COSC 320 - Dr. Spickler

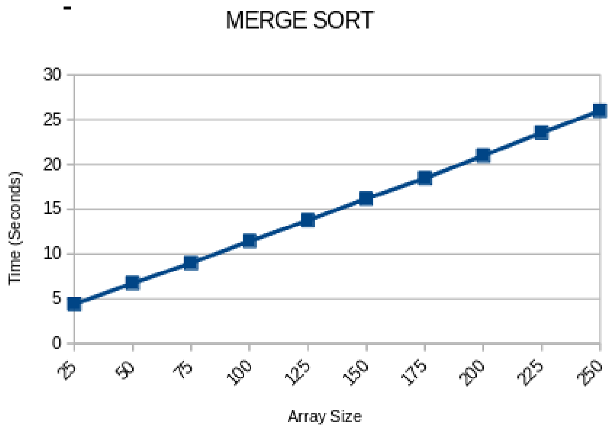
Project 01 - 3/13/24

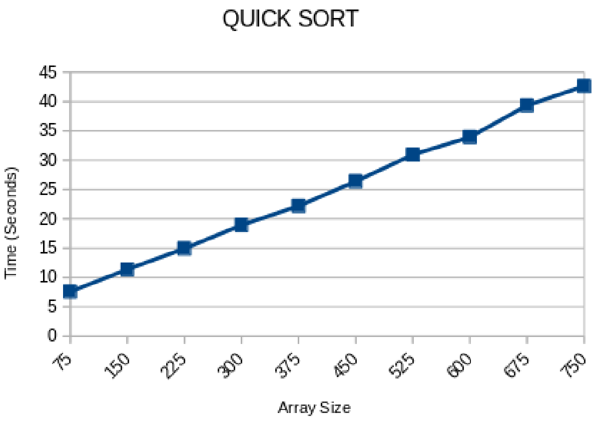
Logan Kelsch

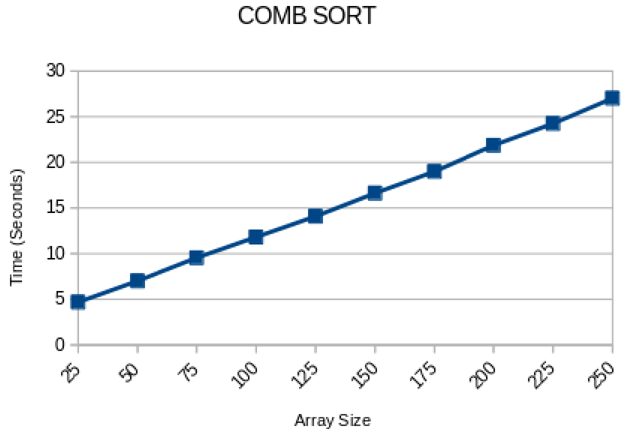
In this project, we analyzed multiple comparison sorts, as well as non-comparison sorts. The comparison sorts are Merge Sort, Quick Sort, Comb Sort, Shell Sort, Heap Sort, and the Algorithm Library Sort. The non-comparison sorts are Radix Sort, Count Sort, and Bucket Sort.

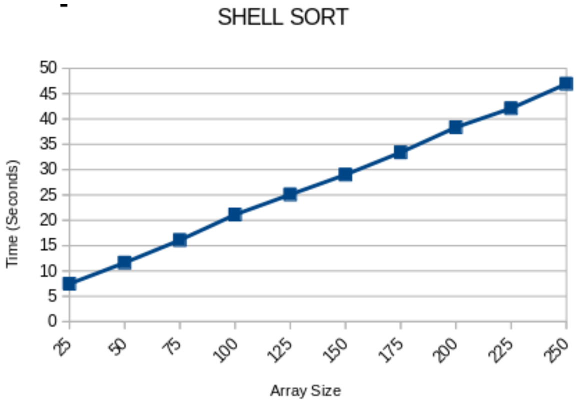
COMPARISON SORTS ANALYSIS

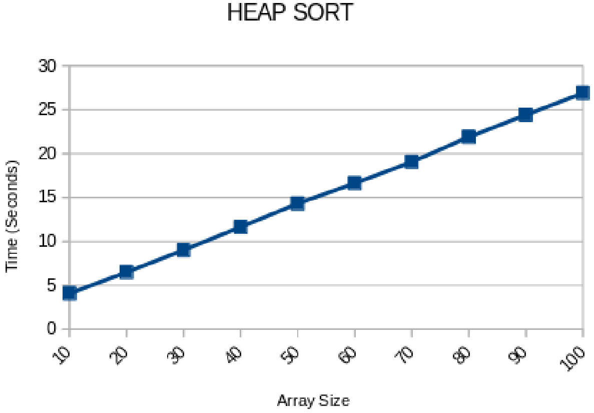
After gathering 10 data points of sort time against various array sizes for each sort, a few differences and similarities were revealed by the statistical data. Each test was run against array sets from the range of 10 million to 750 million, and ranging from around 4 seconds to about 50 seconds. The following page consists of all sets of timing tests against each 6 comparison sorts included in this project.

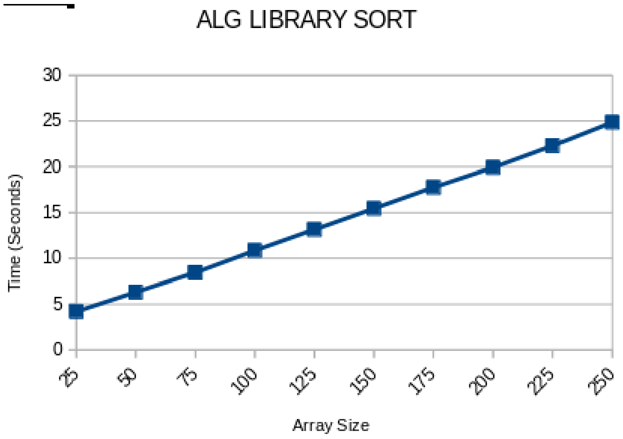

MERGE SORT
Time (Seconds)
0
5
10
15
20
25
30
25
50
75
100
125
150
175
200
225
250
Array Size


QUICK SORT
Time (Seconds)
0
5
10
15
20
25
30
35
40
45
75
150
225
300
375
450
525
600
675
750
Array Size


COMB SORT
Time (Seconds)
0
5
10
15
20
25
30
25
50
75
100
125
150
175
200
225
250
Array Size

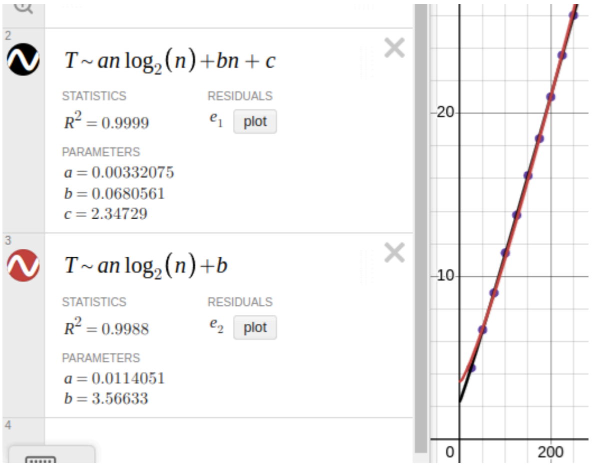

SHELL SORT
Time (Seconds)
0
5
10
15
20
25
30
35
40
45
50
25
50
75
100
125
150
175
200
225
250
Array Size


HEAP SORT
Time (Seconds)
0
5
10
15
20
25
30
10
20
30
40
50
60
70
80
90
100
Array Size


ALG LIBRARY SORT
Time (Seconds)
0
5
10
15
20
25
30
25
50
75
100
125
150
175
200
225
250
Array Size

The data gathered and presented on page two present a noticeably faster and noticeably slower algorithm, while revealing near equality of speed-matter in the rest of the sorts. The noticeably faster comparison algorithm was Quick Sort, with the capacity of sorting 525 million data points in just over 30 seconds. Heap sort was the slowest comparison algorithm, with a 100 million data point sort time slower than most 150 million data point sort times of the other algorithms, at around 27 seconds. The curve fitting function values are presented on the remainder of this page as well as the next.

Merge Sort:



2 $T \sim an\log_2(n)+bn+c$

STATISTICS $R^2 = 0.9999$ RESIDUALS e_1 plot

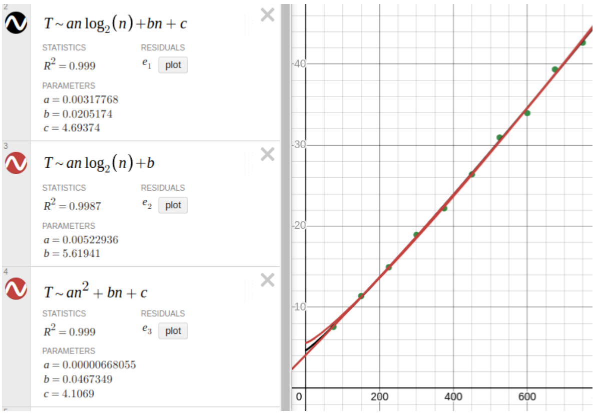
PARAMETERS
$a = 0.00332075$
$b = 0.0680561$
$c = 2.34729$

3 $T \sim an\log_2(n)+b$

STATISTICS $R^2 = 0.9988$ RESIDUALS e_2 plot

PARAMETERS
$a = 0.0114051$
$b = 3.56633$

Quick Sort:



2 $T \sim an\log_2(n)+bn+c$

STATISTICS $R^2 = 0.999$ RESIDUALS e_1 plot

PARAMETERS
$a = 0.00317768$
$b = 0.0205174$
$c = 4.69374$

3 $T \sim an\log_2(n)+b$

STATISTICS $R^2 = 0.9987$ RESIDUALS e_2 plot

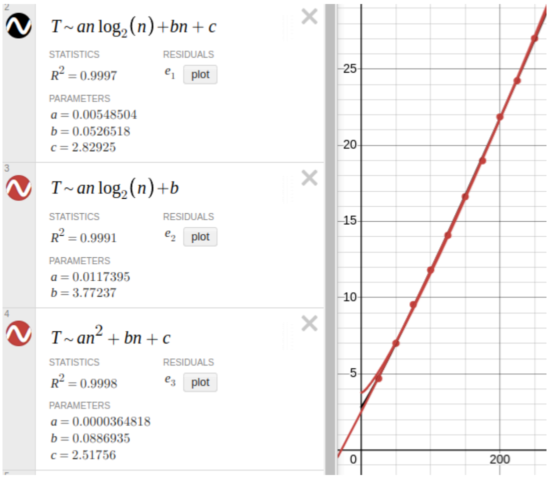
PARAMETERS
$a = 0.00522936$
$b = 5.61941$

4 $T \sim an^2 + bn + c$

STATISTICS $R^2 = 0.999$ RESIDUALS e_3 plot

PARAMETERS
$a = 0.00000668055$
$b = 0.0467349$
$c = 4.1069$

Comb Sort:



2 $T \sim an\log_2(n)+bn+c$

STATISTICS $R^2 = 0.9997$ RESIDUALS e_1 plot

PARAMETERS
$a = 0.00548504$
$b = 0.0526518$
$c = 2.82925$

3 $T \sim an\log_2(n)+b$

STATISTICS $R^2 = 0.9991$ RESIDUALS e_2 plot

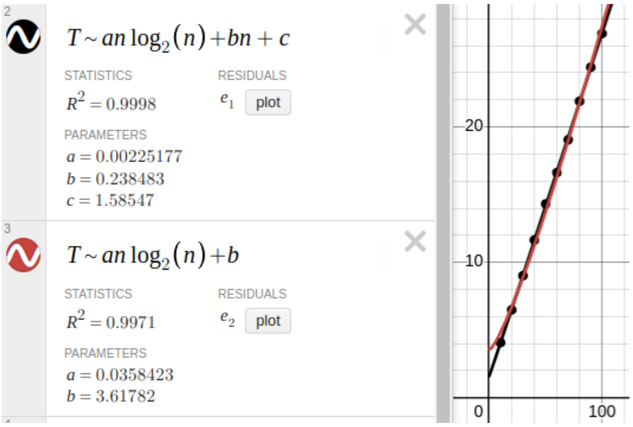
PARAMETERS
$a = 0.0117395$
$b = 3.77237$

4 $T \sim an^2 + bn + c$

STATISTICS $R^2 = 0.9998$ RESIDUALS e_3 plot

PARAMETERS
$a = 0.0000364818$
$b = 0.0886935$
$c = 2.51756$

Heap Sort:



2 $T \sim an\log_2(n)+bn+c$

STATISTICS $R^2 = 0.9998$ RESIDUALS e_1 plot

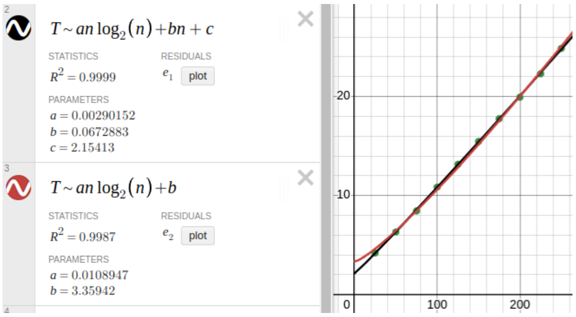
PARAMETERS
$a = 0.00225177$
$b = 0.238483$
$c = 1.58547$

3 $T \sim an\log_2(n)+b$

STATISTICS $R^2 = 0.9971$ RESIDUALS e_2 plot

PARAMETERS
$a = 0.0358423$
$b = 3.61782$

Algorithm Library Sort:



$T \sim an\log_2(n) + bn + c$

STATISTICS $R^2 = 0.9999$

RESIDUALS e_1 plot

PARAMETERS

$a = 0.00290152$

$b = 0.0672883$

$c = 2.15413$

$T \sim an\log_2(n) + b$

STATISTICS $R^2 = 0.9987$

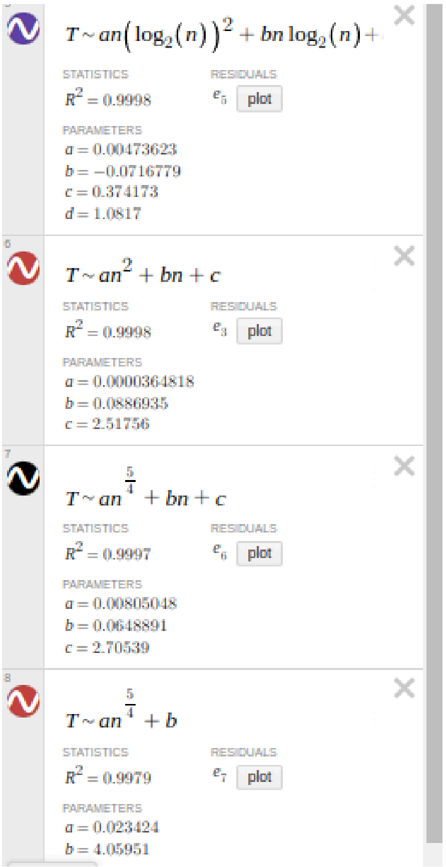
RESIDUALS e_2 plot

PARAMETERS

$a = 0.0108947$

$b = 3.35942$

Shell Sort:



$T \sim an\left(\log_2(n)\right)^2 + bn\log_2(n) +$

STATISTICS $R^2 = 0.9998$

RESIDUALS e_5 plot

PARAMETERS

$a = 0.00473623$

$b = -0.0716779$

$c = 0.374173$

$d = 1.0817$

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

STATISTICS $R^2 = 0.9998$

RESIDUALS e_3 plot

PARAMETERS

$a = 0.0000364818$

$b = 0.0886935$

$c = 2.51756$

$T \sim an^{\frac{5}{4}} + bn + c$

STATISTICS $R^2 = 0.9997$

RESIDUALS e_6 plot

PARAMETERS

$a = 0.00805048$

$b = 0.0648891$

$c = 2.70539$

$T \sim an^{\frac{5}{4}} + b$

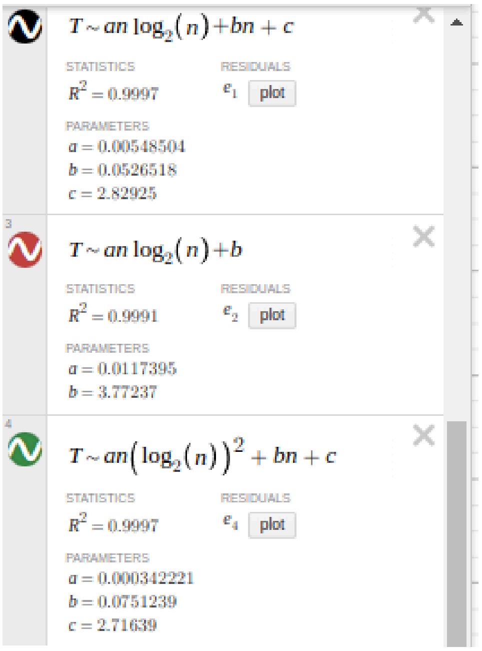
STATISTICS $R^2 = 0.9979$

RESIDUALS e_7 plot

PARAMETERS

$a = 0.023424$

$b = 4.05951$



$T \sim an\log_2(n) + bn + c$

STATISTICS $R^2 = 0.9997$

RESIDUALS e_1 plot

PARAMETERS

$a = 0.00548504$

$b = 0.0526518$

$c = 2.82925$

$T \sim an\log_2(n) + b$

STATISTICS $R^2 = 0.9991$

RESIDUALS e_2 plot

PARAMETERS

$a = 0.0117395$

$b = 3.77237$

$T \sim an\left(\log_2(n)\right)^2 + bn + c$

STATISTICS $R^2 = 0.9997$

RESIDUALS e_4 plot

PARAMETERS

$a = 0.000342221$

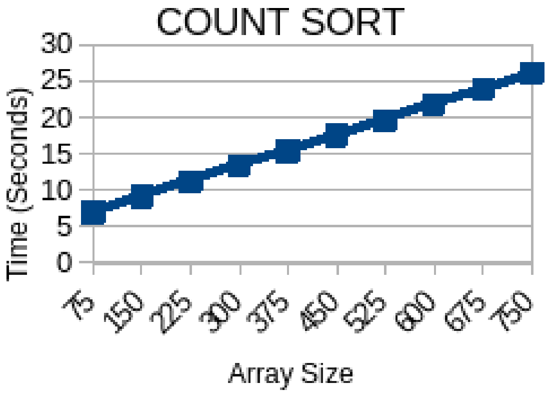
$b = 0.0751239$

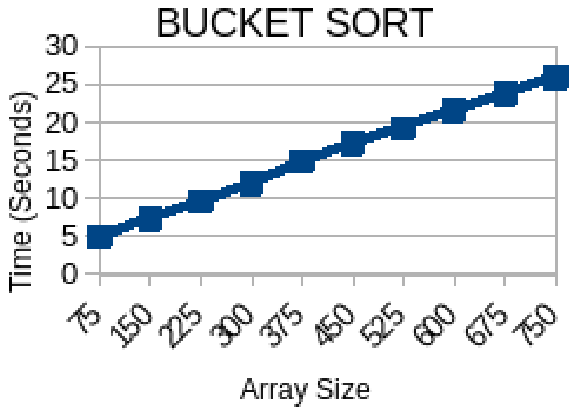
$c = 2.71639$

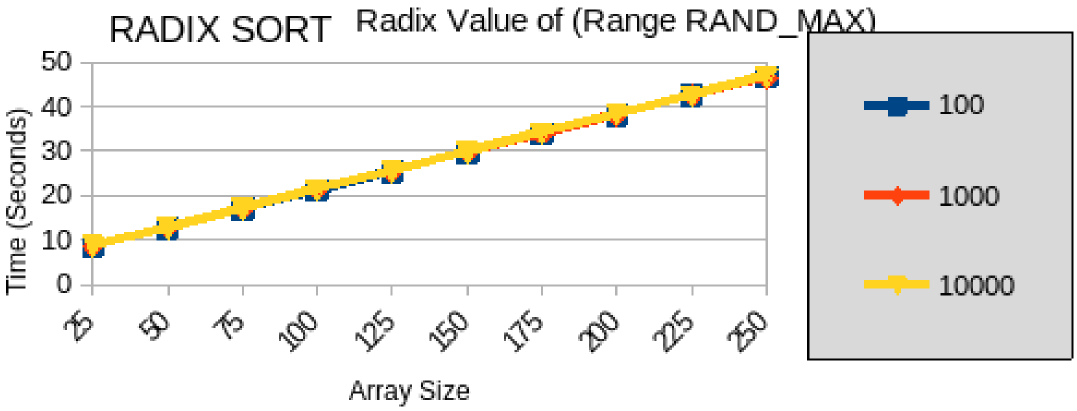
The curve fitting images can be summed up by the closest correlating function, the leading coefficient value, and the R^2 value associated with the curve of best fit function. Merge Sort, an(lg(n))+bn+c was the best function to fit to the collected data, with a leading coefficient value of 0.0032 and an R^2 value of 0.9998. Quick Sort, an(lg(n))+bn+c was the best function to fit to the collected data, with a leading coefficient value of 0.0032 and an R^2 value of 0.999. Comb Sort, an^2+bn+c was the best function to fit to the collected data, with a leading coefficient value of 0.000036 and an R^2 value of 0.9998. Heap Sort, an(lg(n))+bn+c was the best function to fit to the collected data, with a leading coefficient value of 0.0026 and an R^2 value of 0.9998. Algorithm Library Sort, an(lg(n))+bn+c was the best function to fit to the collected data, with a leading coefficient value of 0.0029 and an R^2 value of 0.9999. Shell Sort, an(lg(n))^2+bn(lg(n))+c was the best function to fit to the collected data, with a leading coefficient value of 0.0047 and an R^2 value of 0.9998.

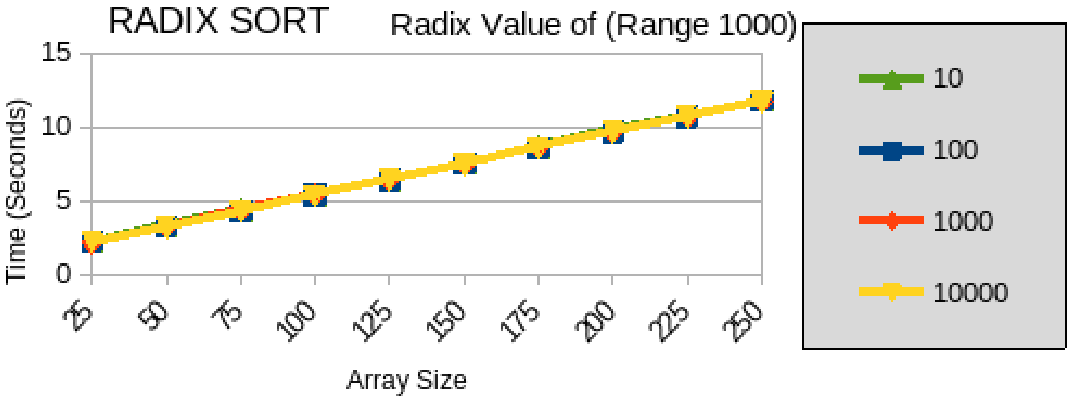
NON-COMPARISON SORT ANALYSIS

After gathering 10 data points of sort time against Count Sort and Bucket Sort, as well as 10 data points for 7 variations of Radix Sort, a strong contrast against comparison sorts was revealed. Among these faster, non-comparison sorts, differences in speed also became statistically apparent. On the following page are charts depicting Count and Bucket Sort timings, as well as timings for all tested variants of Radix Sort.


COUNT SORT
Time (Seconds)
30
25
20
15
10
5
0
75
150
225
300
375
450
525
600
675
750
Array Size

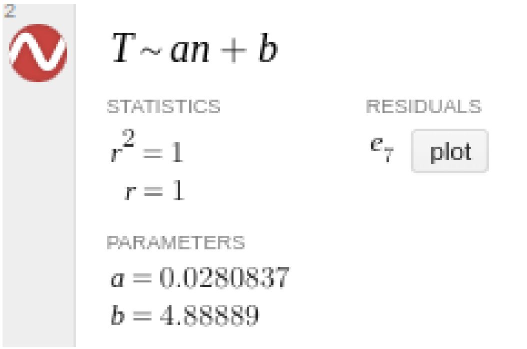

BUCKET SORT
Time (Seconds)
30
25
20
15
10
5
0
75
150
225
300
375
450
525
600
675
750
Array Size


RADIX SORT
Radix Value of (Range RAND_MAX)
Time (Seconds)
50
40
30
20
10
0
25
50
75
100
125
150
175
200
225
250
Array Size
100
1000
10000


RADIX SORT
Radix Value of (Range 1000)
Time (Seconds)
15
10
5
0
25
50
75
100
125
150
175
200
225
250
Array Size
10
100
1000
10000

Firstly and most noticeably, any variance in the Radix Values are almost entirely negligible. Among these three sorts, Buckets Sort appears the fastest, followed closely by near statistical equivalence of Count and Radix Sort timing, with all of their 225 million data point sorts sitting just around 10 seconds! There is also a fairly insignificant difference in sort time between sets of data points up to a value of 1000 against a random maximum value, likely much much higher than 1000. The curve fitting function values are presented on the remainder of this page.

Count Sort:



$T \sim an + b$

STATISTICS

$r^2 = 1$

$r = 1$

RESIDUALS

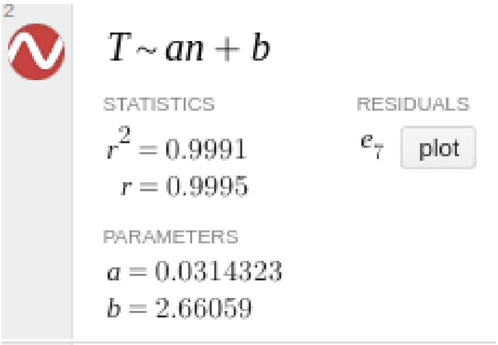
e_7

PARAMETERS

$a = 0.0280837$

$b = 4.88889$

Bucket Sort:



$T \sim an + b$

STATISTICS

$r^2 = 0.9991$

$r = 0.9995$

RESIDUALS

e_7

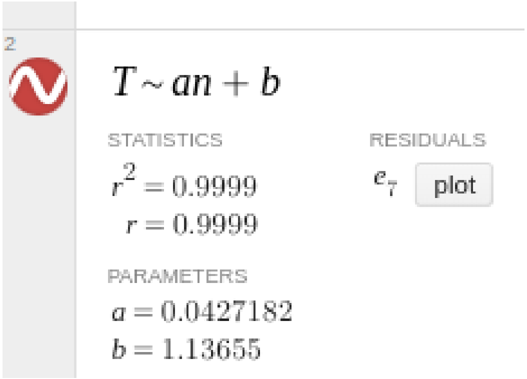
PARAMETERS

$a = 0.0314323$

$b = 2.66059$

Radix Sort

1000 value limit, average:



$T \sim an + b$

STATISTICS

$r^2 = 0.9999$

$r = 0.9999$

RESIDUALS

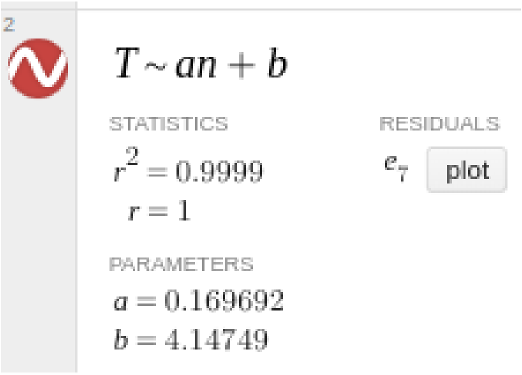
e_7

PARAMETERS

$a = 0.0427182$

$b = 1.13655$

RAND_MAX value limit, average:



$T \sim an + b$

STATISTICS

$r^2 = 0.9999$

$r = 1$

RESIDUALS

e_7

PARAMETERS

$a = 0.169692$

$b = 4.14749$

The curve fitting images can be summed up by the closest correlating function, the leading coefficient value, and the R^2 value associated with the curve of best fit function. Count Sort, an+b was the best function to fit to the collected data, with a leading coefficient value of 0.028 and an R^2 value of 1.0000. Bucket Sort, an+b was the best function to fit to the collected data, with a leading coefficient value of 0.031 and an R^2 value of 0.9991. Radix Sort with a maximum integer value of 1000, an+b was the best function to fit to the collected data, with a leading coefficient value of 0.043 and an R^2 value of 0.9999. Radix Sort with a maximum integer value of RAND_MAX, an+b was the best function to fit to the collected data, with a leading coefficient value of 0.1697 and an R^2 value of 0.9999.

CONCLUSION

Across the sorts discussed in this project, their complexity, and time fit functions, we discover better and worse algorithms for different situations. In an event where data points must be compared to sort, Quick Sort is the most effective for large data sets, while Heap Sort is the least effective for large data sets. The other comparison sorts are nearly equally effective. Given a situation where non-comparison sorting algorithms can be used, any sorting algorithm will provide massively better. Among these non-comparison sorts, Bucket sort performs slightly faster than Radix and Count Sort, while Radix and Count are nearly the same speed.

Beyond the time data collection and speed-matter differences brought to light, space-matter is also important in choosing which sorting algorithm to use. So, the large sum of data collected from this project only makes up one half of what should be considered while creating data storing, fetching, and analysis programs.