Samuel Dickerson

9 April 2024

Dr. Spickler

COSC 320-001

Empirical Testing on the Effect of AVL and Binary Search Trees Sizes and Element Ranges on Population Time

The BST and AVL Tree insert nodes to trees differently. Of course, the AVL Tree balances after each insertion, while the BST doesn't balance on an insertion or deletion, but the biggest difference in the trees during testing was that the AVL Trees have counts, so the size of the tree will always be determined by the element range. This makes the AVL Trees easier to deal with if the element range is significantly smaller than the tree size, which is shown in the data.

Tree Size (millions)	Element Range (millions)	Time for AVL (s)	Time for BST (s)
25	0.001	3.19028	25930.5
25	0.01	4.87863	2753.62
25	0.1	6.76564	312.486
25	1	12.4993	52.1667
25	10	25.7079	31.9057
25	25	29.1418	31.3305
25	50	30.2308	30.2174
25	100	31.064	30.0446
25	250	31.6861	30.9372
25	500	31.816	30.9769

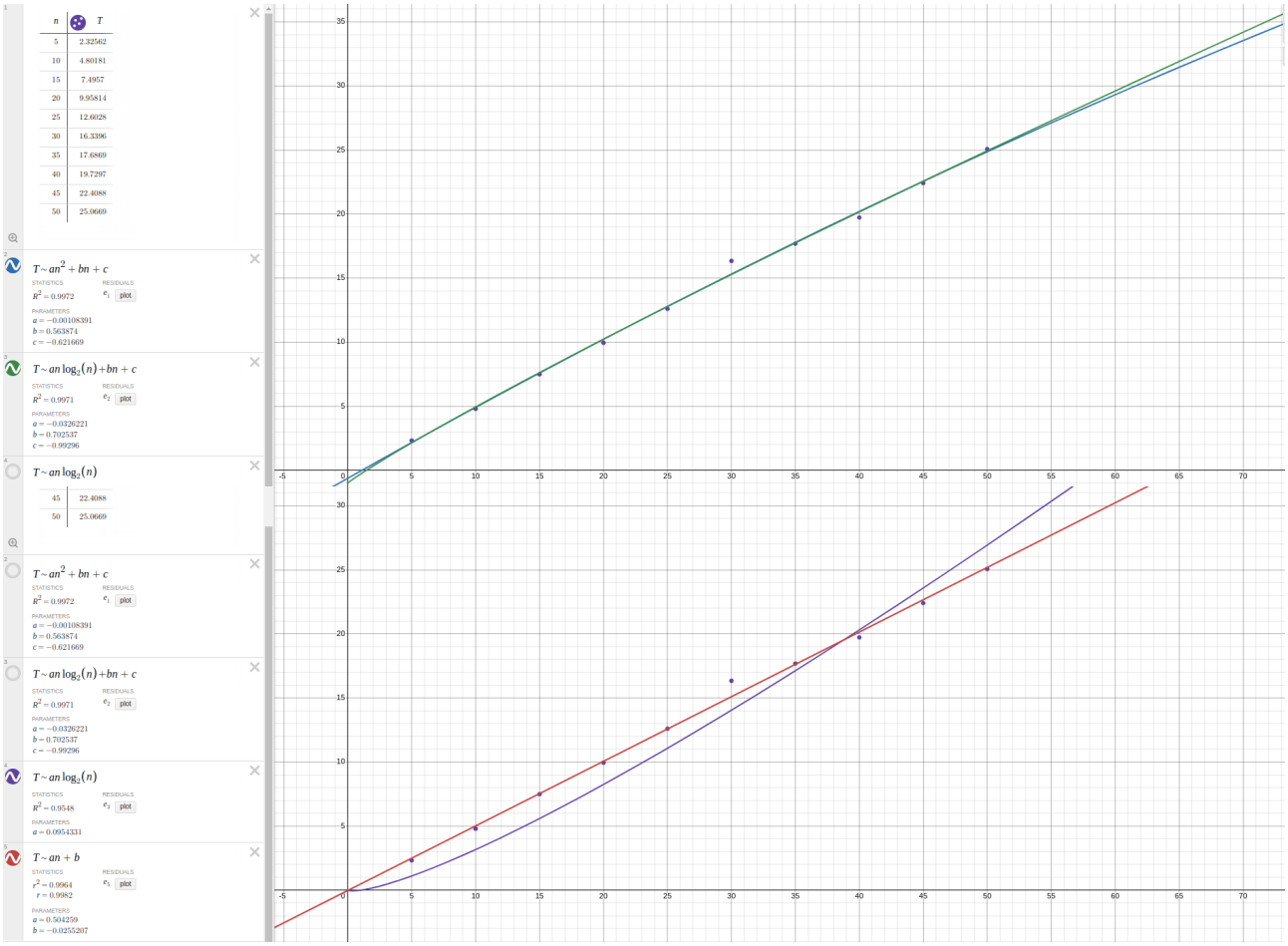
In the figure above, it is easy to see that for an AVL Tree, it is the most efficient with a small element range, and gets less efficient the higher the range goes. However, it appears there is a cut off point for any given tree size, which would demonstrate that the effect of changing the element range decreases as the ratio of tree size to element range gets closer to zero. When the ratio is 0.5, the time is within a second of when the ratio is 0.05. The BST, however, is seriously inefficient when the element range is small. An element range of one thousand took the AVL tree only a few seconds, but took the BST seven hours. Obviously, the AVL tree is a much better implementation for dealing with duplicates than the BST. The BST also demonstrates that once the element range

passes the tree size, the effect of increasing the element range has little to no effect. This is because the chances of duplicates are significantly low after the element range passes the tree size, so increasing the range does not alter the probability in a meaningful way.

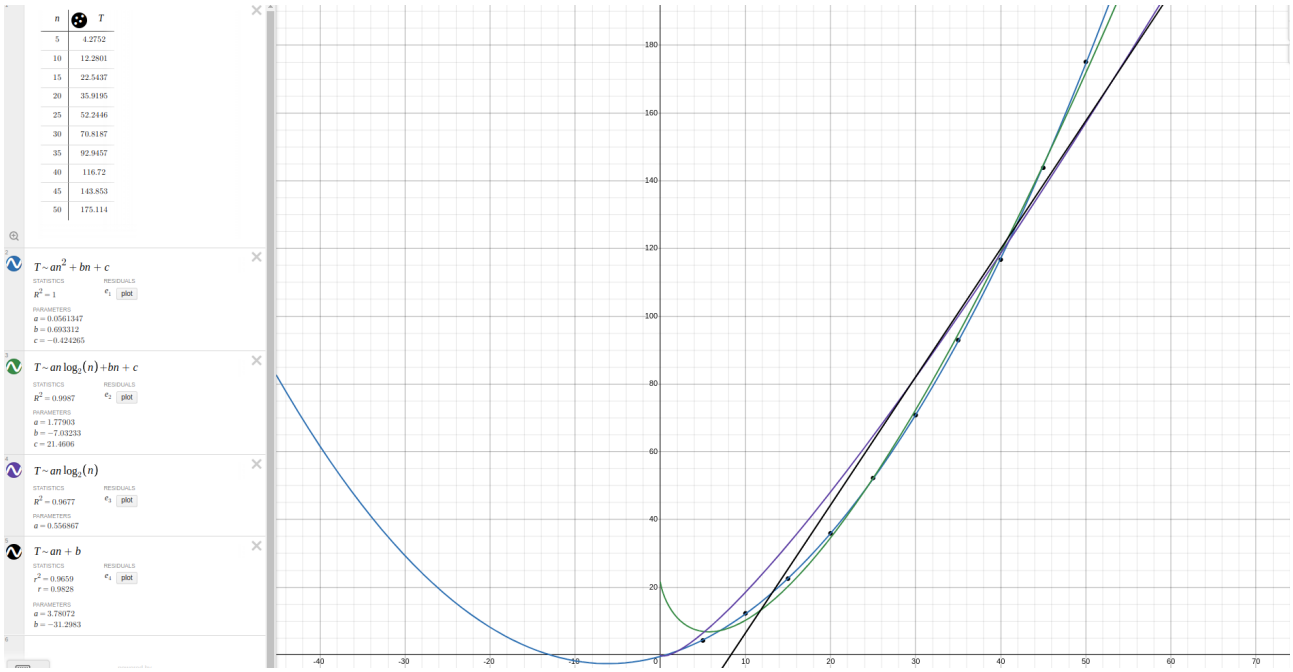
Changing the tree size while maintaining the same element range had a more straightforward impact on the time to insert for both trees. Increasing the tree size simply increased the time it took to insert, and the AVL tree was again much faster because of it's ability to maintain the same tree size no matter the element range. The BST took much longer for each tested tree size, because the tree size was the actual amount of nodes in the BST, whereas the AVL tree only had one million nodes for each tested tree size, because the element range stayed at a constant one million. The AVL tree was able to do much less balancing than the balance function called after the insertion of the BST, because the AVL tree didn't have to do any balancing when a duplicate value was produced. The balancing of the BST significantly increased the time, possibly to a factor of n^2, which will be looked at later. The data for this test can be shown here:

Tree Size (millions)	Element Range (millions)	Time for AVL (s)	Time for BST (s)
5	1	2.32562	4.2752
10	1	4.80181	12.2801
15	1	7.4957	22.5437
20	1	9.95814	35.9195
25	1	12.6028	52.2446
30	1	16.3396	70.8187
35	1	17.6869	92.9457
40	1	19.7297	116.72
45	1	22.4088	143.853
50	1	25.0669	175.114

For the tree size vs. AVL insert curve fit, the typical lines an^2 + bn + c, anlg(n) + bn + c, and an+b were used. The lg(n) and n^2 curves were the best fits with R^2 values of 0.9971 and 0.9972 respectively. The linear fit had a value of 0.9964. All of the usual fits were strong for this data, so the complexity is most likely around nlg(n).

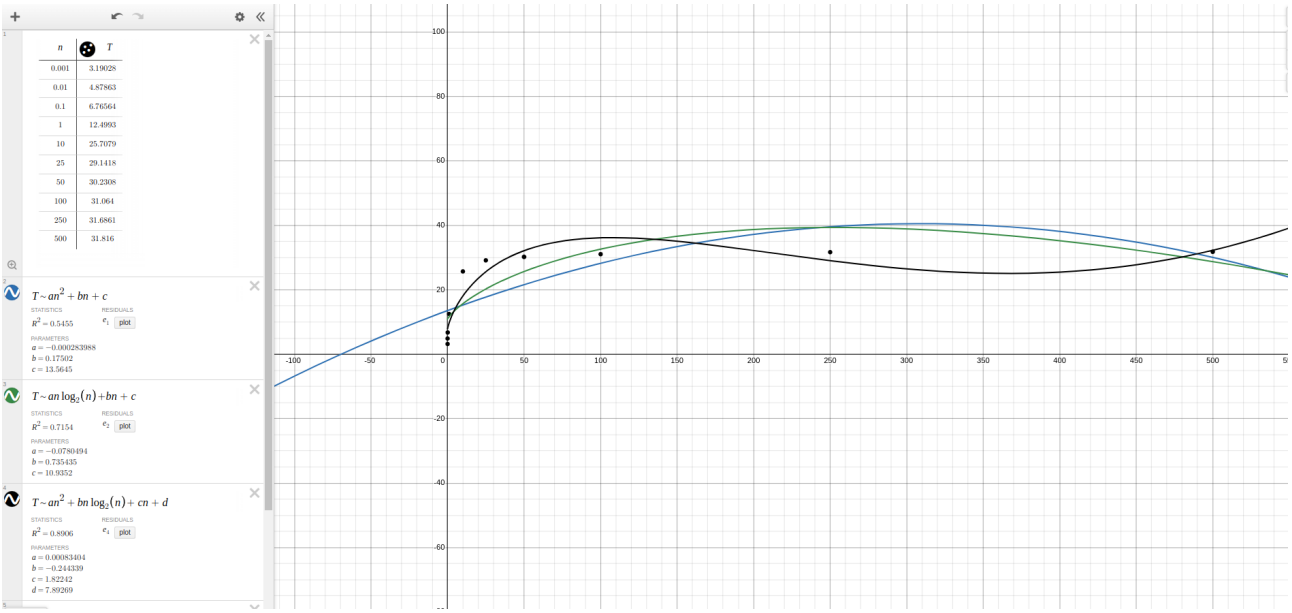


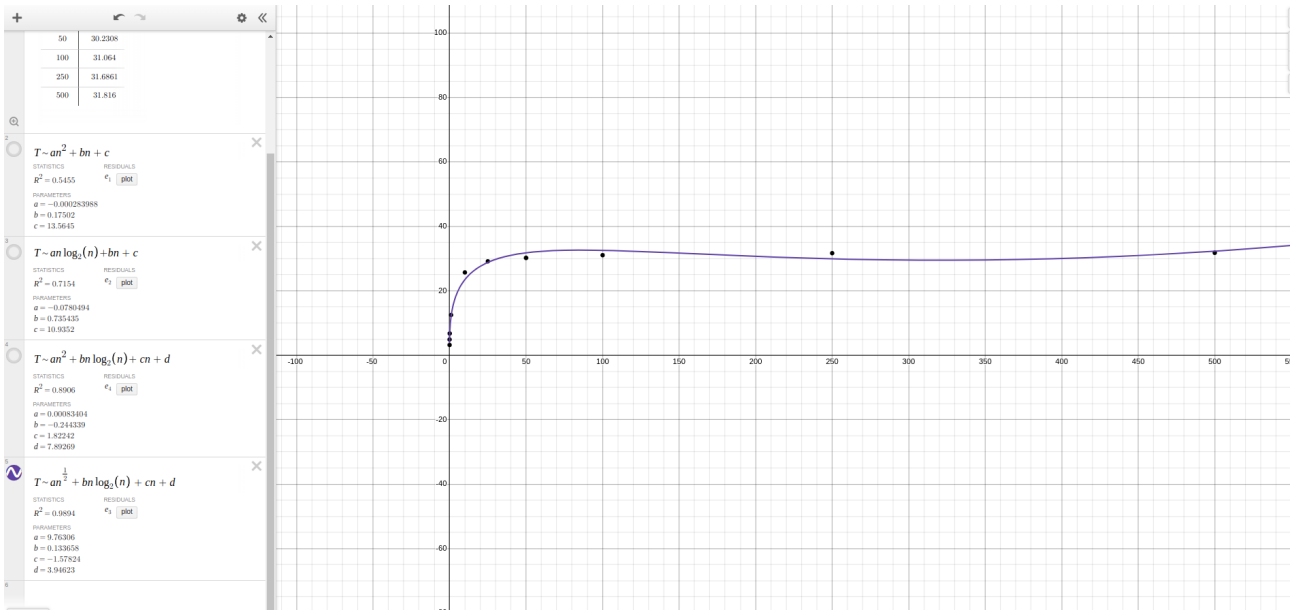
The BST was tested with the same fits as the AVL when analyzing based on tree size. As predicted earlier, the BST runs n^2, assumably because it has to balance every node and the amount of nodes is equal to the tree size.



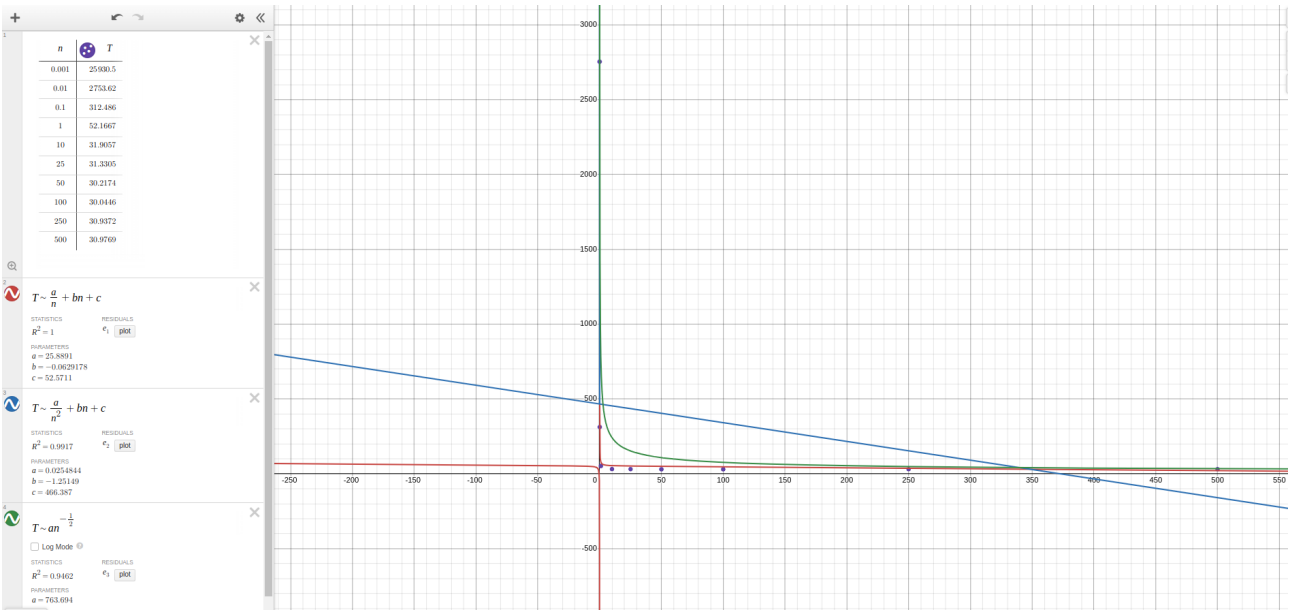
When testing element ranges, the graphs were much more complicated to interpret, and took on shapes that haven't been seen in class much. Each element range was tested with a tree size of 25 million nodes. The AVL tree insertion is faster when the element range is smaller than the tree size, while the BST is the opposite. The BST is tremendously slower when the element range is smaller than the tree size. These graphs were opposite each other, but leveled out to the same point at the end. Both got close to 30 seconds as the element range increased well above the tree size.

The curves used for the AVL tree were an^2 + bn + c, anlg(n)+bn+c, an^2+bnlg(n) + cn + d, and an^(1/2) + bnlg(n) + cn + d. The an^2 curve was a poor fit for the data, with an R^2 of 0.5455. The nlg(n) curve was better, but still poor with a value of 0.7154. The next curve was a combination of those curves, with the addition of a linear and constant term for more accuracy, and it was much higher than the previous two, with an R^2 of 0.8906. From observing the graph, it took the shape similar to the graph of the square root of n, so that was tried last, along with an nlg(n) term and a linear and constant term. The value for that curve was 0.9894, and it was regarded as the best fit for the data. The complexity for this set of data could most accurately be described as some root of n, most likely square. With more testing it could be further identified.





The BST was tested with the most unfamiliar looking curves in the context of this course so far. The BST had a perfect correlation (R^2 of 1) when tested with the curve a/m + bn +c. This was tested due to output getting smaller as the input increased. It makes sense that this would be the best fit for the data, but it was interesting to see something run with this kind of time complexity. 1/n + n would be the best fit for the complexity. The second curve tested was a/n^2 + bn + c, and it was strong as well with a correlation coefficient of 0.9917. The last curve was the weakest and it was an^-(1/2) or a over the square root of n. It can be said with confidence then that element range vs.



insertion time is best described as an inverse relationship.