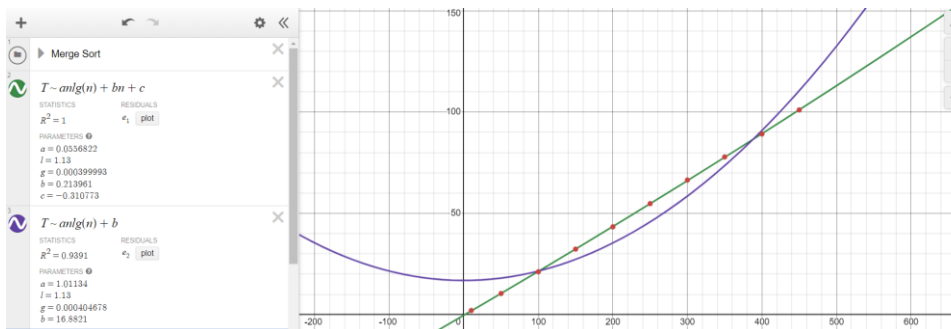
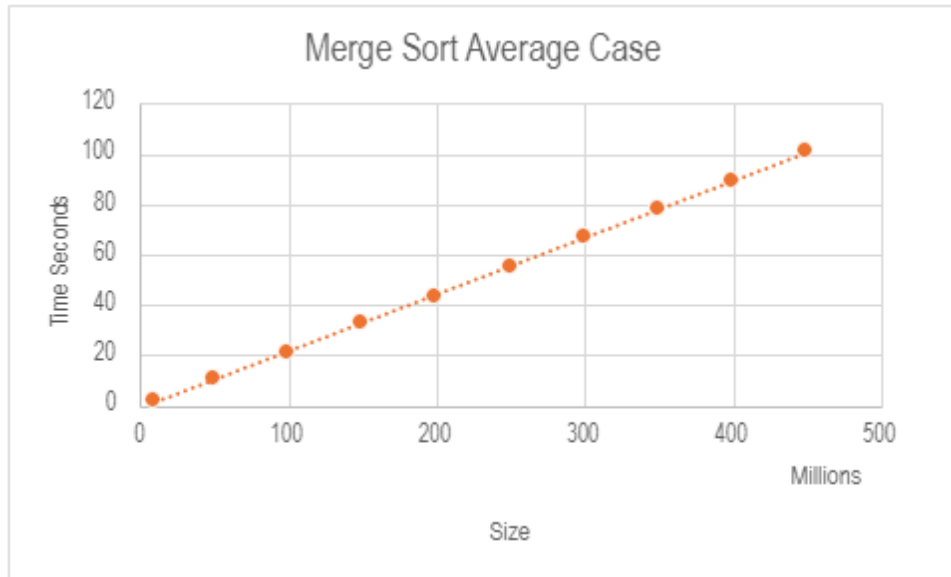


Thomas Pearson

3/15/2024

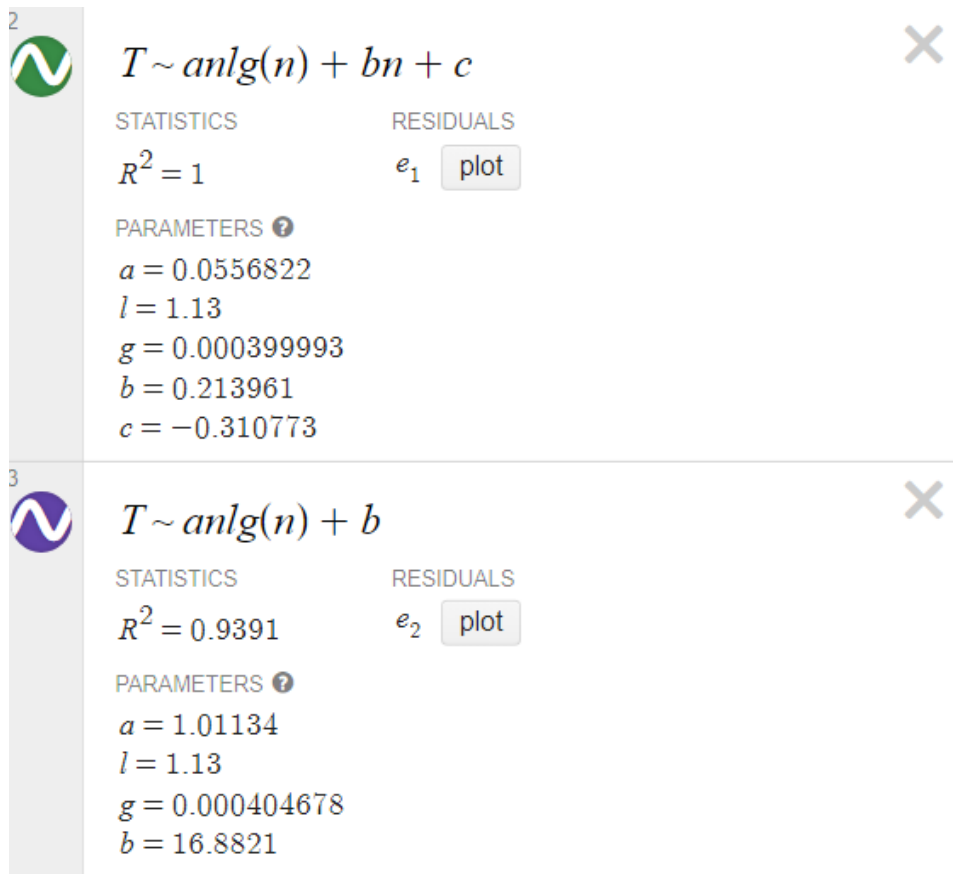
COSC 320

Don Spickler

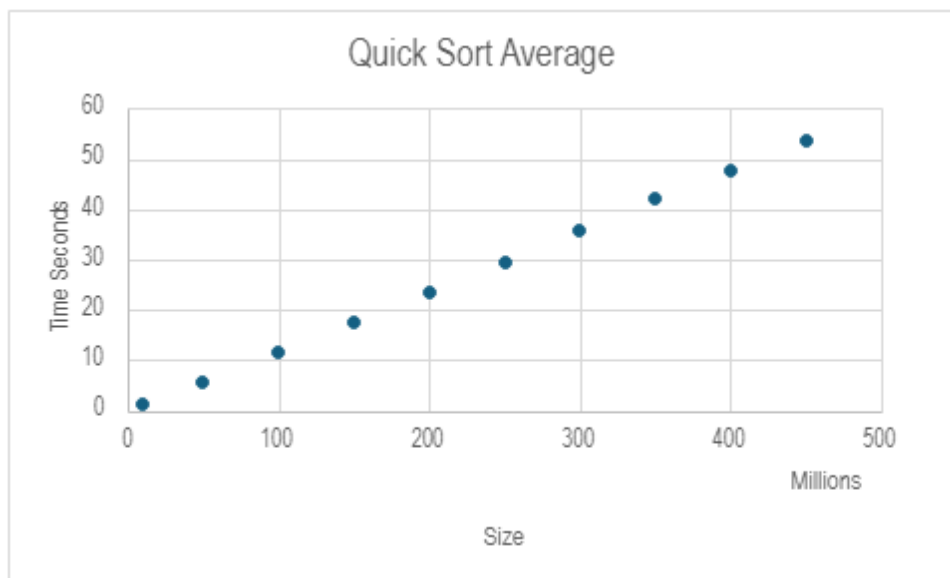


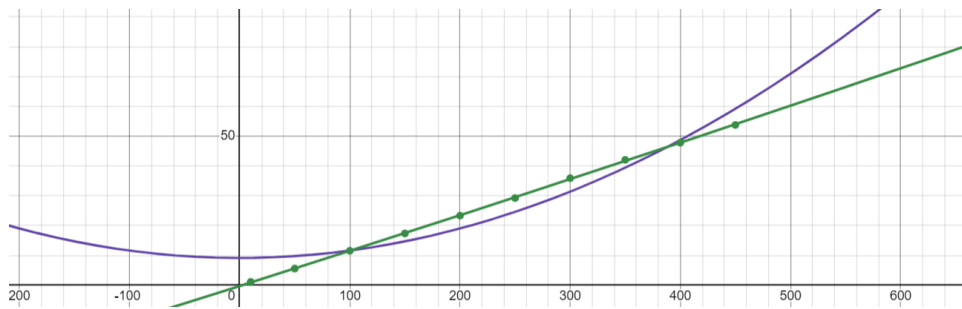
Merge Sort:

Function Fits:



We can clearly see that $f(x)ax\lg(x)+bx+c$ is a perfect fit because the R^2 statistic is 1 which means this function fits the best. We can tell that merge sort on average follows the complexity $n\lg(n) + \theta(n)$.





2

$$T \sim a \ln g(n) + bn + c$$

STATISTICS

RESIDUALS

$R^2 = 0.9999$

e_1

plot

PARAMETERS ?

$a = 0.0164444$
 $l = 1.13$
 $g = 0.000399994$
 $b = 0.117243$
 $c = -0.276849$

3

$$T \sim a \ln g(n) + b$$

STATISTICS

RESIDUALS

$R^2 = 0.936$

e_2

plot

PARAMETERS ?

$a = 0.540276$
 $l = 1.13$
 $g = 0.000404678$
 $b = 9.14425$

4

$$T \sim an^2 + bn + c$$

STATISTICS

RESIDUALS

$R^2 = 0.9999$

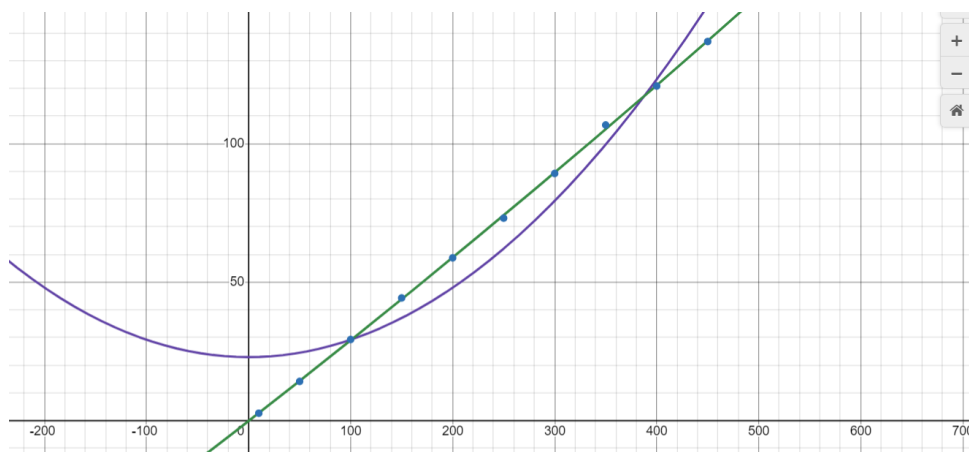
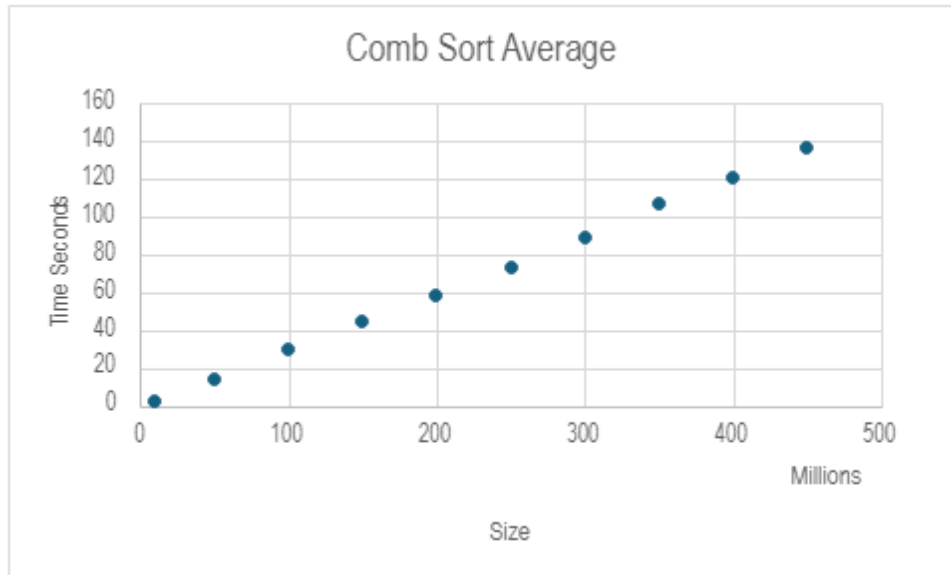
e_3

plot

PARAMETERS

$a = 0.00000743276$
 $b = 0.117243$
 $c = -0.276849$

We can see that the that $f(x)$ and $g(x)$ fit the curve best because they both have an R^2 of 0.9999 which is extremely close to 1 which is an almost perfect fit.



$$T \sim a n l g(n) + b n + c$$

STATISTICS

$$R^2 = 0.9998$$

RESIDUALS

 e_1

plot

PARAMETERS ?

$$a = 0.0875195$$

$$l = 1.13$$

$$g = 0.000399994$$

$$b = 0.287457$$

$$c = -0.193294$$

$$T \sim a n l g(n) + b$$

STATISTICS

$$R^2 = 0.94$$

RESIDUALS

 e_2

plot

PARAMETERS ?

$$a = 1.37131$$

$$l = 1.13$$

$$g = 0.000404675$$

$$b = 22.9053$$

$$T \sim a n^2 + b n + c$$

STATISTICS

$$R^2 = 0.9998$$

RESIDUALS

 e_3

plot

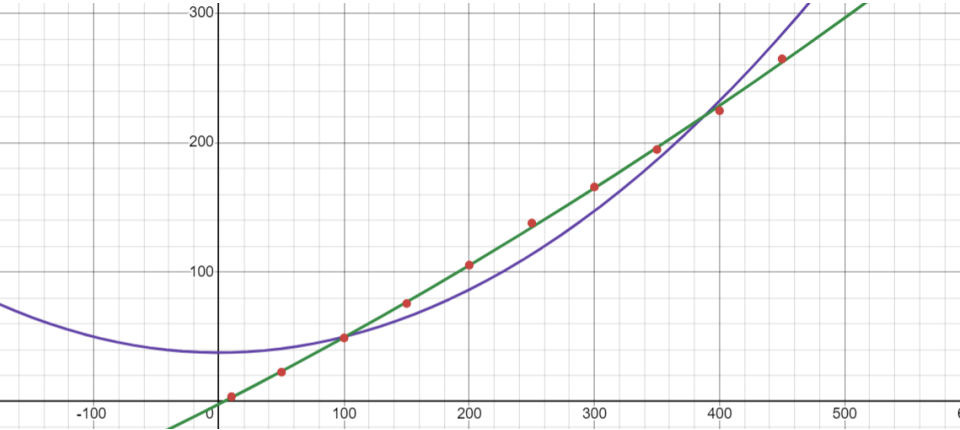
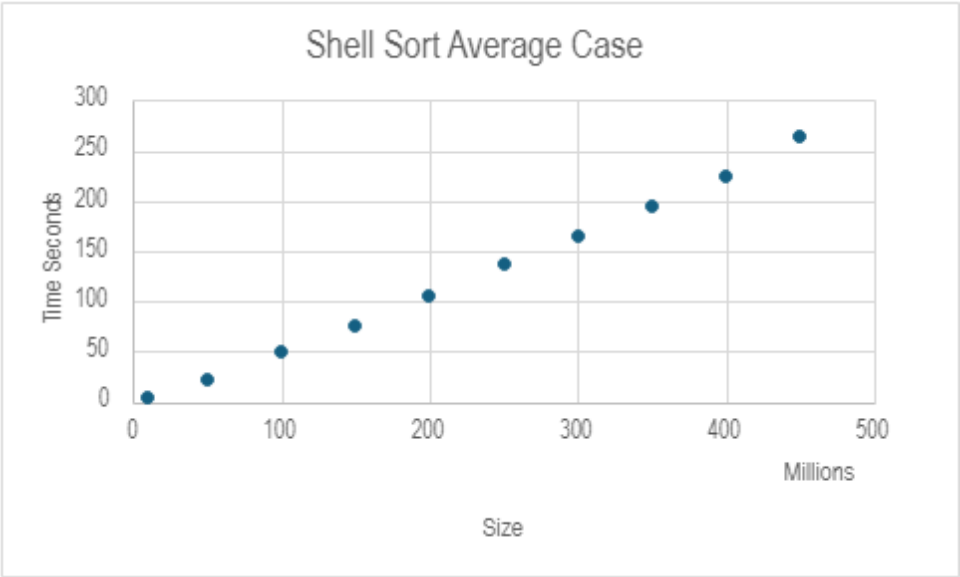
PARAMETERS

$$a = 0.0000395582$$

$$b = 0.287457$$

$$c = -0.193294$$

f(x) and h(x) fit best because the R^2 statistic is both 0.998.



$$T \sim anlg(n) + bn + c$$



STATISTICS

$$R^2 = 0.9994$$

RESIDUALS

 e_1

PARAMETERS ?

$$a = 0.450157$$

$$l = 1.13$$

$$g = 0.000399714$$

$$b = 0.49536$$

$$c = -1.86939$$

$$T \sim anlg(n) + b$$



STATISTICS

$$R^2 = 0.9516$$

RESIDUALS

 e_2

PARAMETERS ?

$$a = 2.38838$$

$$l = 1.13$$

$$g = 0.000450473$$

$$b = 37.9353$$

$$T \sim an^2 + bn + c$$



STATISTICS

$$R^2 = 0.9994$$

RESIDUALS

 e_3

PARAMETERS

$$a = 0.000203325$$

$$b = 0.49536$$

$$c = -1.86939$$

$$T \sim an^{\left(\frac{5}{4}\right)} + bn + c$$

STATISTICS

$$R^2 = 1$$

RESIDUALS

e_6

PARAMETERS

$$a = 0.0133147$$

$$b = 0.209064$$

$$c = -0.245148$$

$$T \sim an^{\left(\frac{5}{4}\right)} + b$$

STATISTICS

$$R^2 = 0.9961$$

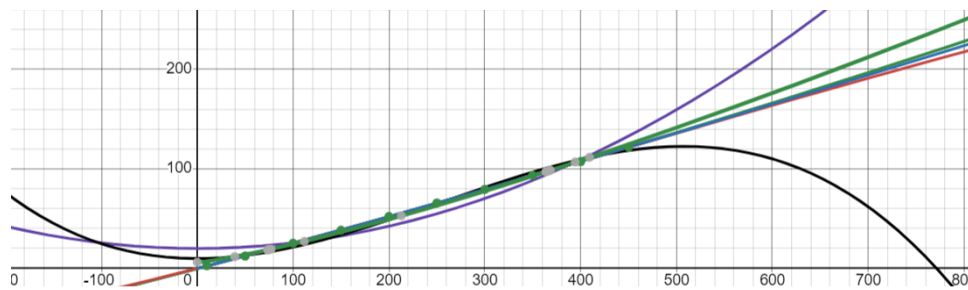
RESIDUALS

e_7

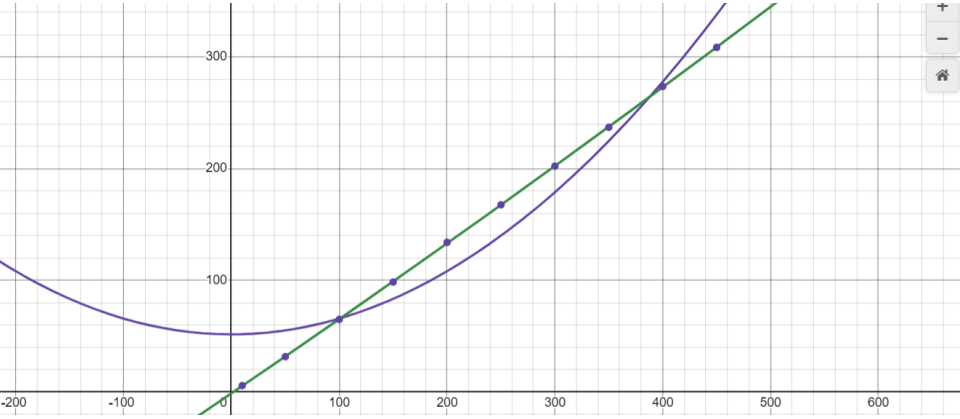
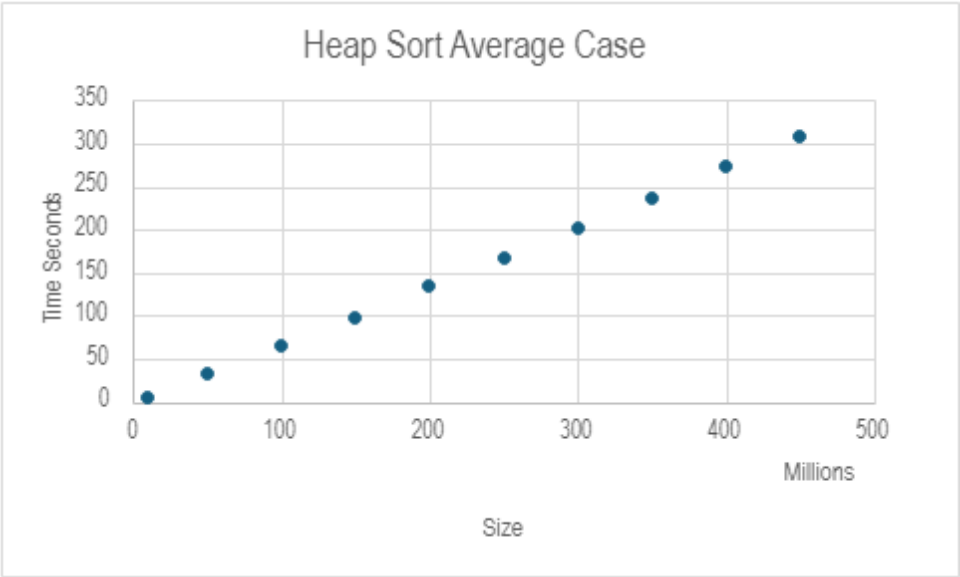
PARAMETERS

$$a = 0.0573033$$

$$b = 5.8213$$



Shell Sort: the best equations here is $f(x) = an^{5/4} + bn + c$ with an R^2 of 1



$$T \sim anlg(n) + bn + c$$

STATISTICS

$$R^2 = 1$$

RESIDUALS

$$e_1$$

PARAMETERS ?

$$a = 0.153626$$

$$l = 1.13$$

$$g = 0.000399992$$

$$b = 0.657869$$

$$c = -1.1451$$

$$T \sim anlg(n) + b$$

STATISTICS

$$R^2 = 0.9385$$

RESIDUALS

$$e_2$$

PARAMETERS ?

$$a = 3.09207$$

$$l = 1.13$$

$$g = 0.000404697$$

$$b = 51.718$$

$$T \sim an^2 + bn + c$$

STATISTICS

$$R^2 = 1$$

RESIDUALS

$$e_3$$

PARAMETERS

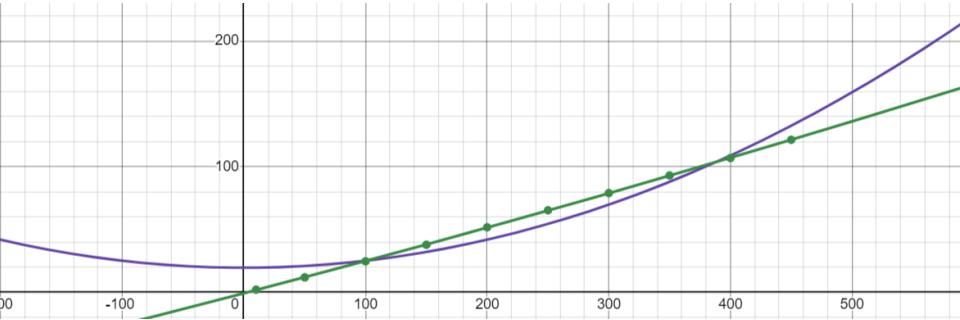
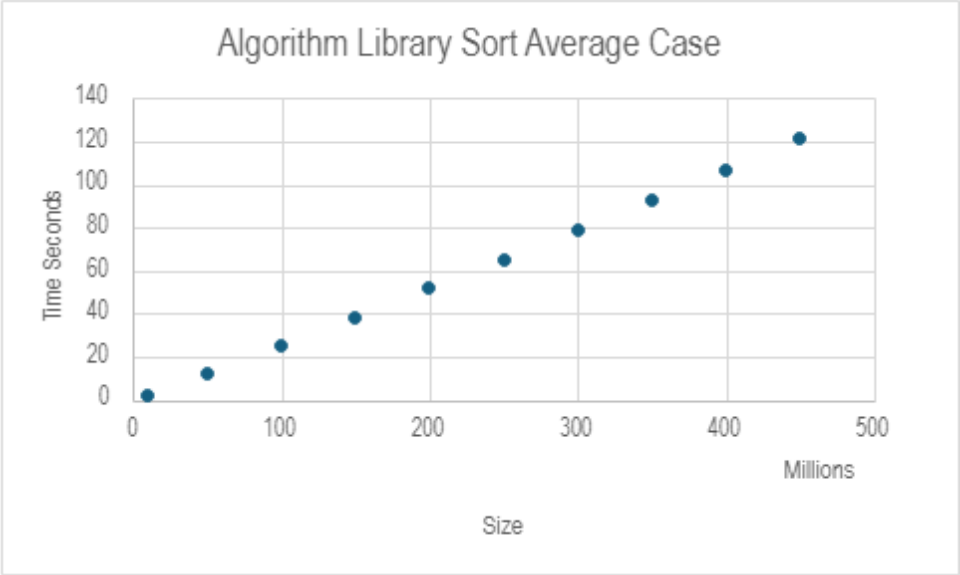
$$a = 0.0000694376$$


$$b = 0.657869$$


$$c = -1.1451$$

The best fits for Heap sort is $f(x)$ and $h(x)$ with R^2 of 1

Given comb, shell, and heap sort all have a similar R^2 stat for the same equations it's safe to say they're quite similar on average.



2


$T \sim a n \lg(n) + b n + c$


STATISTICS


$R^2 = 1$


RESIDUALS

e_1

PARAMETERS ⓘ

$a = 0.0900717$
 $l = 1.13$
 $g = 0.000399994$
 $b = 0.253355$
 $c = -0.690768$

3


$T \sim a n \lg(n) + b$


STATISTICS


$R^2 = 0.9413$


RESIDUALS

e_2

PARAMETERS ⓘ

$a = 1.2214$
 $l = 1.13$
 $g = 0.000404678$
 $b = 19.6676$

4


$T \sim a n^2 + b n + c$


STATISTICS


$R^2 = 1$

RESIDUALS

e_3

PARAMETERS

$a = 0.0000407118$
 $b = 0.253355$
 $c = -0.690768$

$T \sim a n^2 + b n + c$


STATISTICS

$R^2 = 1$

RESIDUALS

e_3

PARAMETERS

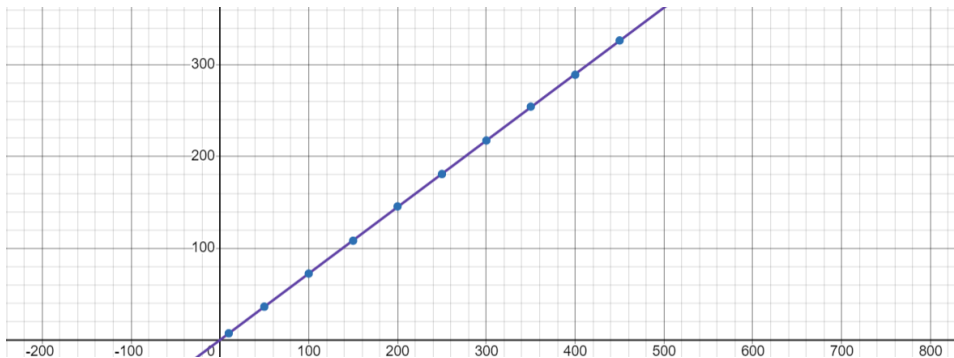
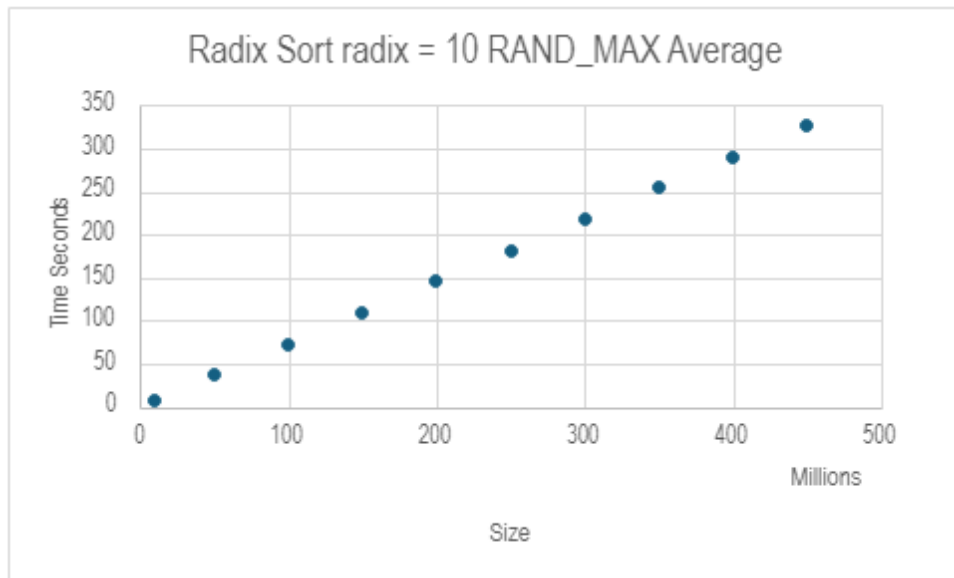
$a = 0.0000407118$

$b = 0.253355$

$c = -0.690768$

F(x) and h(x) are the best R^2 of 2

End of comparison sorts



$T \sim bn + c$

STATISTICS

$r^2 = 1$

$r = 1$

PARAMETERS

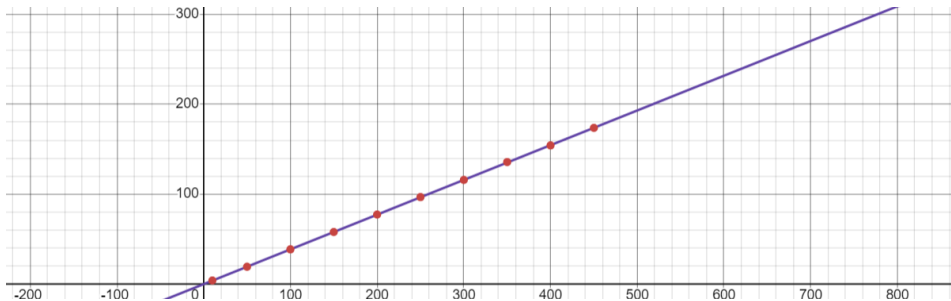
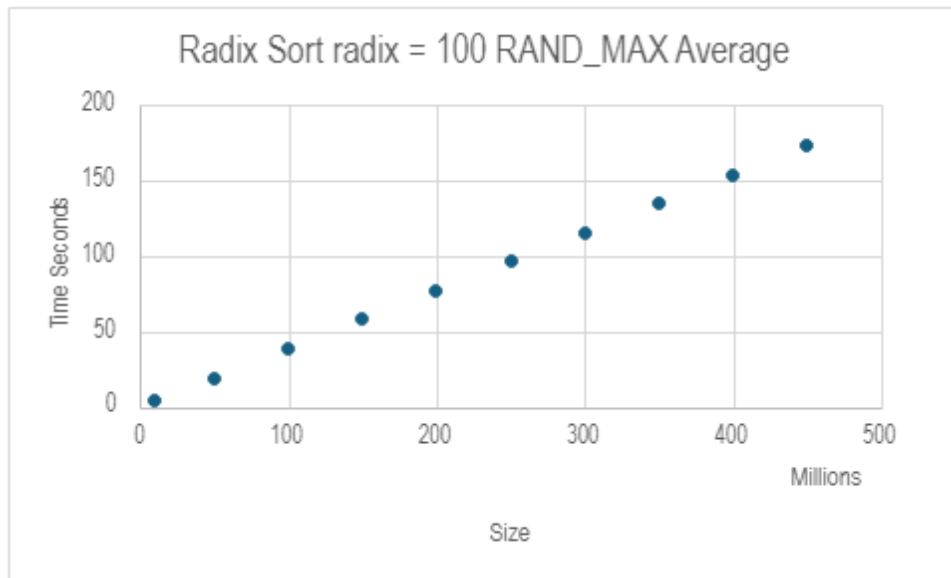
$b = 0.724678$

$c = -0.0401892$

RESIDUALS

e_8

Given that $f(x)$ has an r^2 of 1 that means radix sort runs in linear time



$$T \sim bn + c$$

STATISTICS

$$r^2 = 1$$

$$r = 1$$

PARAMETERS

$$b = 0.386391$$

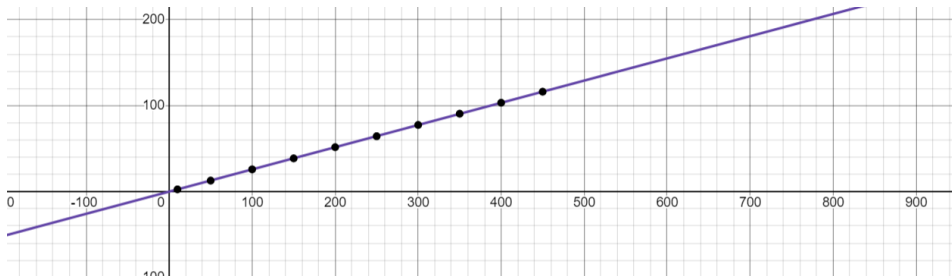
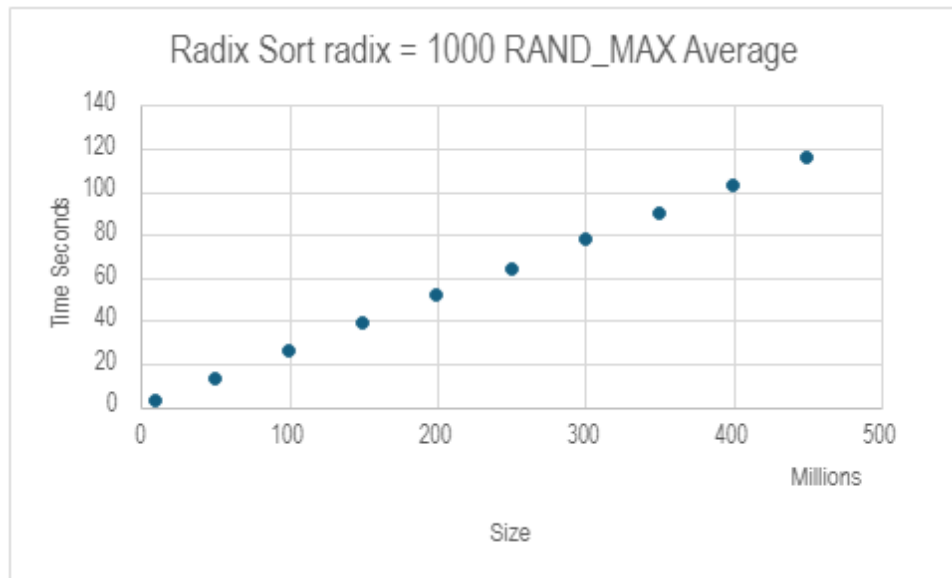
$$c = -0.130865$$

RESIDUALS

e_8

plot

We know that radix sort still runs in linear time because it has an r^2 of 1



$$T \sim bn + c$$

STATISTICS

$$r^2 = 1$$

$$r = 1$$

PARAMETERS

$$b = 0.258402$$

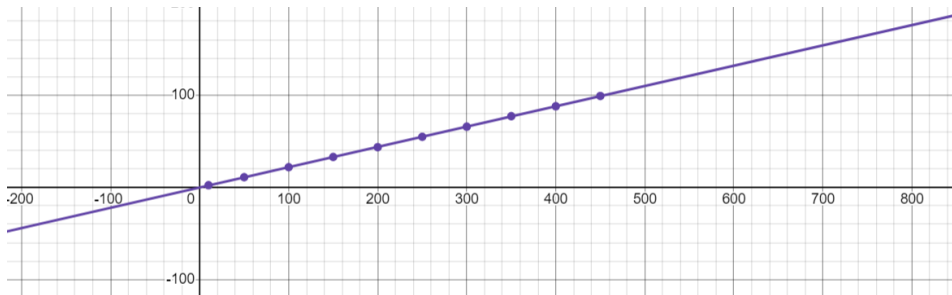
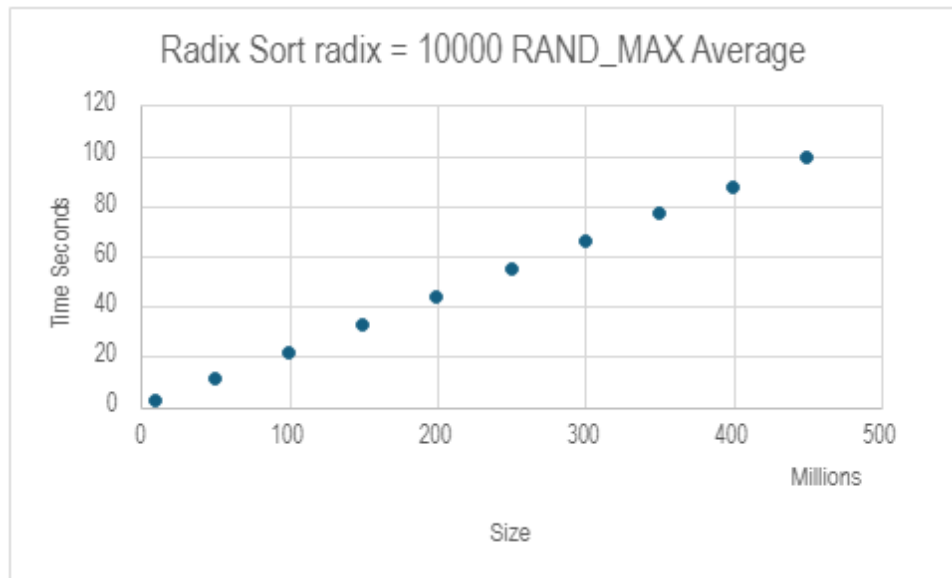
$$c = -0.106405$$

RESIDUALS

$$e_8$$

plot

We know that radix sort is still linear time because $r^2 = 1$



$$T \sim bn + c$$

STATISTICS

$$r^2 = 1$$

$$r = 1$$

PARAMETERS

$$b = 0.220031$$

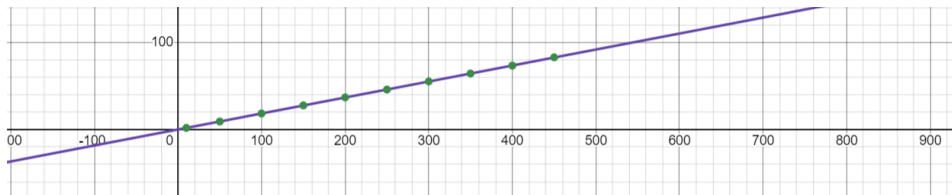
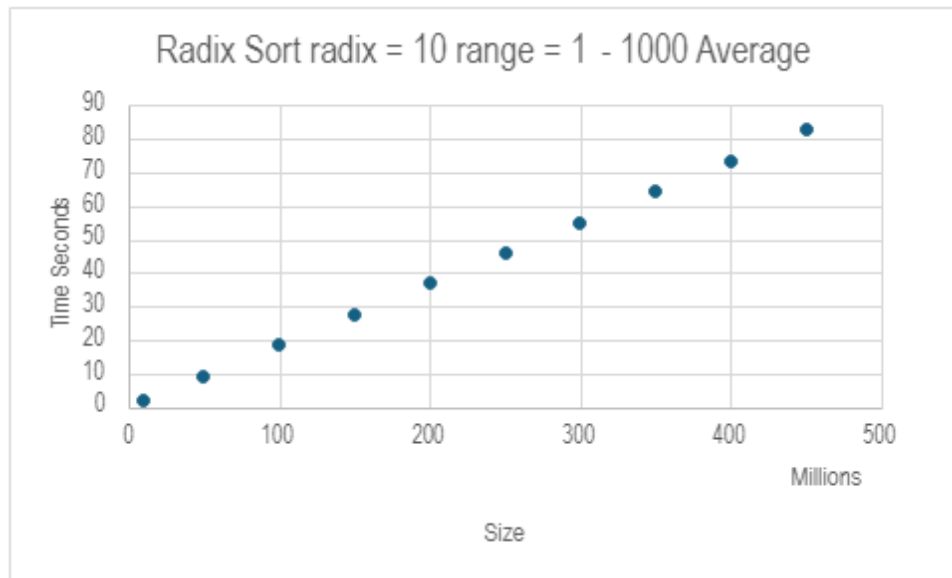
$$c = -0.219483$$

RESIDUALS

e_s

plot

Still linear time, but at this point I will point out that given the nature of how radix sort function, the higher the radix in comparison to input values the faster the sort.



✕

$T \sim bn + c$

STATISTICS

$r^2 = 1$

$r = 1$

PARAMETERS

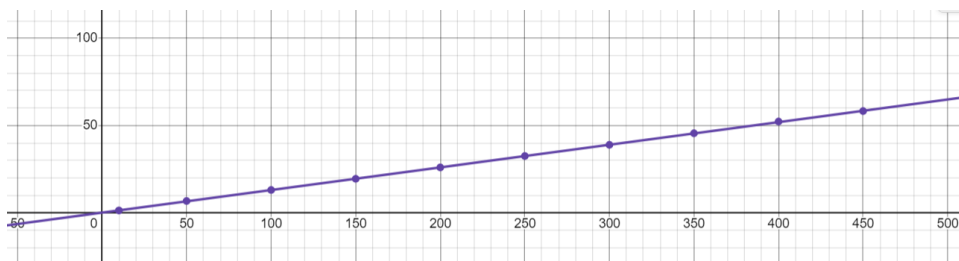
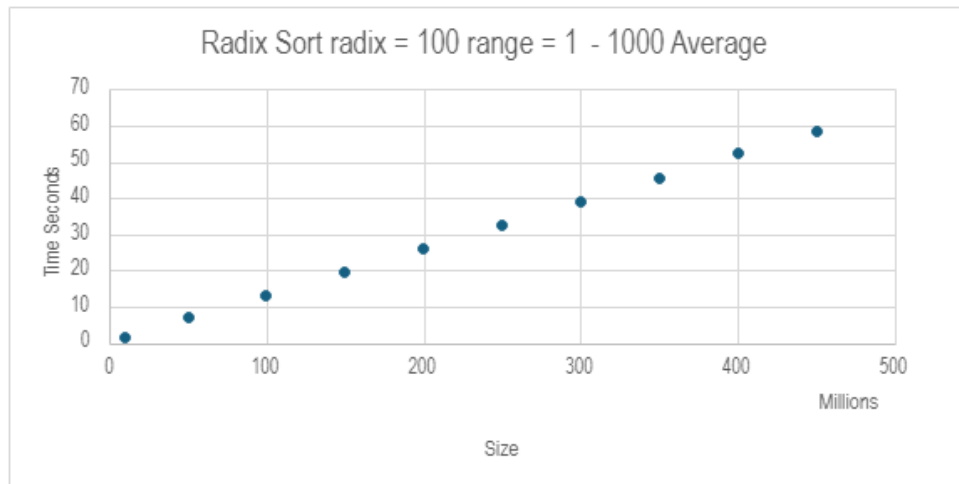
$b = 0.183464$

$c = 0.0498752$

RESIDUALS

e_8

LINEAR TIME this is the fastest one though because the range of possible numbers was significantly reduced.



$$T \sim bn + c$$

STATISTICS

$$r^2 = 0.9999$$

$$r = 1$$

PARAMETERS

$$b = 0.129568$$

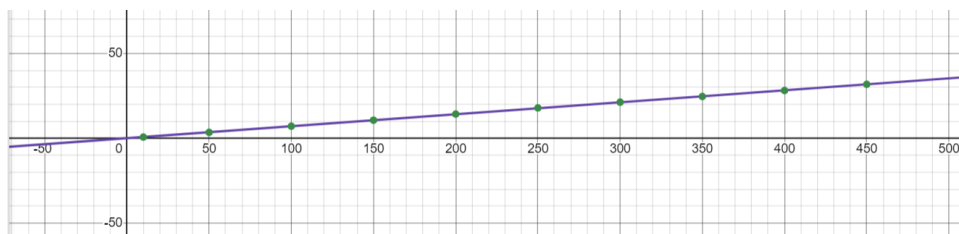
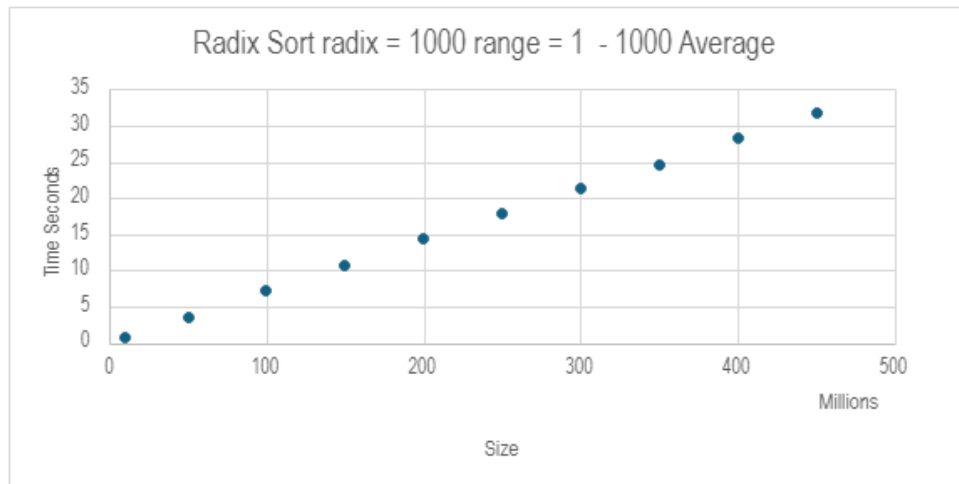
$$c = 0.1917$$

RESIDUALS

e_8

plot

On average radix sort still runs in linear time $r^2=0.9999$



$$T \sim bn + c$$

STATISTICS

$$r^2 = 0.9999$$

$$r = 1$$

PARAMETERS

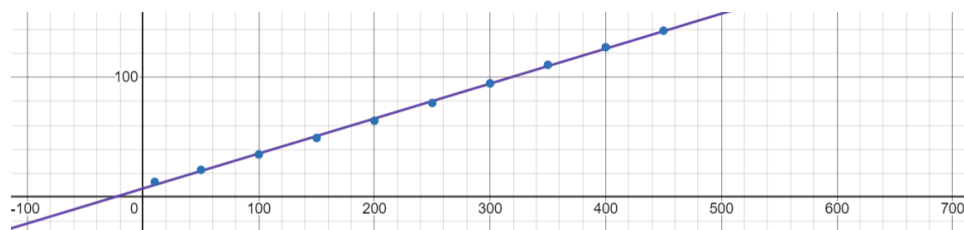
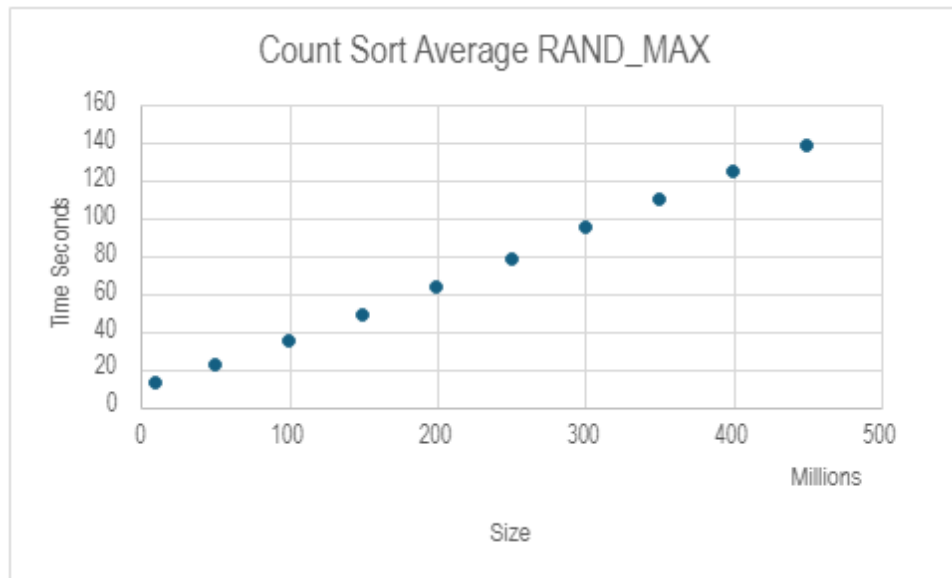
$$b = 0.0704582$$

$$c = 0.109102$$

RESIDUALS

e_s plot

It is still linear time 0.9999 but we also see that radix sort performs the best the closer the radix is to the max possible integer input. We see in the data that if we know the max value then radix sort can beat out every algorithm in terms of integers on average except for count sort when the max value is relatively small like 1 – 1000.



$$T \sim bn + c$$

STATISTICS

$$r^2 = 0.9989$$

$$r = 0.9994$$

PARAMETERS

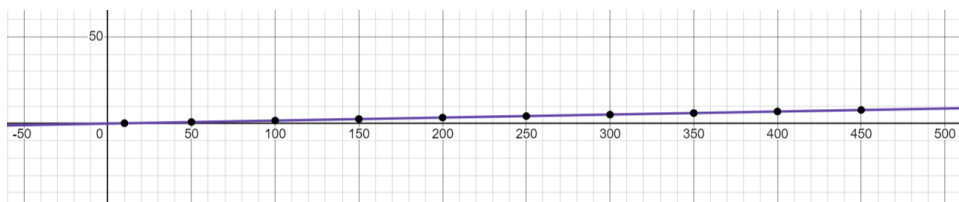
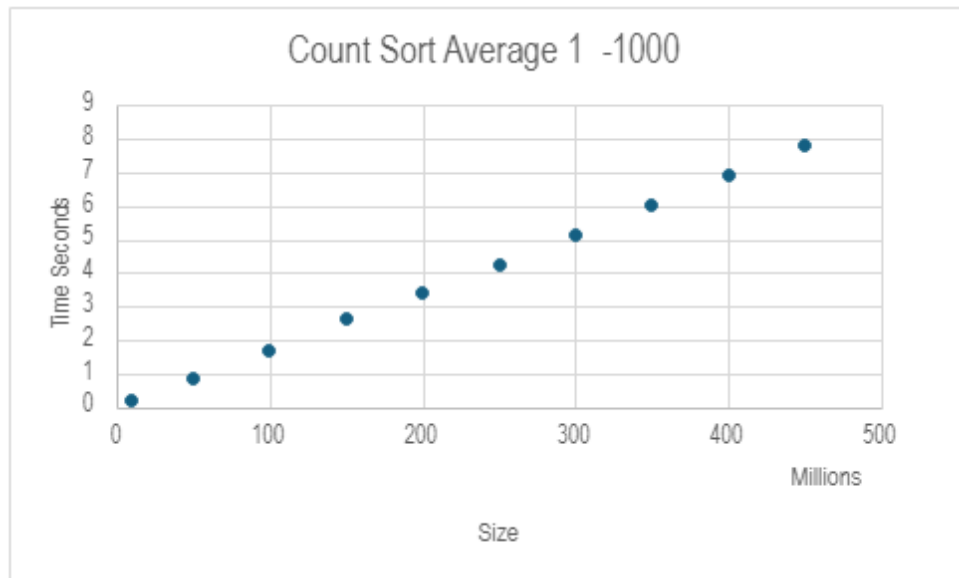
$$b = 0.292644$$

$$c = 7.12999$$

RESIDUALS

e_s

This is count sort is in range rand max and it's pretty close to linear time given $r^2 = 0.9989$



$$T \sim bn + c$$

STATISTICS

$$r^2 = 0.9999$$

$$r = 0.9999$$

PARAMETERS

$$b = 0.0173468$$

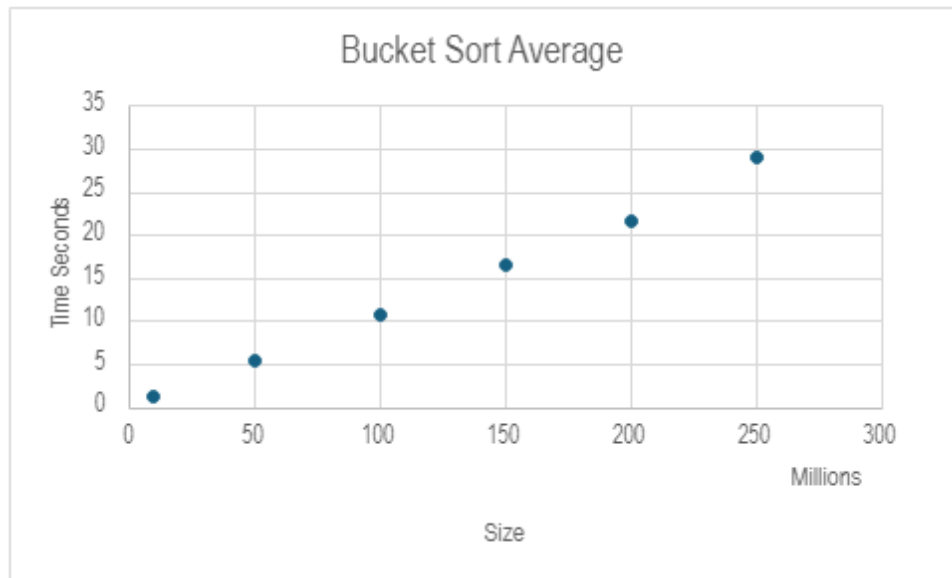
$$c = -0.0336126$$

RESIDUALS

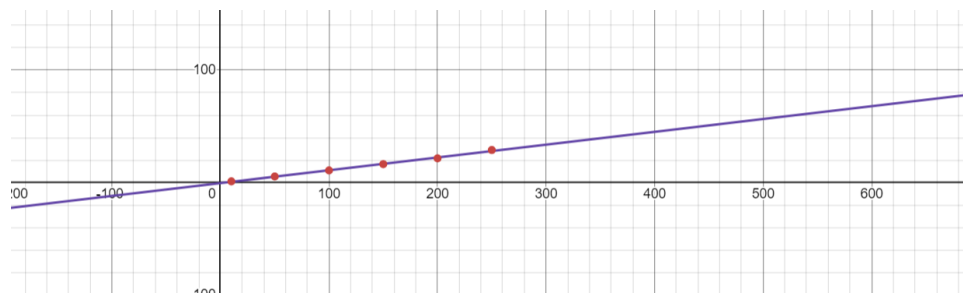
e_s

plot

We can see that given the r^2 of 0.9999 that count sort from range 0 – 1000 is still close to linear time. We can also notice that count sort really shines when the integers being sorted are relatively small.



DUE TO MY HARDWARE, I COULD NOT GET FURTHER THAN 250 MILLION.



$$T \sim bn + c$$

STATISTICS

$$r^2 = 0.9969$$

$$r = 0.9984$$

PARAMETERS

$$b = 0.113983$$

$$c = -0.387037$$

RESIDUALS

e_s

As we can see bucket sort is close to linear time given the r^2 value of 0.9969 and based on the data I was able to gather, this algorithm looks to be the fastest sort we have tested. Beating out quick sort on average.

Summary:

For all the comparison sorts, they almost all perfectly fit with the equation $f(x) = a \lg(x) + bn + c$. The exception being shell sort which still fit this equation nearly perfectly however it fits another equation better with an r^2 of 1 that being $f(x) = ax^{5/4} + bn + c$. On average, quick sort is generally going to be the end all be all non-comparison sorts, and in some cases will outperform count and radix. That is if the data to be sorted is not already sorted of course.

In terms of the non-comparison sorts we noticed that they all ran in linear time. Meaning they all fit the equation $f(x) = bx + c$ which represents linear time. However, counting sort is not directly better than radix sort and vice versa. They both had their moments where they shined through all the other sorts we looked at. Bucket Sort of course is just being used for the specific case of doubles between 0 and 1. Counting sort was shown to be the top contender for smaller ranges of numbers like 1 – 1000 whereas radix sort was extremely efficient for higher ranges given the radix was also high or directly matched the maximum number in the range. I am not sure if this was a flaw with my code but if the radix was higher than the range i.e. radix = 10000 and the range was between 1 and 1000 the program would kill itself.

We notice that when the max value for data to be stored is in small ranges such as 1 – 1000 count sort takes the cake easily in terms of sorting integers. If the data is big but the radix for our radix sort is close or matches the max value, there's no contest for integers. Of course, on average with no known conditions if we know that we're not getting a sorted array quick sort will still outperform every sorting algorithm that utilizes comparisons, and it also has potential to beat out count and radix sort depending on the conditions. Bucket sort is of course just the end all be all for doubles.