**Annotation to overlay graph:**

In this graph we can see several interesting patterns.

Firstly, when the array size increases, the time to sort it increases as well. That obviously makes sense, but yet it is beautiful to see.

Furthermore, it seems that there’s a pivot point where the number of recursions is around 20 – 25. Before and after that point, the sorting takes a bit longer. It is easier to see on the larger sized arrays.

Lastly, something very odd to me, is that in almost every array size we have at least one spike, pointing upwards. I don’t know to explain it, especially when these time values are an average of three runs; so if something happened during one run it should have been leveled down. The highest density of the spikes occurs between 114 to 202 recursions.

**Zoomed overlay graph:**

I decided to focus my graph right around the pivot point since it’s the purpose of this lab: to find the ideal number of recursions to sort an array. As mentioned before, it seems that the pivot points to each graph occur when the recursion number is around 20+-. When thinking about that, it makes sense. We used quicksort to sort the elements in the array. Quicksort divides the array by half on every recursion, so if we take an array size of 1 million to 10 million, and keep dividing it by 2, we need 19-24 recursions (log base 2 of that number). When we do more or less recursions than it needs, it doesn’t help the array to make it faster.