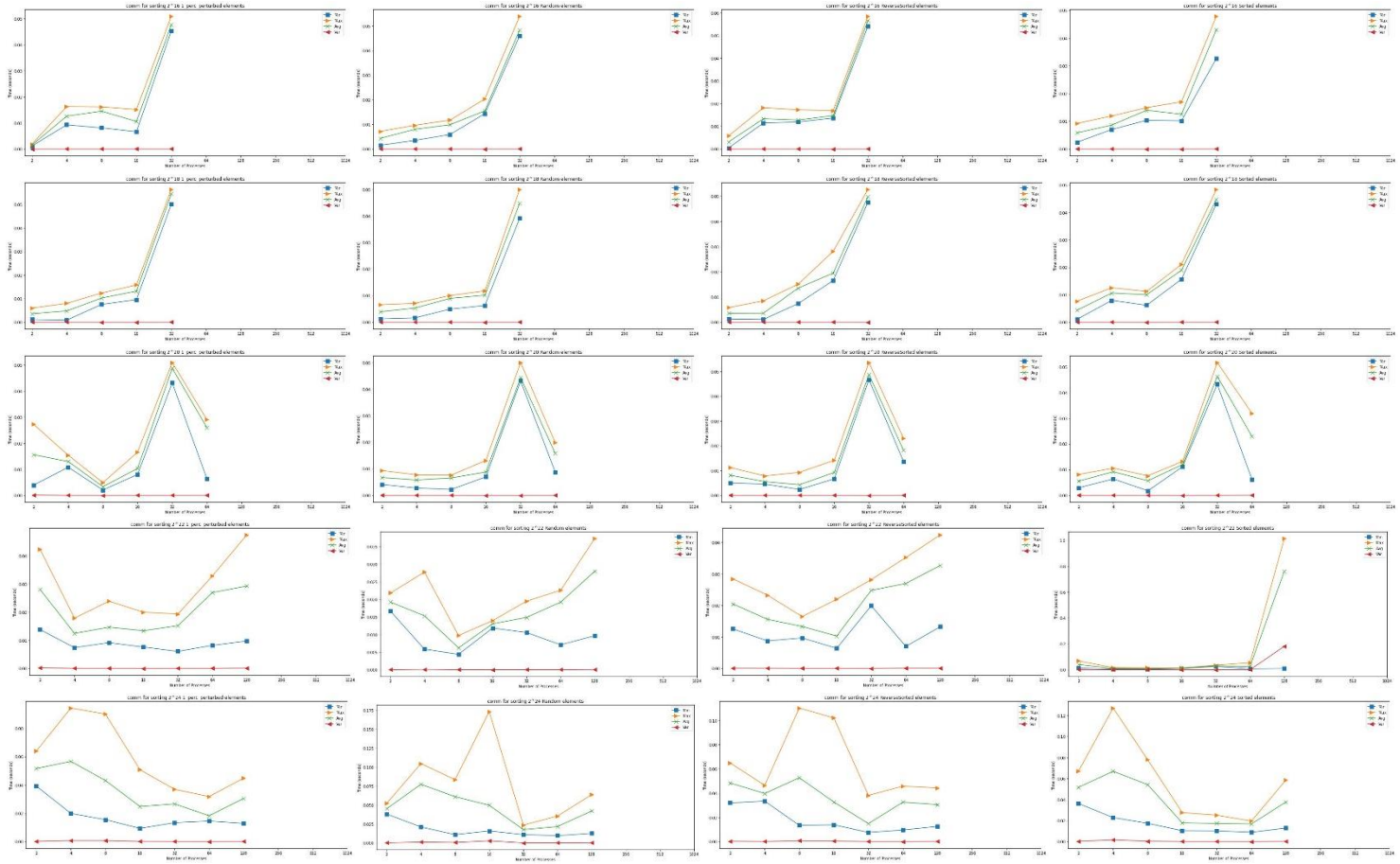
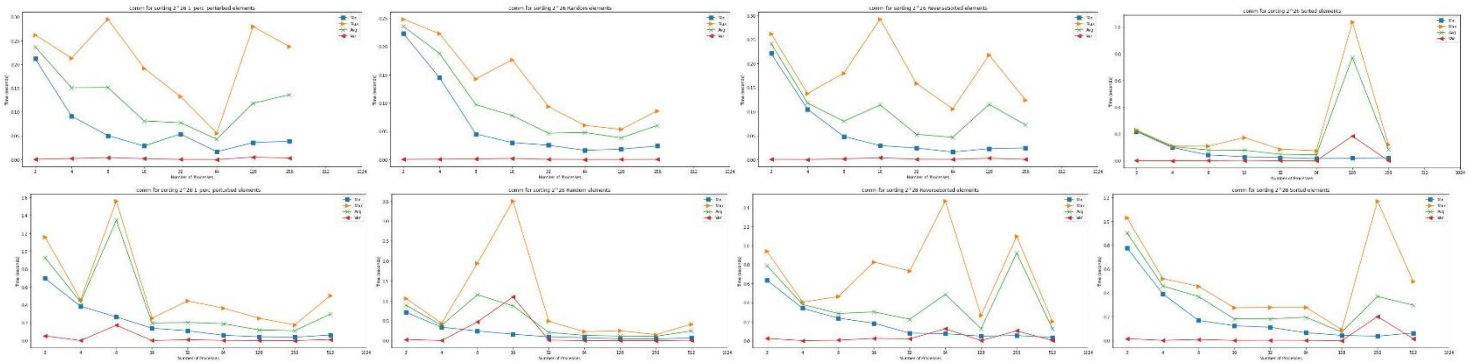
Every row is input size, starting from 2^16.

Every column is input type, in the order of 1% perturbed, Random, Reverse Sorted, Sorted.

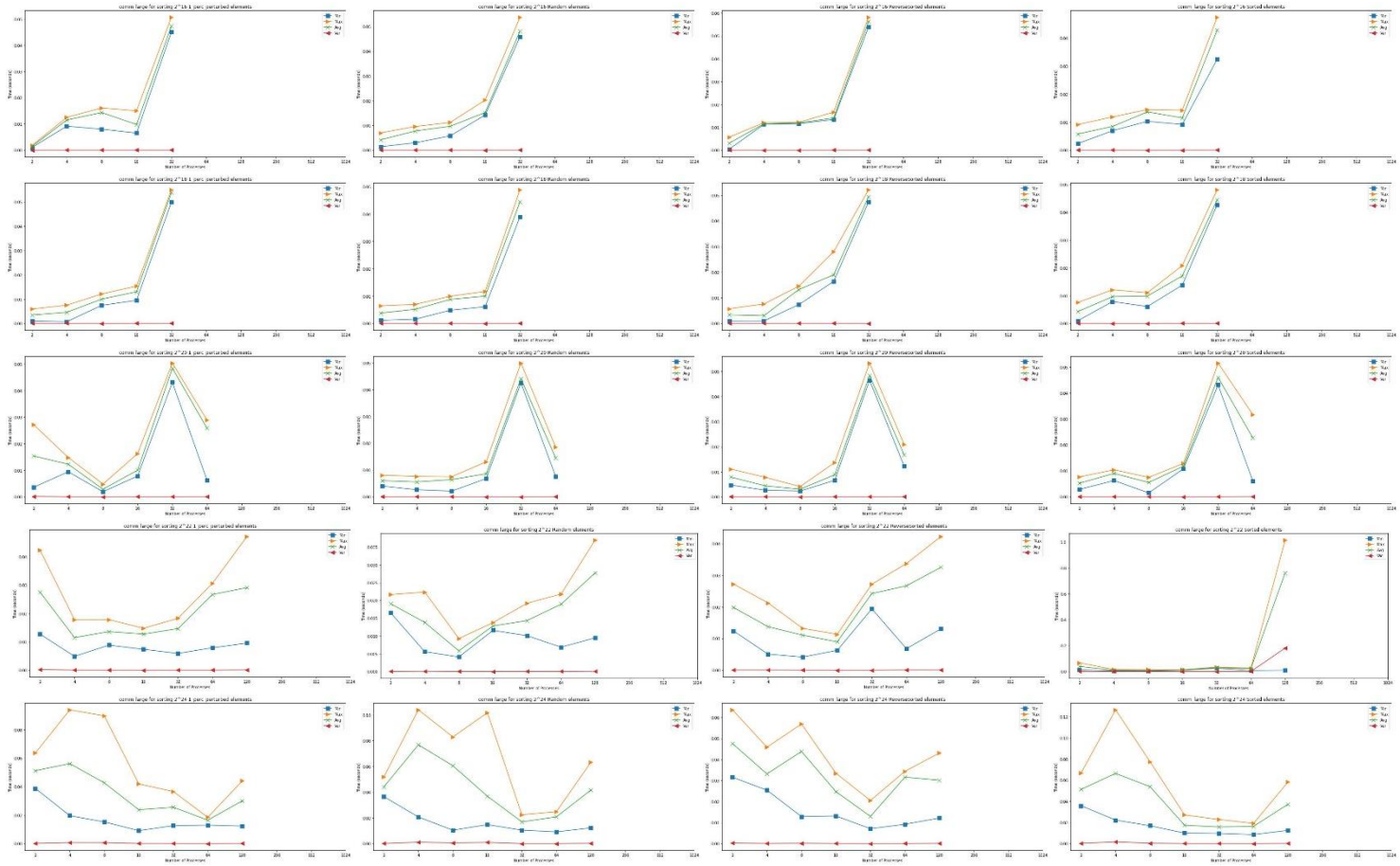
Min/max/avg/variance per Rank Graphs for comm, each row is every input type for every input size, starting with 2^16.

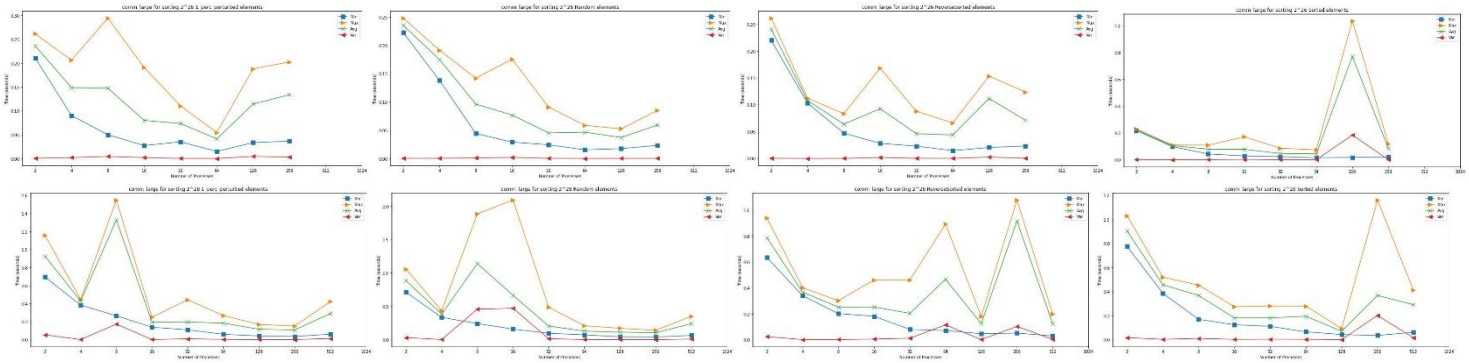




There is no clear trend for the time it takes for communication between processes. Within each input size (row), there are some patterns, suggesting that the input size has an effect on the communication time, but the randomness of the times likely comes from the varying distance between nodes. Everything up to 32 processors (1 node) has a small communication time, but when I expand out to have multiple nodes, the time typically jumps higher. Additionally, as the input size per process increases, the communication time also generally increases as there is more data that needs to be transferred between processes. Variance is also low, suggesting that the communication time is constant for all processors. The communication time is also low, suggesting that there is not much overhead for communicating between processes.

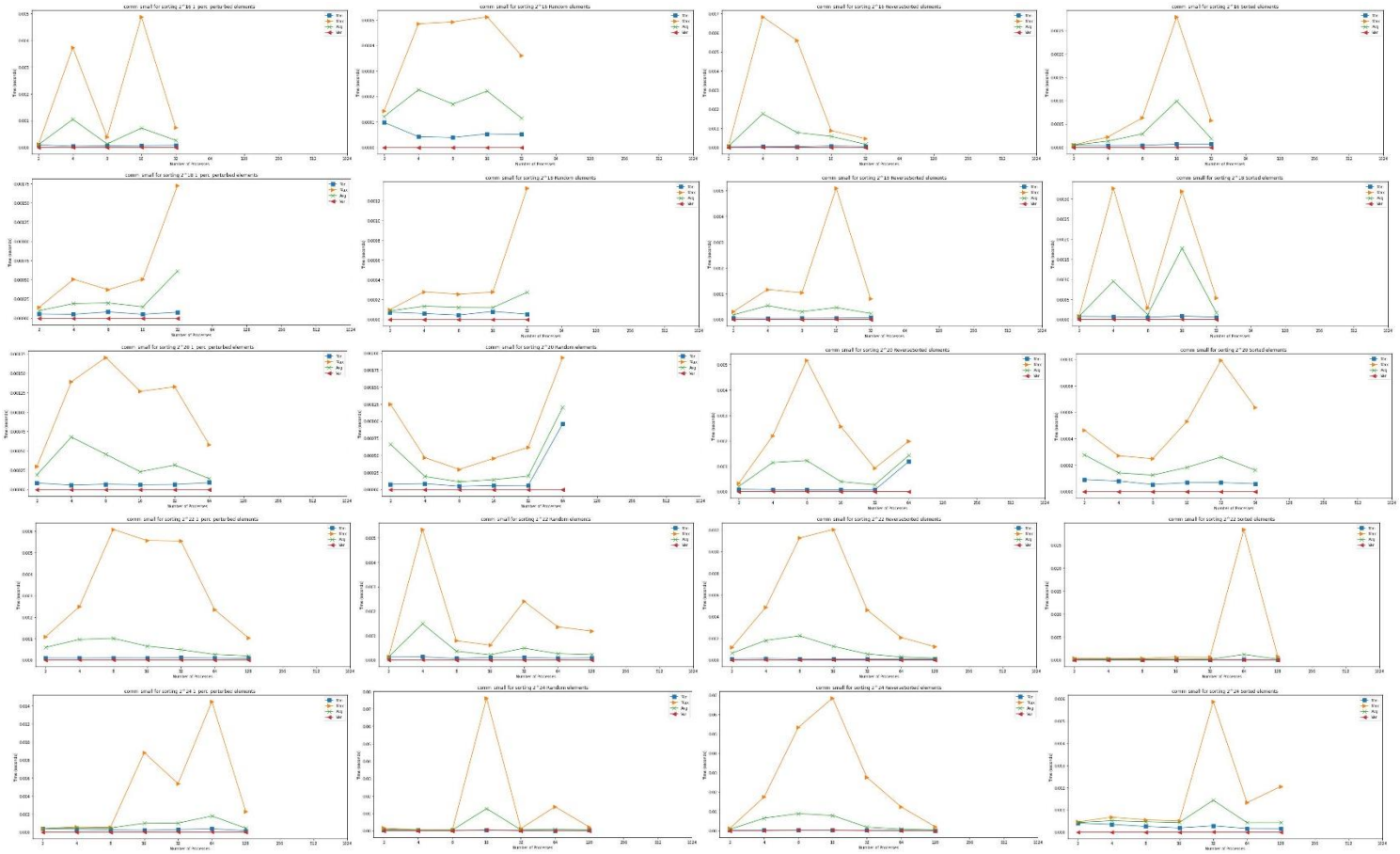
Min/max/avg/variance per Rank Graphs for comm_large, each row is every input type for every input size, starting with 2^16.

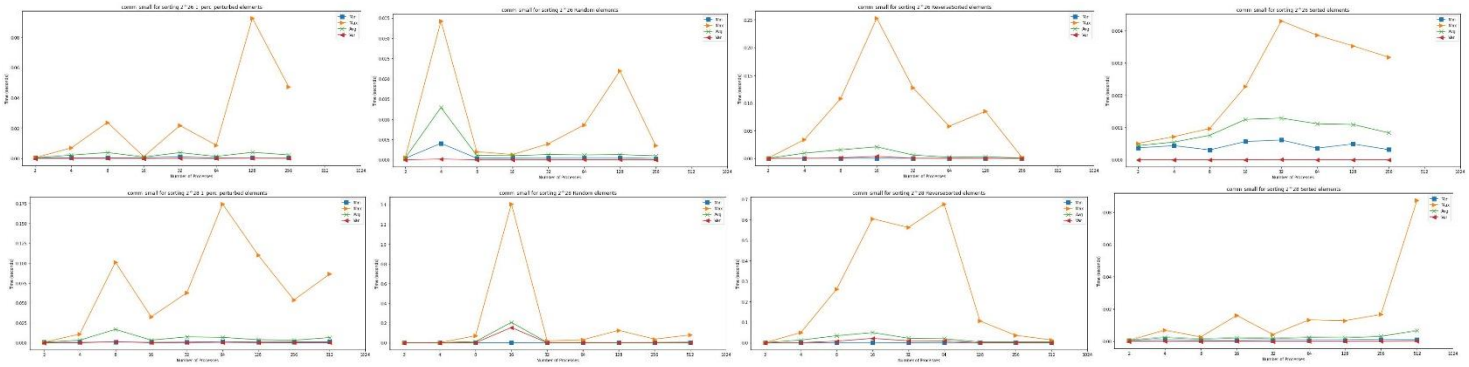




There is no clear trend for the time it takes for communication between processes. Within each input size (row), there are some patterns, suggesting that the input size has an effect on the communication time, but the randomness of the times likely comes from the varying distance between nodes. Everything up to 32 processors (1 node) has a small communication time, but when I expand out to have multiple nodes (64 processors and higher), the time typically jumps higher before slowing decreasing. Additionally, as the input size per process increases, the communication time also generally increases as there is more data that needs to be transferred between processes. This has more variance than the overall comm, suggesting that hardware variance affects this more than comm_small.

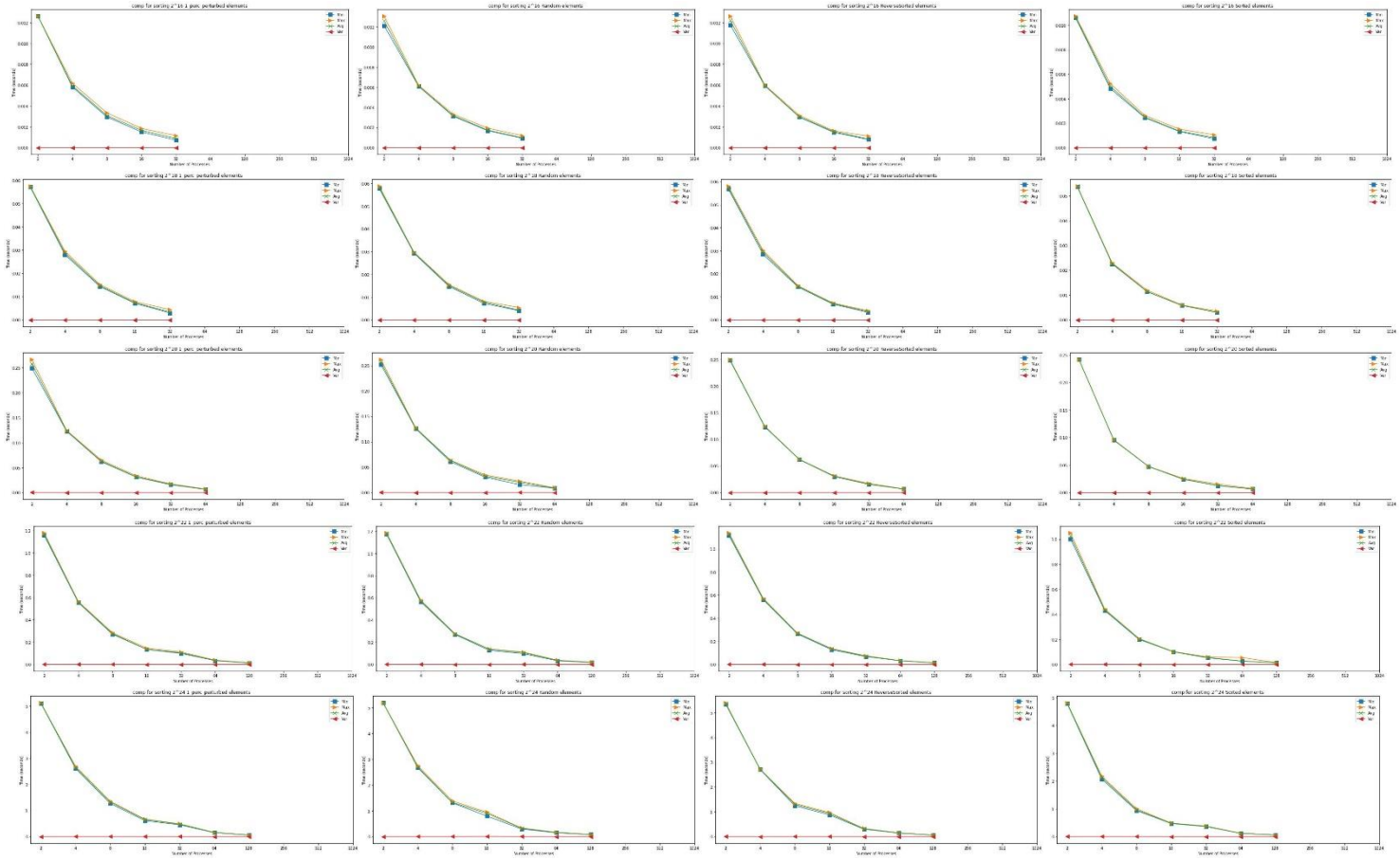
Min/max/avg/variance per Rank Graphs for comm_small, each row is every input type for every input size, starting with 2^16.

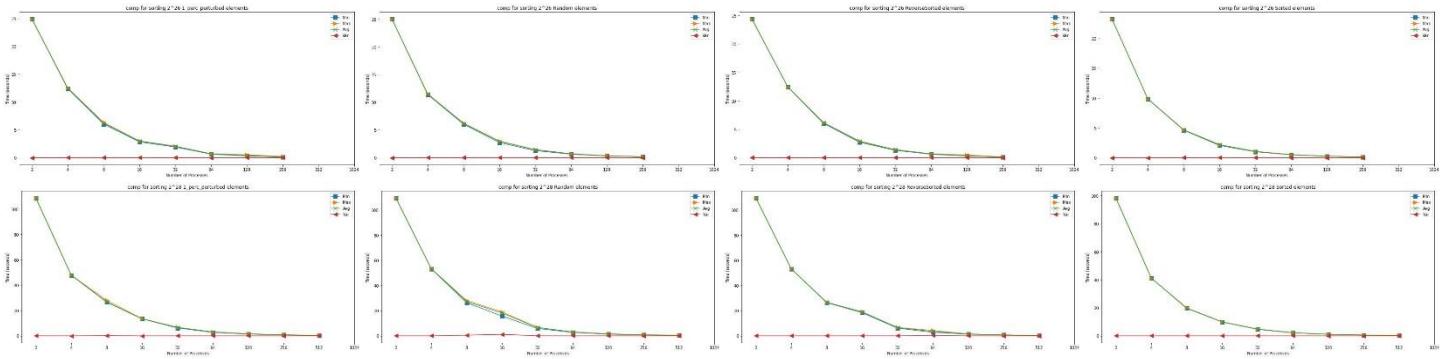




These all look like hats! For all of them, the maximum communication time is a lot higher than the min and average communication time, despite having low variance. This means that a few processes spent a long time communicating between other processes. This may be due to their distance from the nodes that it needed to communicate to, or it could just be hardware variation because these times are extremely small.

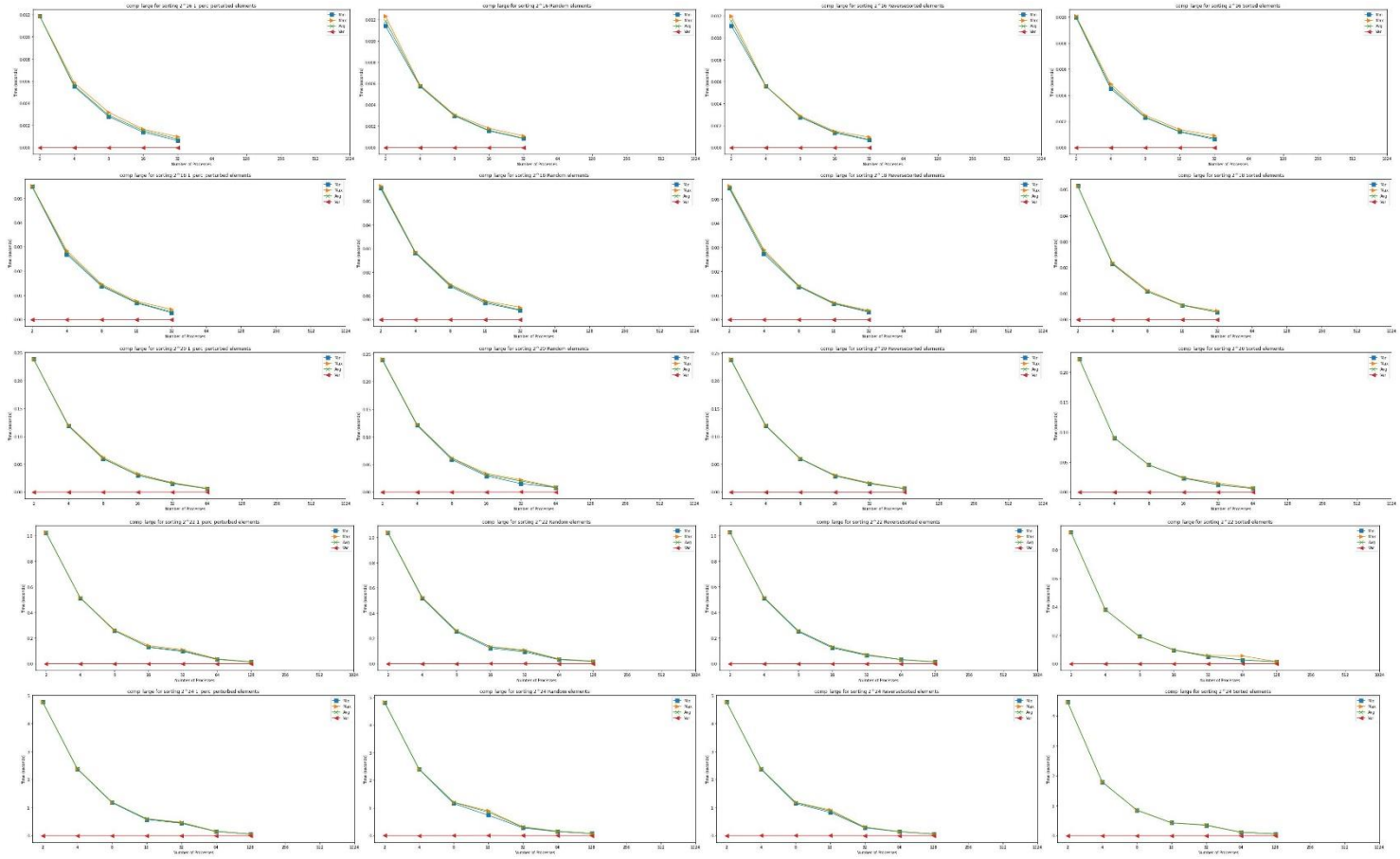
Min/max/avg/variance per Rank Graphs for comp, each row is every input type for every input size, starting with 2^16.

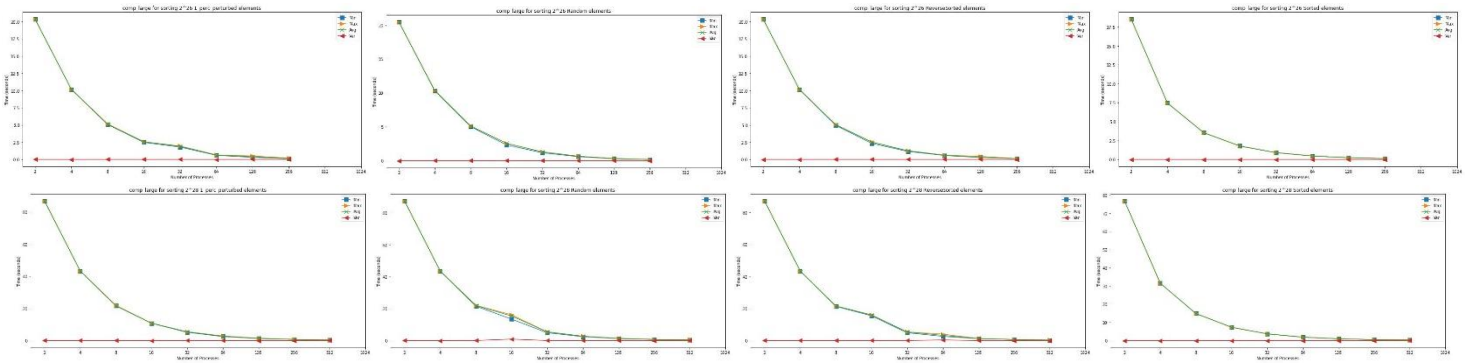




In all of the graphs, there is a clear pattern in the time it takes to compute and the number of processes. The line suggests an inverse linear relationship between the computation time and the number of processes, which is expected because the parallelization of the computation will tend towards an inverse relationship, especially when the algorithm can be completely parallelized. Additionally, the min, max, and average time of each process are extremely similar. This can be explained by every process handling the same amount of data and making the same operations on the data. Because of this, every process will have a similar number of cache misses and hits, so the amount of time loading from memory, which can have a large variability, will be similar, and this is reflected with the low variance.

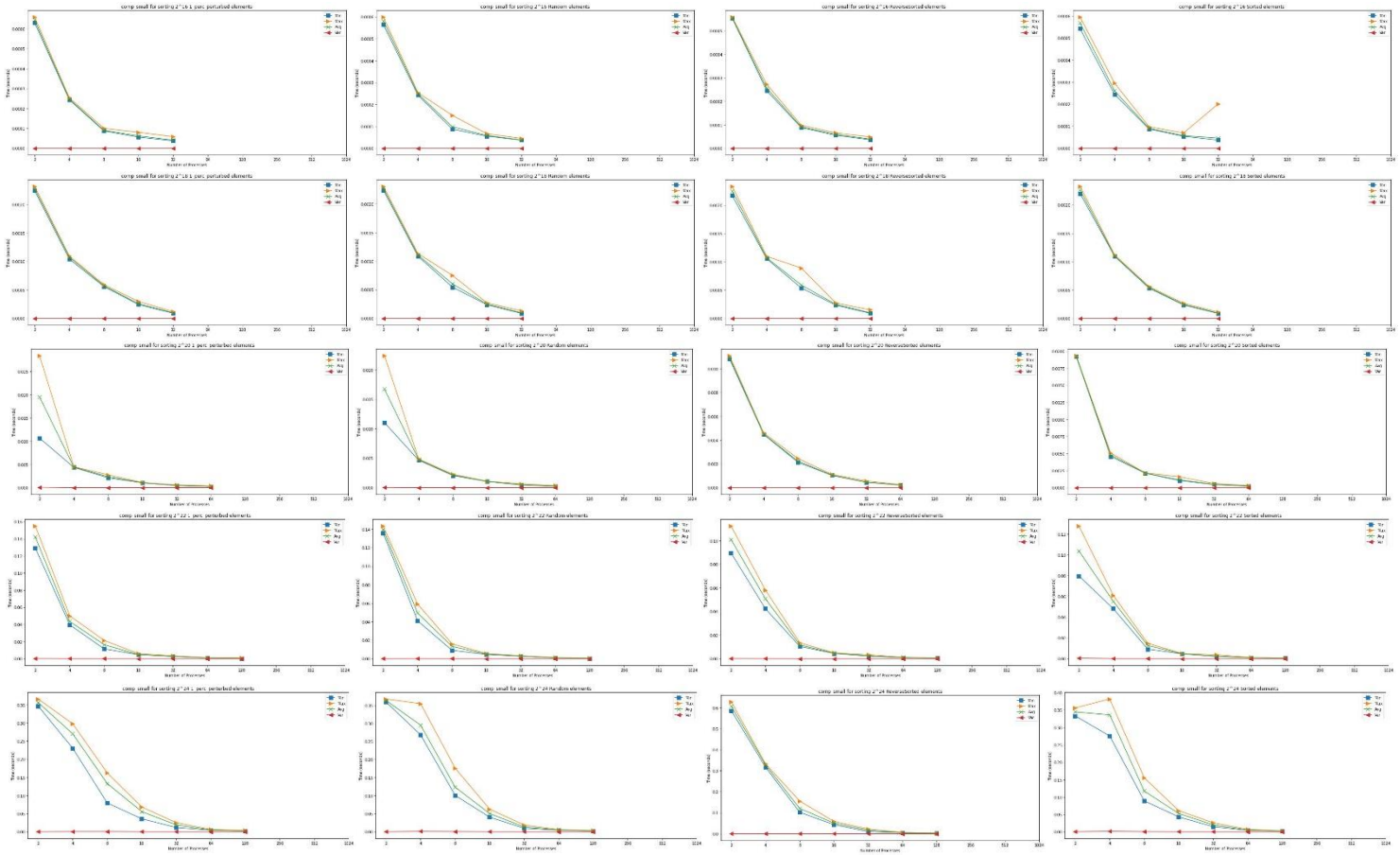
Min/max/avg/variance per Rank Graphs for comp_large, each row is every input type for every input size, starting with 2^16.

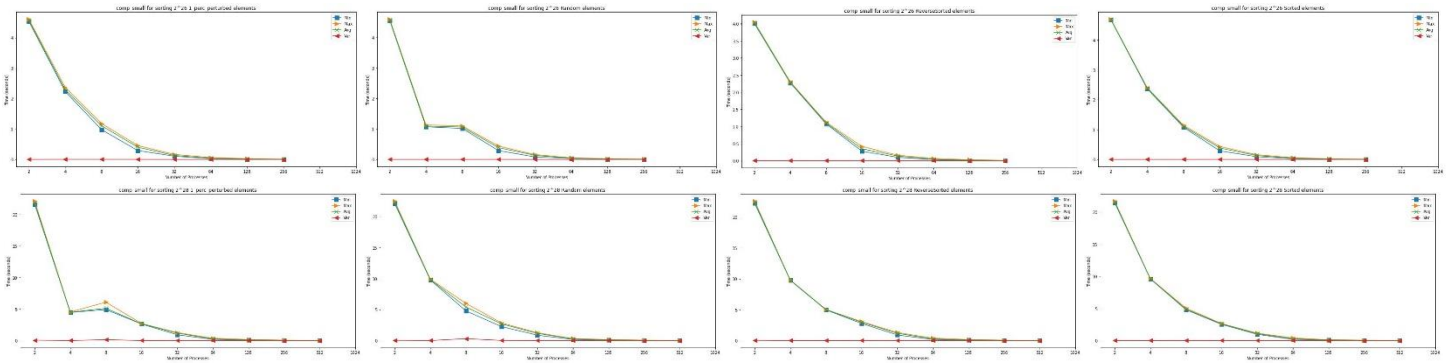




In all of the graphs, there is a clear pattern in the time it takes to compute and the number of processes. The line suggests an inverse linear relationship between the computation time and the number of processes, which is expected because the parallelization of the computation will tend towards an inverse relationship, especially when the algorithm can be completely parallelized. The graphs measure the total time the algorithm sorts the columns of the matrix. Because the number of columns per process decreases as the number of processes increases, there are less columns for each process to sort, decreasing the time.

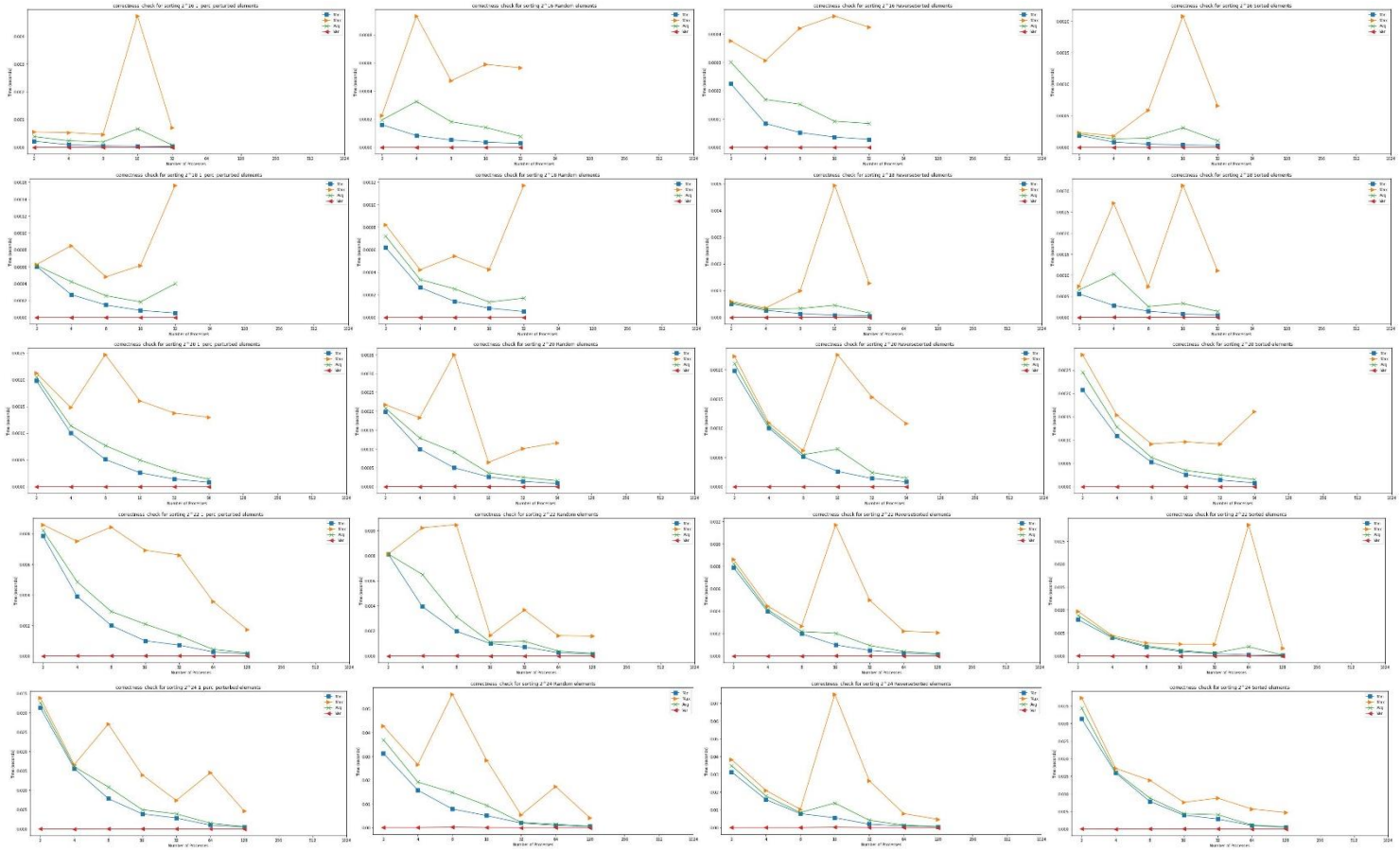
Min/max/avg/variance per Rank Graphs for comp_small, each row is every input type for every input size, starting with 2^16.

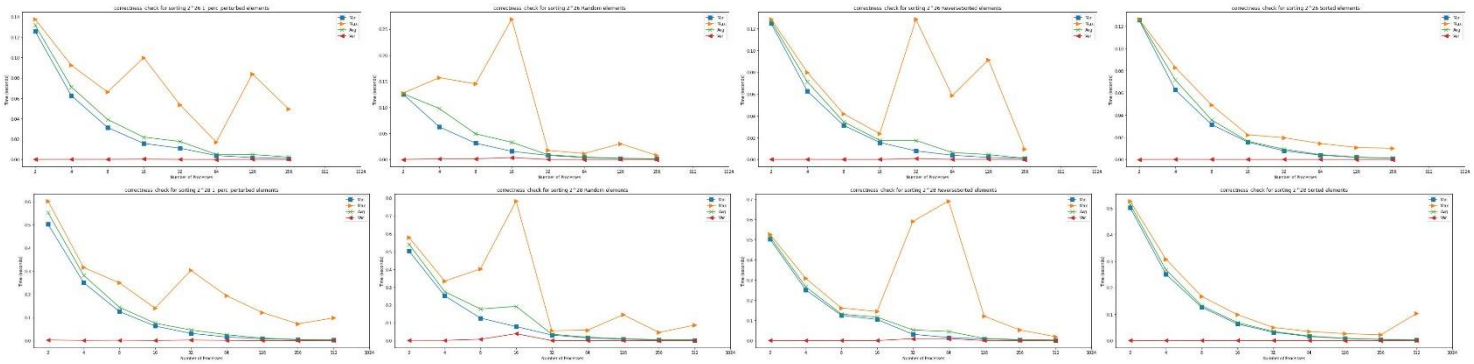




Once again, there is a clear pattern in the time it takes to compute and the number of processes. The line suggests an inverse linear relationship between the computation time and the number of processes, which is expected because the parallelization of the computation will tend towards an inverse relationship, especially when the algorithm can be completely parallelized. These graphs measure the amount of time the algorithm is performing operations on the data that does not involve sorting the data, and they show that the algorithm spends more time sorting the columns than transposing, untransposing, shifting, and unshifting the matrix.

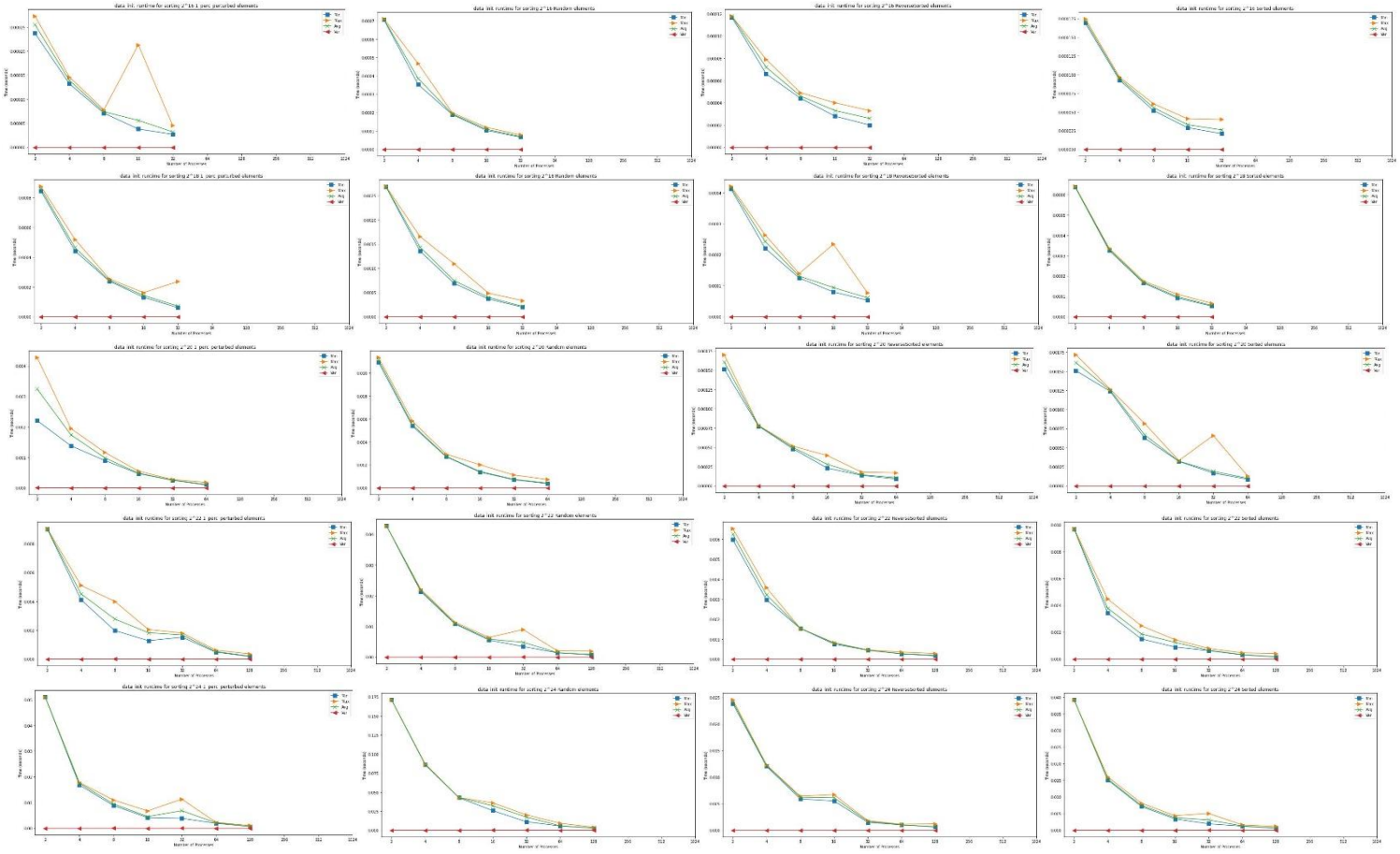
Min/max/avg/variance per Rank Graphs for correctness_check, each row is every input type for every input size, starting with 2^16.

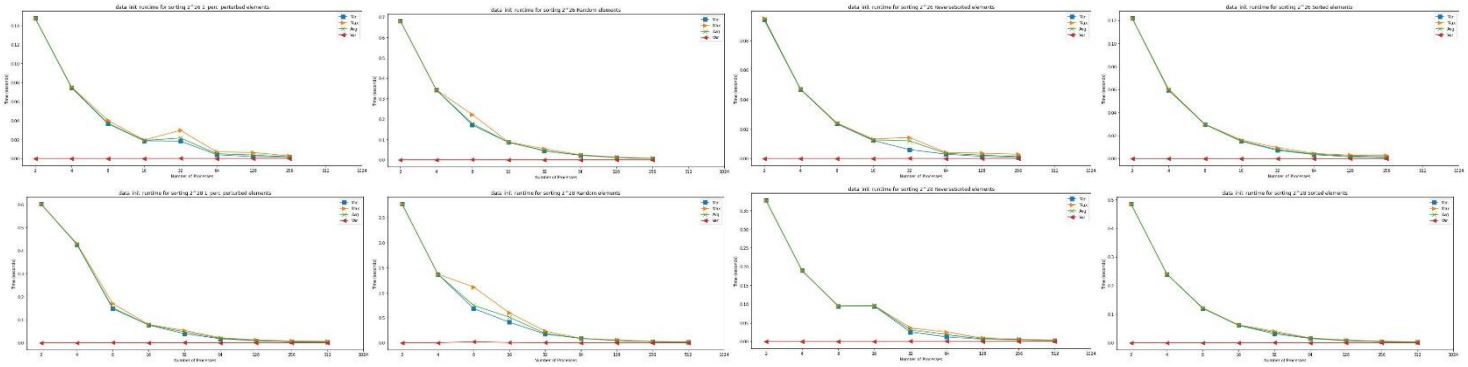




The graphs show a generally inversely proportional relationship to the time it takes to check where the matrix is sorted and the number of processes. This is expected because, as the number of processes increases, the less elements each process needs to iterate over to check for sorted output. Additionally, the maximum times can be attributed to the distance between processes because each process will send one element to the next element for that process to check. This will check the sort between processes since the matrix is split between all of them. The communication can take longer depending on the distance between the communicating processes.

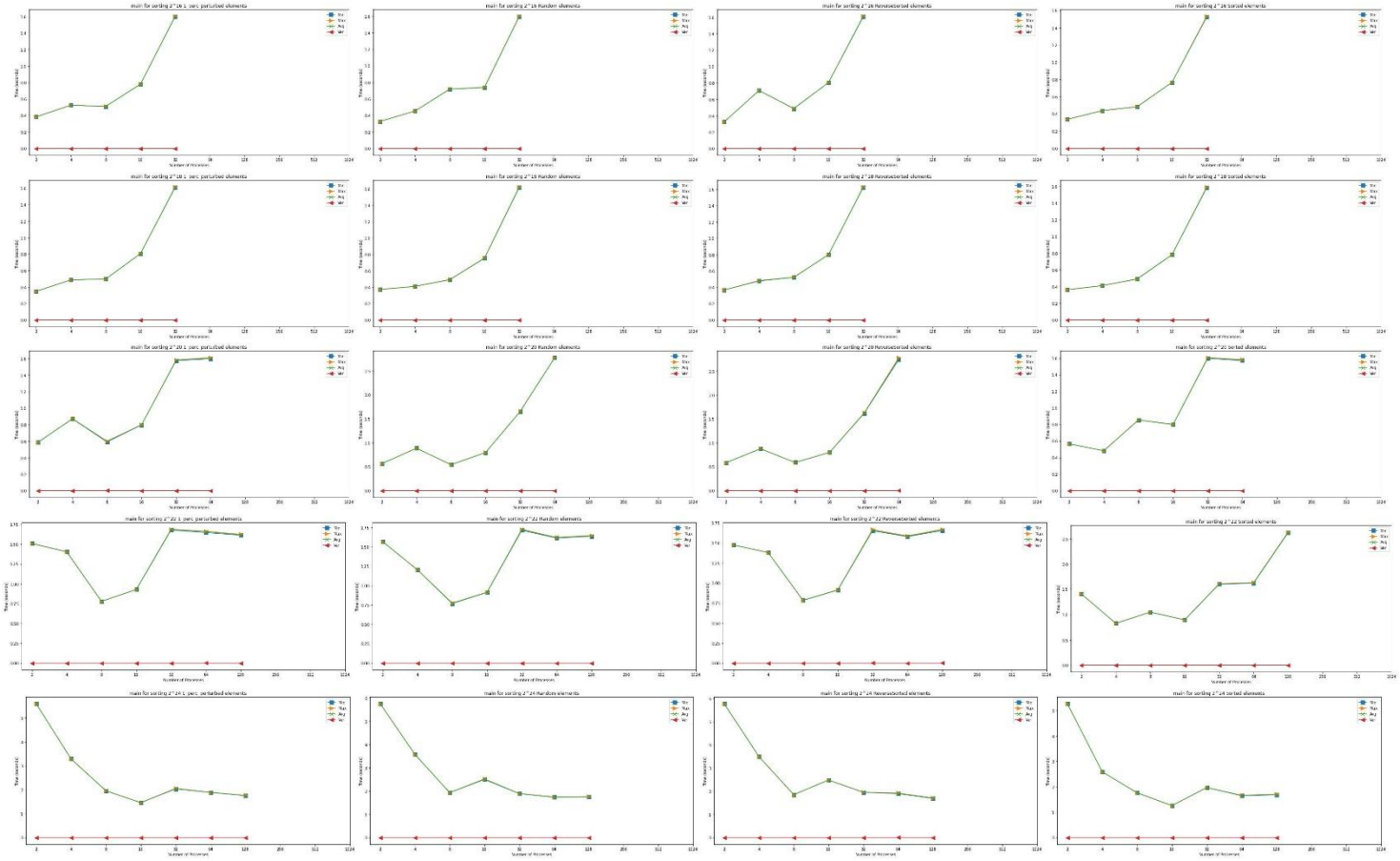
Min/max/avg/variance per Rank Graphs for data_init_runtime, each row is every input type for every input size, starting with 2^16.

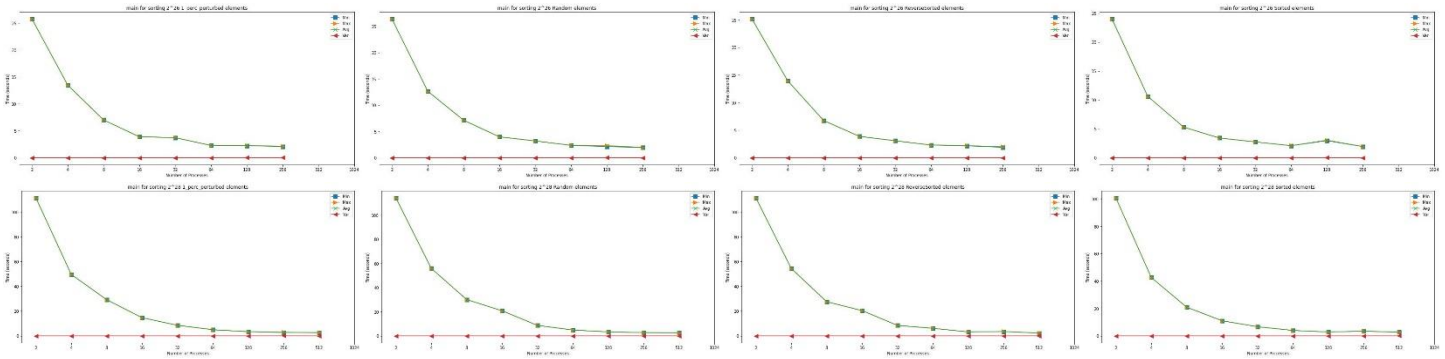




The graphs show that the number of processes and the time to generate data has an inversely proportional relationship. This is expected because each process will generate its own data, rather than a master process generating all the data and then sending it to each process. As the number of processes increases, the less data each process needs to generate, so the time will decrease in proportion to the amount of data that the process needs to generate.

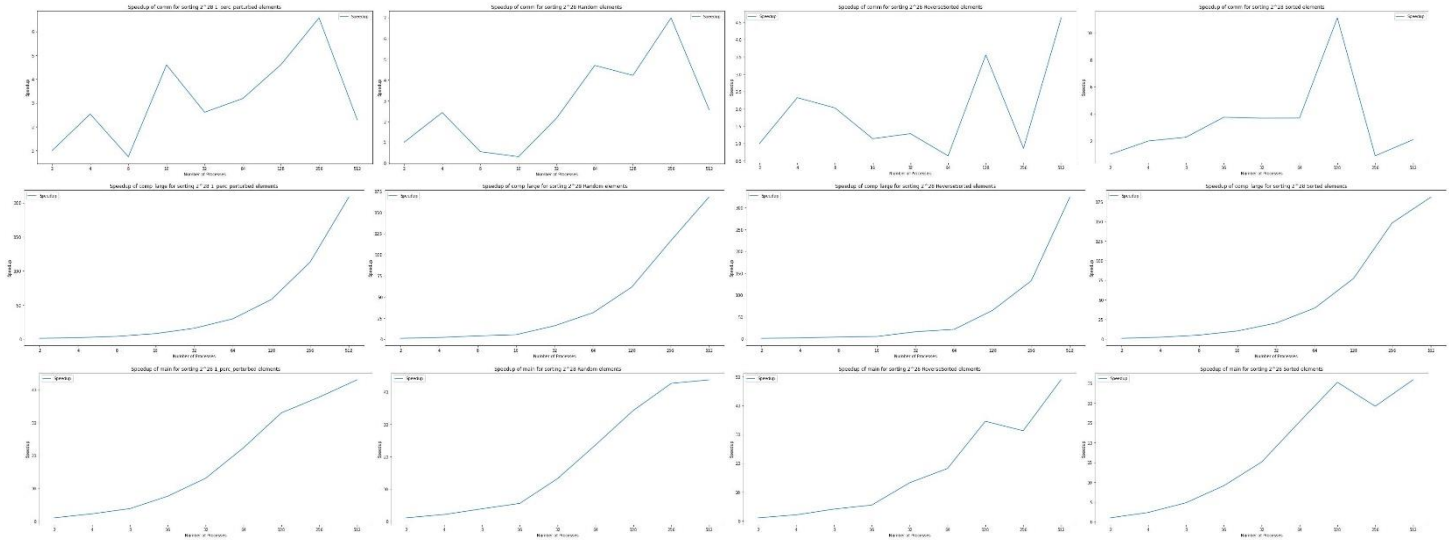
Min/max/avg/variance per Rank Graphs for main, each row is every input type for every input size, starting with 2^16.





These graphs are more interesting than the others. The graphs start out by showing that time to run main actually increases as the number of processes increases. This can be explained by the overhead costs of creating multiple processes and assigning ranks to them. Once the input size reaches 2^24, however, the graphs start to transform into the inversely proportional graphs that is expected of the algorithm. This means that the overhead cost of using more processes only starts to be fully offset when the input size is greater than or equal 2^24, so using less processes on input sizes that are less than 2^24 would be more efficient.

Graphs for Speedup of comm, comp_large, main for input size of 2^28 and all input types

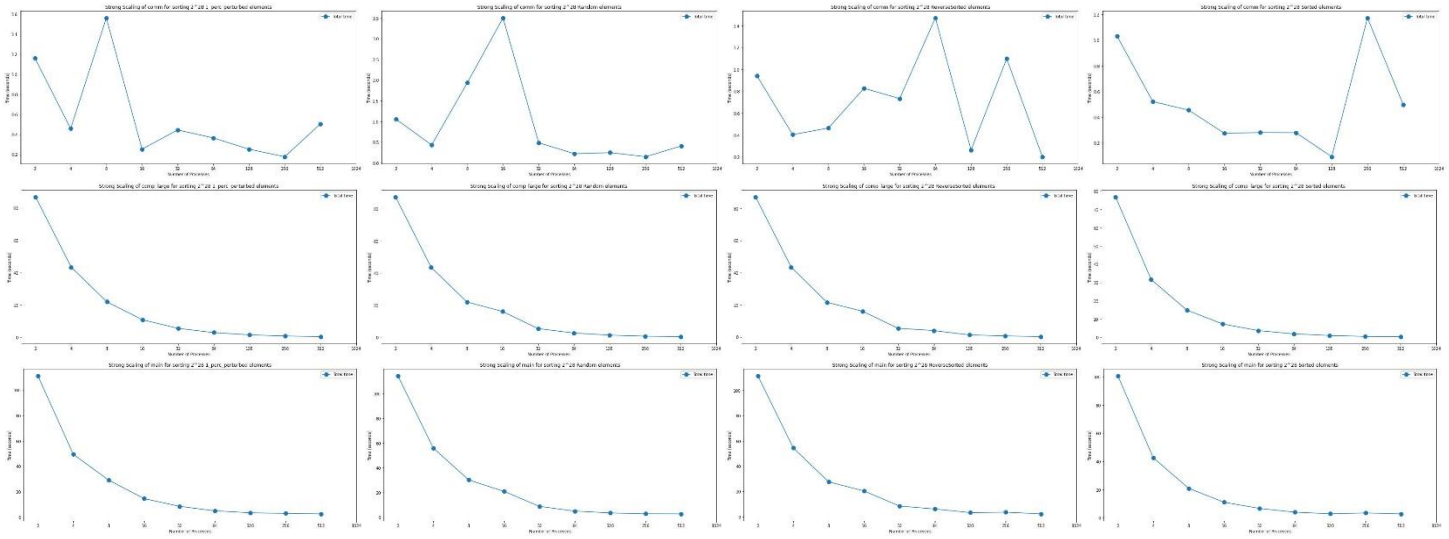


For both 1% perturbed and Random input types, the speedup for communication between processes maxes out at 256 processes, with about a 7x speedup compared to 2 processes. But, for Reverse Sorted, the speedup maxes out at 512 processes, with about a 4.5x speedup, and for Sorted the speedup maxes out at 128 processes, with about an 11x speedup. This implies that the speedup is not dependent on the algorithm itself, but rather the hardware.

The speedup for comp_large maxes out at 512 processes, with about a 175x speedup compared to 2 processes. This is consistent across all the graphs, so it implies that the speedup is dependent on the algorithm itself, which is expected because as the number of processes increases, the less columns each process must sort.

The speedup for main maxes out at 512 processes, with about a 40x speedup compared to 2 processes, for all input types. This means that, despite the speedup for communication not being very high for 3 of the input types, the speedup for the computation completely overshadowed it. This means that if 512 processes is used for this algorithm, it is expected that it is the most efficient.

Graphs for Strong Scaling of comm, comp_large, main for an input size of 2^28 and all input types

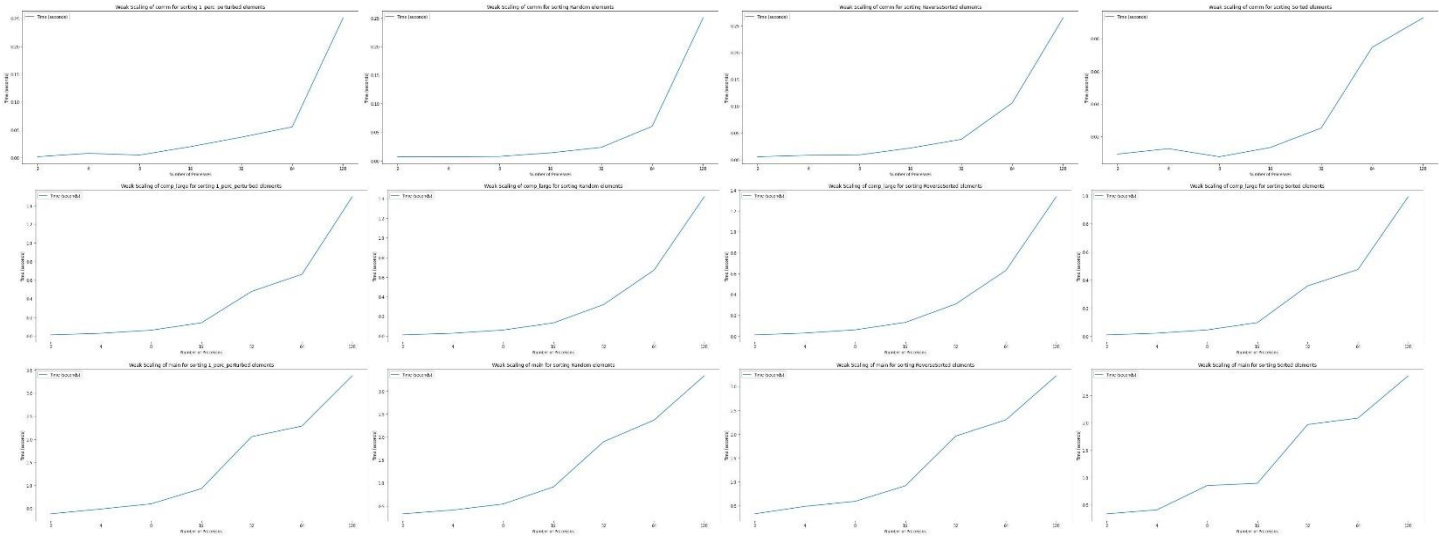


The graphs for comm are very random and do not indicate any scaling whatsoever. If anything, it shows that it is highly variable. However, this could be due to how small the values are.

The graphs for comp_large indicate that the computation is strongly scaled, meaning that parallelization is very effective on the computation. The graphs indicate an asymptote of about 1 second, meaning that the sequential portion takes about 1 second to run, and that it will be difficult to see great performance speedups on sorting the columns from adding more parallelization.

The graphs for main indicate that the algorithm is strongly scaled, meaning that parallelization is very effective on the algorithm. The graphs indicate an asymptote of about 2 seconds, meaning that the total sequential portion of the algorithm takes about 2 seconds to run. Since the asymptote was reached for the last 4 number of processes, this means that it will be extremely difficult to see performance gains from more parallelization without reducing the sequential runtime.

Graphs for Weak Scaling of comm, comp_large, main, from 2^16 to 2^28, and for all input types



The graphs for communication somewhat indicate that the communication, computation, and the entire algorithm is weakly scaled. It shows a linear relationship between the time and the number of processes as the input size increases, which is expected because the number of processes doubles every time the input size quadruples, meaning that we should expect the work per process to double.

Other observations and notes

The algorithm does not care about the input type. It will always take about the same amount of time to sort 1% perturbed (almost sorted), random, reverse sorted, and sorted inputs.

The number of processes that can be used for the algorithm is bounded by the maximum number of columns. This is because the algorithm only works under the restriction that

numRows > (numCols – 1)^2 *2,

numRows mod numCols = 0

So, for an input size of 2^16, the maximum number of columns is 32, so the maximum number of processes is also 32. For all of the input sizes given, 1024 processes could not be used because there was no way to increase the maximum number of columns to at least 1024 while still satisfying the restrictions. Additionally, the algorithm cannot split a column in half as that would be equivalent to slicing the number of rows by half, and since the algorithm is using the maximum number of columns, the restrictions would not be satisfied.