
Time = 46.629646 secs
Sorted array (size 10000000)
```


~/Doc/k/year32/hpc/pj1_quicksort ./a.out 9 biginput.txt
Initialize array!
set nthread=9
Time = 42.599232 secs
Sorted array (size 10000000)