Part 1)

Part 2)

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Both charts in part C and D are actually very similar. They both experience exponential growth at about the same rate. This is because the time of computation grows at the same rate as the number of elements in the array. There are more elements to compare and more step to complete the insertion sort. Also, I thought that the specifications of your computer would mitigate the reason of change, but because the amount of elements increase along with the amount of elements needing to be compared, the computer would still have to compare them all so that the rate of change would be similar.

Part 3)

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

The Part C graph shows the time of computation versus the number of elements. The graph with the exception of the array of one element is a constant graph. In other words, the slope is close to 0 for Part C. Every execution of the merge sort took about 1.2 seconds. The Part D graph is majorly a linear function graph (y=x). This graph shows the number of comparisons versus the number of elements. In this graph the array of one element had 0 comparisons while the array of 1000 elements had an amount of around 9500 comparison give or take. Both of these graphs are different.

When comparing graphs 2C and 3C, we can see that graph 2C is exponential while graph 3C is more or less constant. It could be because my creation of the merge sort is threaded. Also, we could see that in graph 2C, the times are generally better, however, since the graph shows an exponential time increase for amount of elementats in the array sorted, there will be a point where for a large amount of unsorted elements, the threaded merge sort will have better execution time.

Now comparing graphs 2D and 3D, we can see that graph 2D is exponential and graph 3D is linear. This means that the insertion sort will exponential much higher number of comparsions than the merge sort. Also, we can see throughout the majority of both graphs, the merge sort had very much less amount of comparisons compared to the insertion sort.

Between merge sort and insertion sort, I would say the overall best sorting algorithm would be merge sort because there are very much less amount of comparisons, and for very large arrays, the execution time would be constant whereas in the insertion sort, it will exponentially rise.