Samuel Dickerson

Dr. Spickler

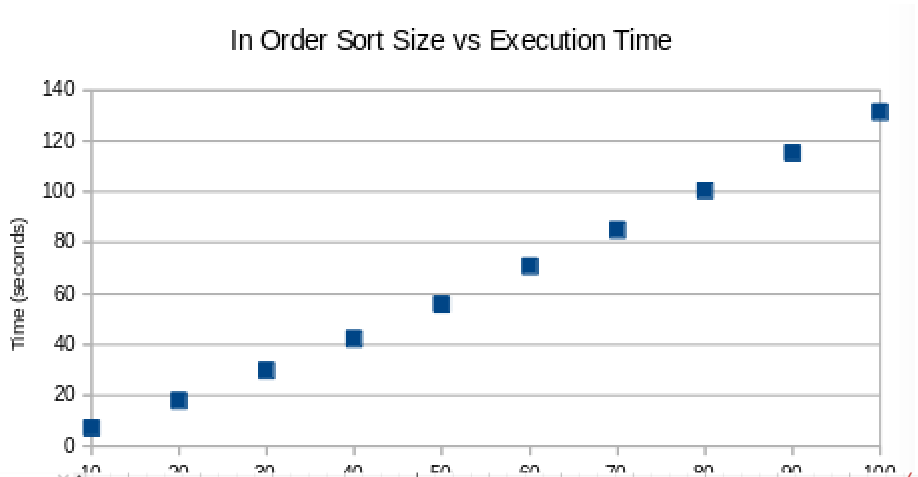
15 March 2024

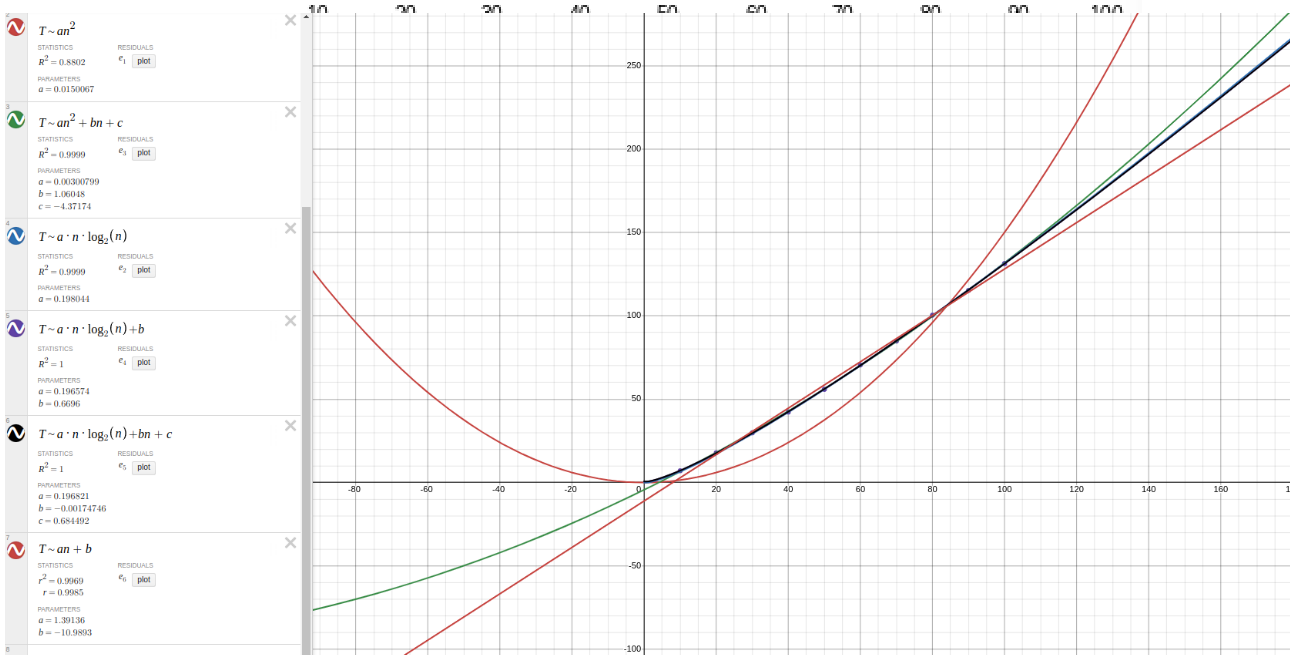
COSC 320

Tree Sorting Size vs Execution and Complexity Analysis

In Order Sort:

The In Order Sort curve appears to be n*lg(n). The worst fit for the data is an^2, with an R^2 statistic of 0.8802. The best fits for the data were an^2 + bn + c with an R^2 of 0.9999, an*lg(n) with an R^2 of 0.9999, an*lg(n) + b with an R^2 of 1, and an*lg(n) + b*n + c. The linear fit an + b was the next best for the data with an R^2 of 0.9969. The data has the strongest correlation to the complexities O(nlg(n)), n(lg(n)) + O(n), and n^2 + O(n).





$T \sim an^2$

STATISTICS $R^2 = 0.8802$

RESIDUALS e_1 plot

PARAMETERS $a = 0.0150067$

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

STATISTICS $R^2 = 0.9999$

RESIDUALS e_3 plot

PARAMETERS $a = 0.00300799$ $b = 1.06048$ $c = -4.37174$

$T \sim a \cdot n \cdot \log_2(n)$

STATISTICS $R^2 = 0.9999$

RESIDUALS e_2 plot

PARAMETERS $a = 0.198044$

$T \sim a \cdot n \cdot \log_2(n) + b$

STATISTICS $R^2 = 1$

RESIDUALS e_4 plot

PARAMETERS $a = 0.196574$ $b = 0.6696$

$T \sim a \cdot n \cdot \log_2(n) + bn + c$

STATISTICS $R^2 = 1$

RESIDUALS e_5 plot

PARAMETERS $a = 0.196821$ $b = -0.00174746$ $c = 0.684492$

$T \sim an + b$

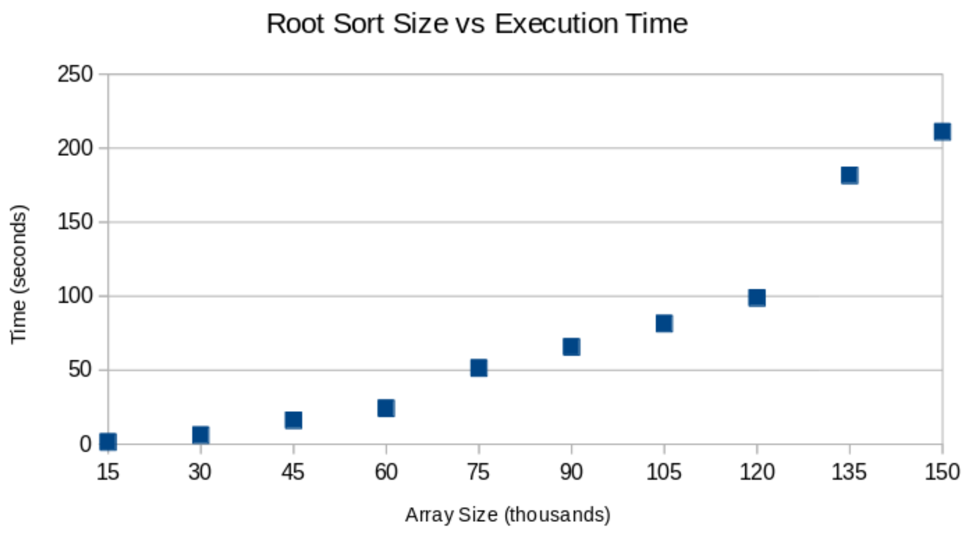
STATISTICS $r^2 = 0.9969$ $r = 0.9985$

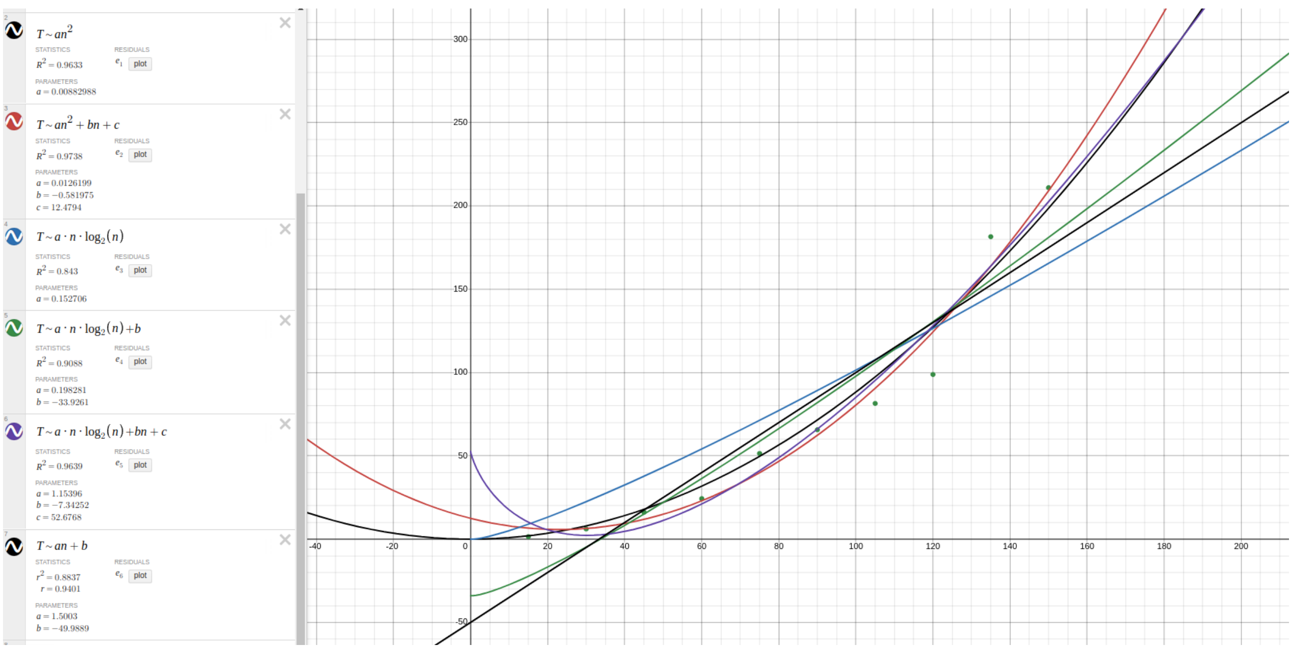
RESIDUALS e_6 plot

PARAMETERS $a = 1.39136$ $b = -10.9893$

Root Sort:

The root sort curved appeared to be a quadratic function with the shape an^2 + bn + c or a polynomial with the shape an*lg(n) + bn + c. The worst fit for the data was an*lg(n), with an R^2 statistic of 0.843. The linear fit an + b was slightly better with a value of 0.8837. The first curve with an R^2 over 0.9 was the curve an*lg(n) + b, with a statistic of 0.9088. The rest of the curves were significantly better than these first three. The next best was an^2 with an R^2 of 0.9633. The curve an*lg(n) + b*n + c was extremely close to the an^2 curve, with a value of 0.9639. Finally, the best fit for the data was an^2 + b*n +c. The complexity for the data is most likely O(n^2), O(n^2) + n, or O(n*lg(n)) + n.





$T \sim an^2$
STATISTICS $R^2 = 0.9633$
RESIDUALS e_1 plot
PARAMETERS $a = 0.00882988$

$T \sim an^2 + bn + c$
STATISTICS $R^2 = 0.9738$
RESIDUALS e_2 plot
PARAMETERS $a = 0.0126199$, $b = -0.581975$, $c = 12.4794$

$T \sim a \cdot n \cdot \log_2(n)$
STATISTICS $R^2 = 0.843$
RESIDUALS e_3 plot
PARAMETERS $a = 0.152706$

$T \sim a \cdot n \cdot \log_2(n) + b$
STATISTICS $R^2 = 0.9088$
RESIDUALS e_4 plot
PARAMETERS $a = 0.198281$, $b = -33.9261$

$T \sim a \cdot n \cdot \log_2(n) + bn + c$
STATISTICS $R^2 = 0.9639$
RESIDUALS e_5 plot
PARAMETERS $a = 1.15396$, $b = -7.34252$, $c = 52.6768$

$T \sim an + b$
STATISTICS $r^2 = 0.8837$, $r = 0.9401$
RESIDUALS e_6 plot
PARAMETERS $a = 1.5003$, $b = -49.9889$

Comparison:

The In Order Sort was tremendously faster than the Root Sort. The In Order Sort was able to sort 100 times the amount of elements the Root Sort did in about the same time span. It took Root Sort 65.6782 seconds to sort 90 thousand elements, while the In Order Sort did 90 million elements 115.262 seconds.

The In Order Sort was better on arrays of every size than the Root Sort. The difference between the arrays was consistent. Root Sort was always far slower than In Order Sort, mostly to the magnitude of 100 times.

The In Order Sort is faster than the insertion, selection, and bubble sorts that we began class with, but it is not faster than any of the sorts from the first project. If this sort was tested in project one, it would be the slowest out of all the comparison sorts. It took 131.373 seconds to sort 100 million elements, while the merge sort, for example, took a little over 7 seconds. The Root Sort is similar to the insertion, selection, and bubble sorts. It can sort arrays below 130 thousand in a reasonable amount of time, but during testing it was found that root sort makes a considerable increase in execution time somewhere between 130 and 132 thousand. This may be because the height of the tree increases, and it is the most significant amount observed since 135 thousand elements was on the upper end of the data tested. It most likely makes larger and larger jumps between every new height, because 2^(height – 1) nodes are being added to the bottom of the tree.

The Root Sort could potentially be faster if a queue structure was implemented to reduce the number of comparisons (in much the same way count sort does), but it most likely could not exceed the speed of the In Order Sort, because there will always be extra nodes to go through in the above layer of the tree, and the amount of those nodes increases exponentially. In the In Order Sort, the amount of nodes is always the number of the array, so there are less structures to create and operate with, therefore reducing the time. Also, every structure only has to be covered once, which would reduce the time.