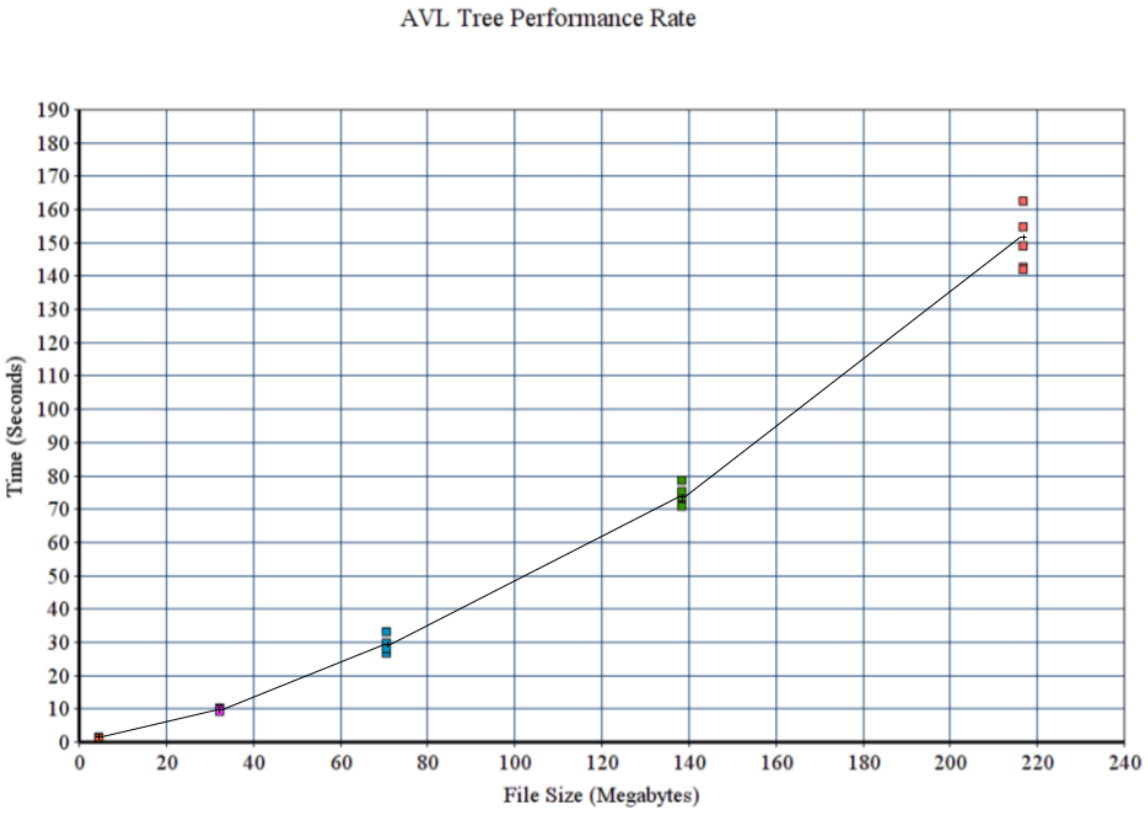
Michael Young & Alex Ingrando

Functionality Report of the AVL Tree vs the Hash Table

In this paper, we will compare the underlying functionality of how we implemented the AVL Tree and the Hash Table in our program. We are trying to discover which implementation is better and quantify this comparison.



The graph above shows the various times that the AVL Tree was able to sort different file sizes. Five sizes of files were chosen to compare the AVL Tree's performance to. The five sizes were: 4.7 Megabytes, 32.5 Megabytes, 70.7 Megabytes, 138.6 Megabytes, and 216.8 Megabytes. The performance of the AVL Tree was measured five times for each file size.

For a file size of 4.7 Megabytes, the five recorded times were: 1.12 seconds, 1.28 seconds, 1.16 seconds, 1.13 seconds, and 1.08 seconds, with an average time of 1.15 seconds and a standard variation of 0.07 seconds. At this rate, it is processing 4.21 Megabytes per second.

For a file size of 32.5 Megabytes, the five recorded times were: 9.12 seconds, 10.19 seconds, 9.61 seconds, 9.86 seconds, and 9.08 seconds, with an average time of 9.57 seconds and a standard variation of 0.47 seconds. At this rate, it is processing 3.88 Megabytes per second.

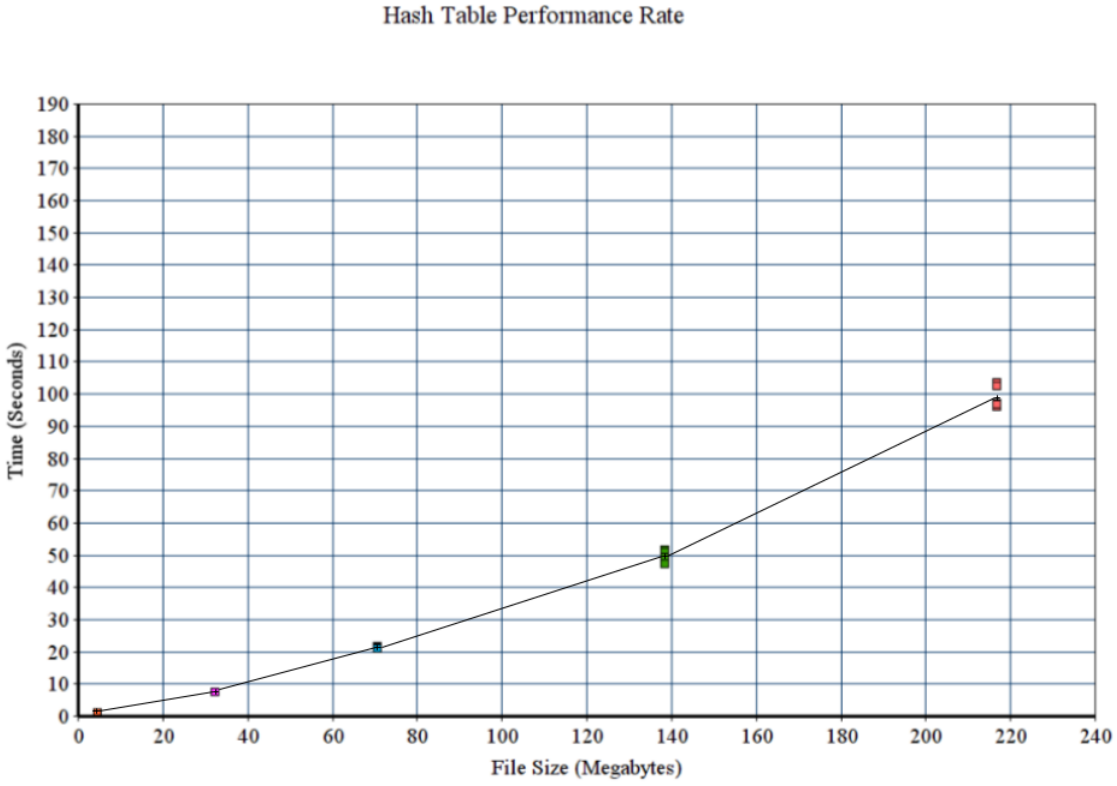
For a file size of 70.7 Megabytes, the five recorded times were: 27.62 seconds, 32.98 seconds, 29.50 seconds, 26.51 seconds, and 27.98 seconds, with an average time of 28.92 seconds and a standard variation of 2.51 seconds. At this rate, it is processing 3.73 Megabytes per second.

For a file size of 138.6 Megabytes, the five recorded times were: 70.89 seconds, 78.35 seconds, 74.25 seconds, 73.56 seconds, and 74.96 seconds, with an average time of 74.40 seconds and a standard deviation of 2.69 seconds. At this rate, it is processing 3.31 Megabytes per second.

For a file size of 216.8 Megabytes, the five recorded times were: 142.65 seconds, 162.26 seconds, 141.85 seconds, 149.06 seconds, and 154.50 seconds, with an average time of 150.10 seconds and a standard deviation of 8.54 seconds. At this rate, it is processing 3.08 Megabytes per second.

By the data extrapolated here, the efficiency of processing each Megabyte of data decreases as the file size increases by a rate of .1 Megabyte per second every 50 Megabytes. At this point, we cannot determine if the Hash Table is going to be better or not, but if implemented correctly, it's O(1) access time should prove to be faster than the AVL Tree's O(log n) time.

The Hash Table has a O(1), making it constant time. This is due to the fact that a Hash Table is able to store and access each element through the use of a hash function converting an element into a number. This number represents the location of the element within the Hash Table. This is not a perfect O(1) because the hash function must be called to find the location, but it does remove the need to performing any searching within the Hash Table under most circumstances.



The graph above shows the various times that the Hash Table was able to sort different file sizes. The exact same file sizes used for the AVL Tree were used for the Hash Table as well. The five sizes were: 4.7 Megabytes, 32.5 Megabytes, 70.7 Megabytes, 138.6 Megabytes, and 216.8 Megabytes. The performance of the Hash Table was measured five times for each file size.

For a file size of 4.7 Megabytes, the five recorded times were: 0.91 seconds, 0.96 seconds, 0.93 seconds, 0.96 seconds, and 0.91 seconds, with an average time of 0.93 seconds and a standard deviation of 0.2 seconds. At this rate, it is processing 5.05 Megabytes per second.

For a file size of 32.5 Megabytes, the five recorded times were: 7.25 seconds, 7.33 seconds, 7.28 seconds, 7.36 seconds, and 7.28 seconds, with an average time of 7.30 seconds and a standard deviation of 0.04 seconds. At this rate, it is processing 5.09 Megabytes per second.

For a file size of 70.7 Megabytes, the five recorded times were: 21.54 seconds, 21.08 seconds, 20.98 seconds, 21.26 seconds, and 21.02 seconds, with an average time of 21.18 seconds and a standard deviation of 0.23 seconds. At this rate, it is processing 5.09 Megabytes per second.

For a file size of 138.6 Megabytes, the five recorded times were: 51.68 seconds, 49.61 seconds, 49.25 seconds, 50.78 seconds, and 47.31 seconds, with an average time of 49.73 seconds and a standard deviation of 1.65 seconds. At this rate, it is processing 4.95 Megabytes per second.

For a file size of 216.8 Megabytes, the five recorded times were: 103.53 seconds, 97.04 seconds, 95.87 seconds, 102.39 seconds, and 96.61 seconds, with an average time of 99.09 seconds and a standard deviation of 3.58 seconds. At this rate, it is processing 4.67 Megabytes per second.

By the data extrapolated here, the efficiency of processing each Megabyte of data decreases as the file size increases by a rate of .04 Megabytes per second every 50 Megabytes. These numbers are considerable improvements over the AVL Tree's numbers. For every file size, the Hash Table performed better than the AVL Tree and was more consistent in its time spread.

At 4.7 Megabytes, the average AVL Tree time was 1.15 seconds, compared to the average Hash Table time of 0.93 seconds. Marking a 0.22 second difference in the Hash Table's favor. The standard Deviation of the AVL Tree was 0.07 seconds and the Hash Table was 0.2 seconds, making the AVL Tree have a more reliable time spread at this level.

At 32.5 Megabytes, the average AVL Tree time was 9.57 seconds, compared to the average Hash Table time of 7.30 seconds. Marking a 2.27 second difference in the Hash Table's favor. The standard Deviation of the AVL Tree was 0.47 seconds and the Hash Table was 0.04 seconds, making the Hash Table have a more reliable time spread at this level.

At 70.7 Megabytes, the average AVL Tree time was 28.92 seconds, compared to the average Hash Table time of 21.18 seconds. Marking a 7.74 second difference in the Hash Table's favor. The standard Deviation of the AVL Tree was 2.51 seconds and the Hash Table was 0.23 seconds, making the Hash Table have a more reliable time spread at this level.

At 138.6 Megabytes, the average AVL Tree time was 74.40 seconds, compared to the average Hash Table time of 49.73 seconds. Marking a 24.67 second difference in the Hash Table's favor. The standard Deviation of the AVL Tree was 2.69 seconds and the Hash Table was 1.65 seconds, making the Hash Table have a more reliable time spread at this level.

At 216.8 Megabytes, the average AVL Tree time was 150.10 seconds, compared to the average Hash Table time of 99.09 seconds. Marking a 51.01 second difference in the Hash Table's favor. The standard Deviation of the AVL Tree was 8.54 seconds and the Hash Table was 3.58 seconds, making the Hash Table have a more reliable time spread at this level.

In terms of average time and standard deviation, the Hash Table performed better than the AVL Tree, with the sole exception of the standard deviation of the AVL Tree performing better than the Hash Table's at 4.7 Megabytes. This possibly makes the AVL Tree more reliable

in it's time spread at smaller datasets. Though reliability is not much of a benefit compared to speed, in which case, the Hash Table still beat the AVL Tree at this level.

As predicted earlier, the Hash Table performed better, and scales at a slower pace than the AVL Tree did. Though the Hash Table did not scale linearly due to the hash function still needing to convert every element into a number. As more data is required, the hash function will be needed to perform more often, increasing the Hash Table's time on a level that is above O(1), but not enough for it to ever be worse off than O(log n).

In conclusion, with the evidence provided, we can determine that the implemented Hash Table performs better than the implemented AVL Tree at every file size, though the AVL Tree may be more consistently timed at smaller data sets.