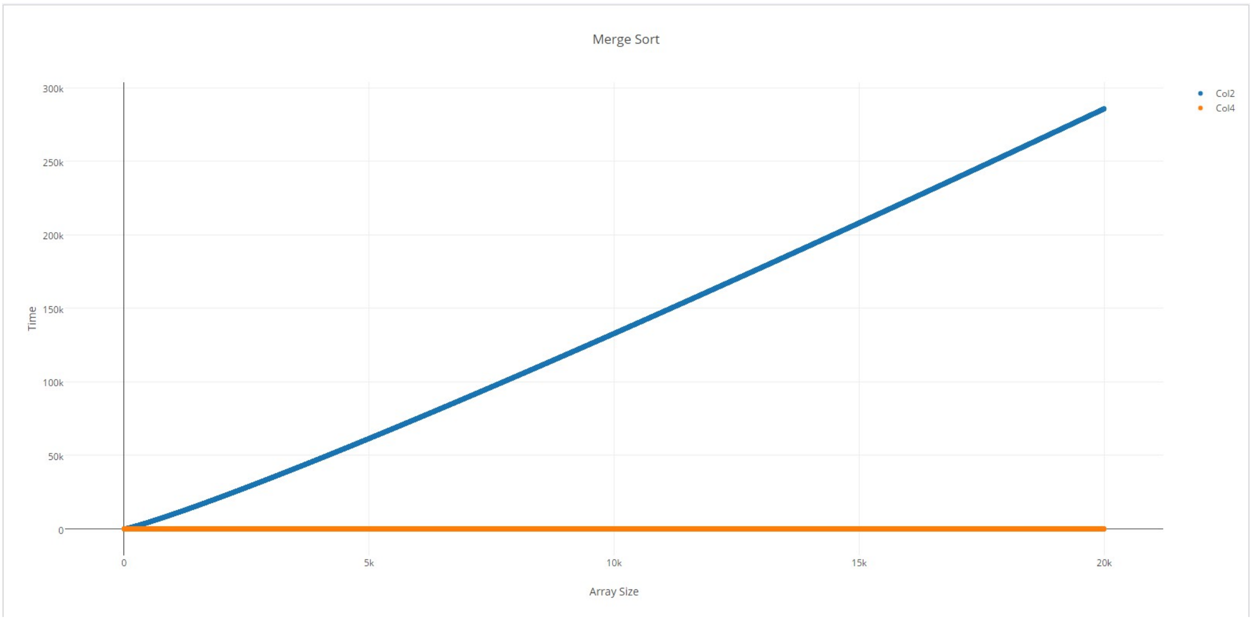
This time around I've decided to test three different algorithms because their behavior should be interesting. The algorithms I've chosen are Merge Sort, Heap Sort, and Counting Sort. I chose Merge Sort and Heap Sort because they boast a best, worst and average case performance of O(n log n). So I wanted to see which of the two actually perform better on my system. I chose to include Counting Sort because all of my inputs are going to be random integers from 0 to 1000 so at some point counting sort should have performance O(n).

The way my solution works is it makes and array of X random elements, starts a timer sorts the array and stops the timer, then records that time in a file. This process is done on a loop from 1 to 20,000.

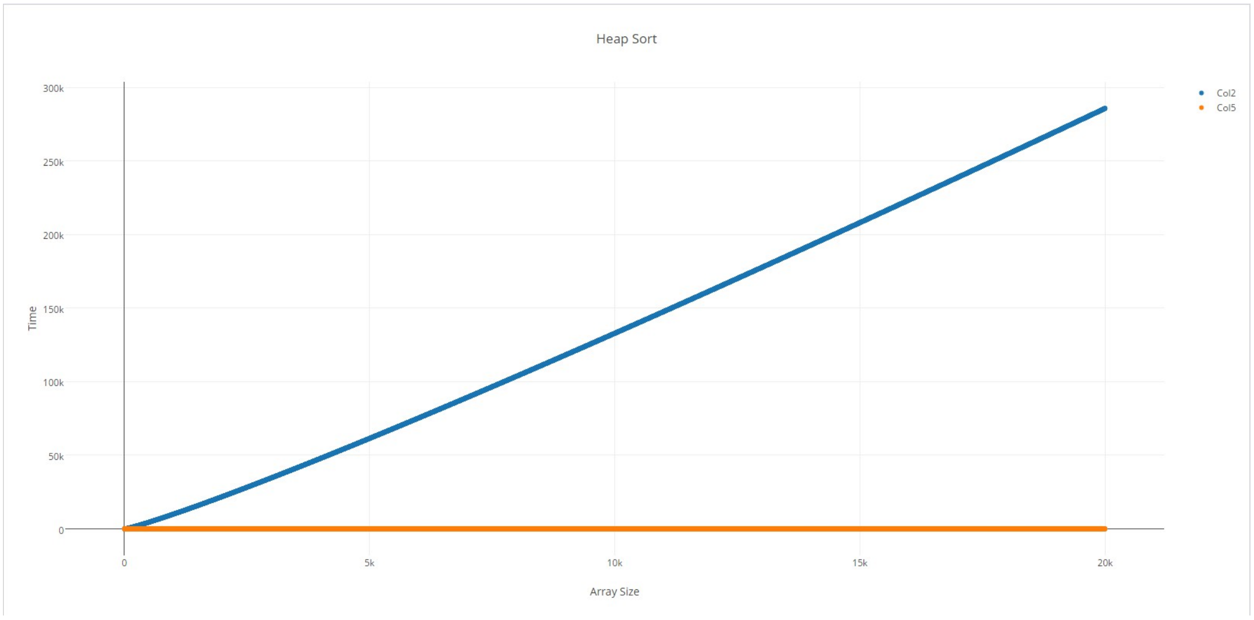
The resulting Graphs for the 3 algorithms are as follows:

(Blue line is N log N for Merge and Heap Sort and N for Counting Sort.

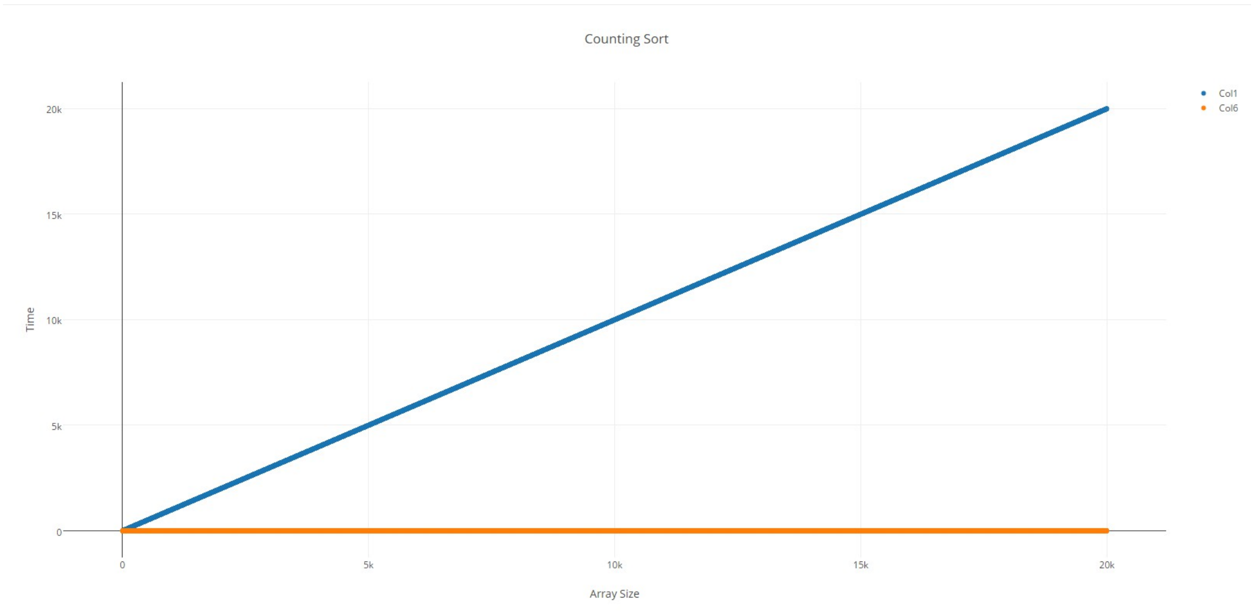
Merge Sort:



Heap Sort:



Counting Sort:



From these original graphs I wanted to test if my original hypothesis would hold true as the value for X got larger. So I reran all 3 algorithms for X = 70,000 and the results were interesting. My hypothesis didn't hold true. As the numbers got larger Heap Sort outperformed Merge Sort, and Counting sort was greatly superior.

45k Bench Mark:

Merge Sort: 0.450343779576

Heap Sort: 0.328048297466

Counting Sort: 0.247777442426

60k Bench Mark:

Merge Sort: 0.572350034268

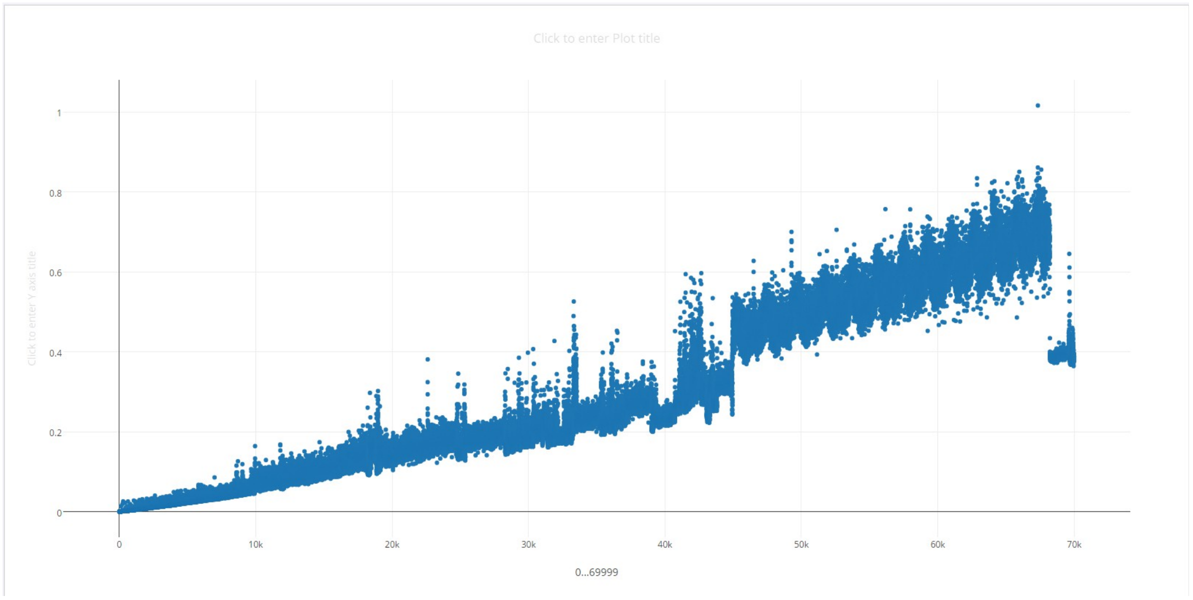
Heap Sort: 0.47849193076

Counting Sort: 0.393967266369

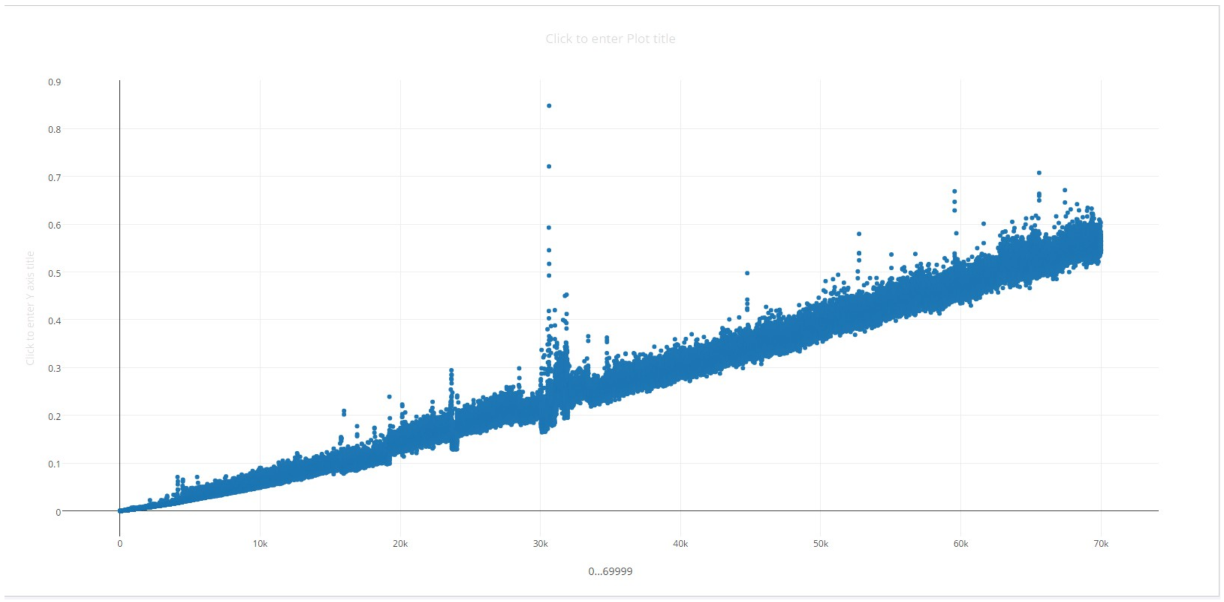
After I reran the algorithms for 70,000 the graphs looked a lot different.

(These aren't graphed next to their expected asymptotic complexity)

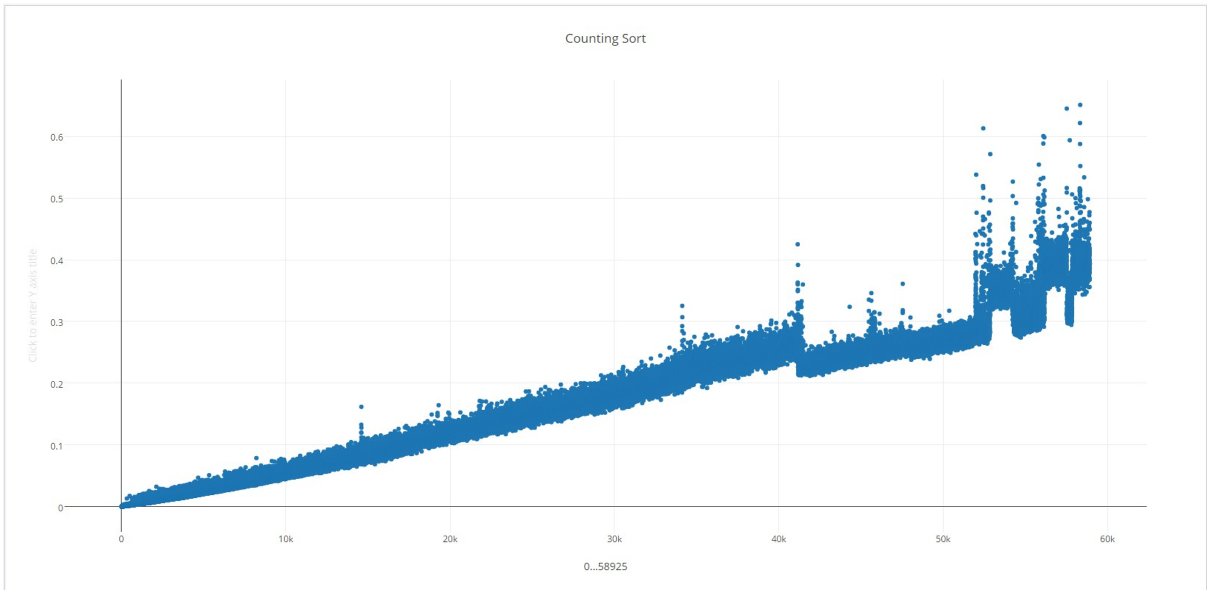
Merge Sort:



Heap Sort:



Counting Sort: (only goes to 60k)



From the graphs and the bench marks we can see that while all the algorithms are extremely quick, counting sort is the best when we know something about the input.

The next task was to find constants C1 and C2 to try and bound one of my algorithms, I ended up with

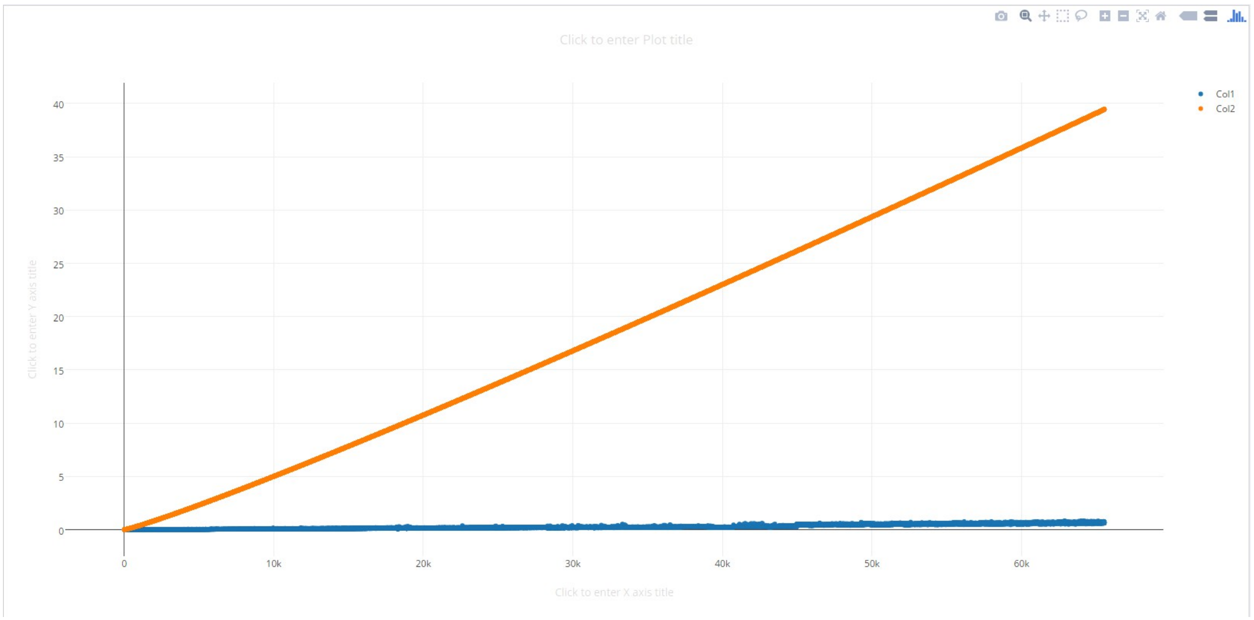
I chose to use Merge Sort. The results were as follows:

C1 = 1/800,000

C2 = Left this as 1 because there was no need to inflate G(n)

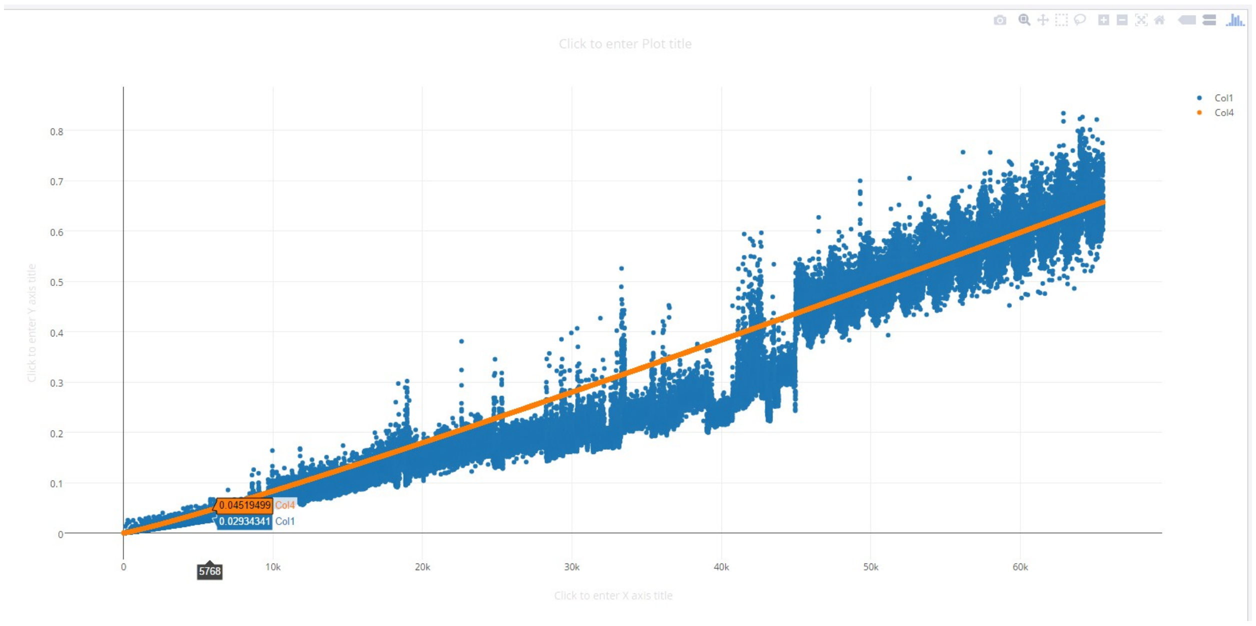
N0 = Approx. 46,000

This was my attempt at c = 1/8000



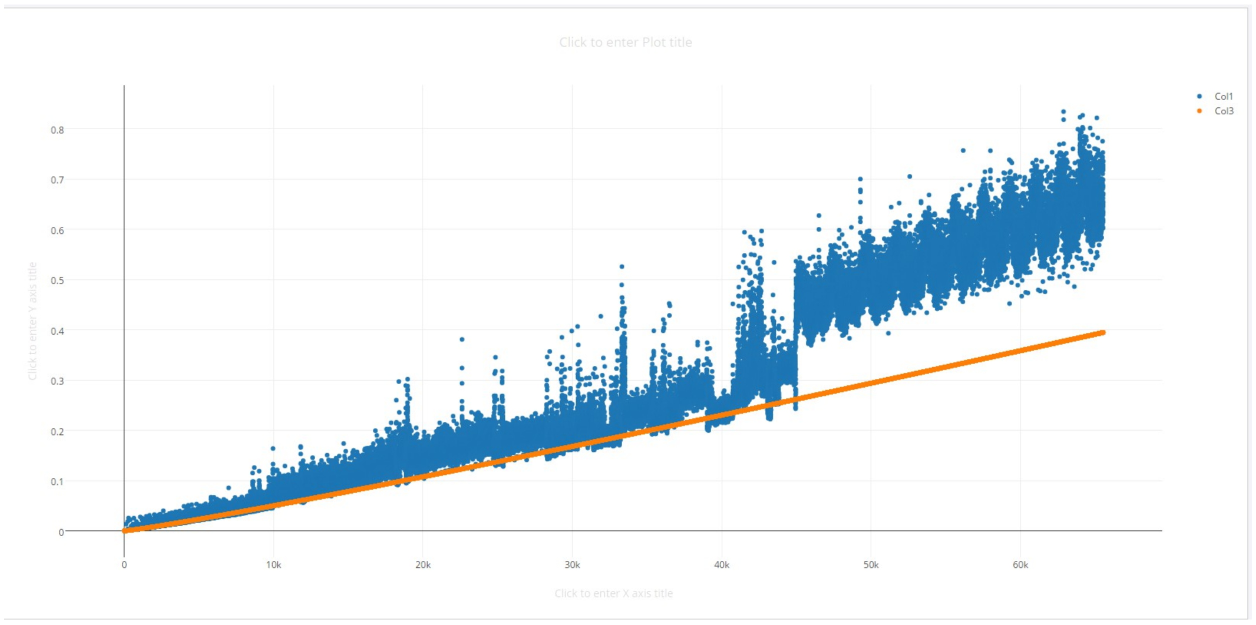
So I realized I have to get much smaller, I did some rough math and decided to go 55 times smaller so next 1/480,000

this was the result:



This was so much closer so I'm gonna try something smaller, 1/800,000

This was the result:



With this you can see that when n0 = ~46k 1/800,000 can lower bound Merge Sort.