CS590 Assignment 3

1. **Abstract**

In this experiment, the performance of sorting using a regular binary search tree is compared to that of a self-balancing red-black binary search tree. The performance is defined as the time it takes to build a tree, and report back the data in sorted order. Various classes of input data sets of multiple different dimensions are applied, in order to evaluate the algorithm growth rates under varying input conditions. Overall, the trees performance matched the theory very closely. When using random inputs, both tree sort algorithms behaved as O(nlg(n)). With both ascending and descending data sets, the binary search tree was O(n2), where the red-black tree remained as O(nlg(n)).

1. **Results**

Both the binary search tree and the red-black tree are tested with ascending, descending, and random data sets. For each type of data, several different data sizes are tested, spanning from 10,000 nodes up to 125,000. Each input condition is tested 10 times in order to reduce noise in the measurements. Additionally, some meta-data about the tree performance is collected, including the number of duplicates in the data set, as well as the frequency of rotation/fixup cases in the red-black tree during the insertion routine. All of this data is tabulated below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data** | **Nodes** | **Duplicates** | **"New Size"** | **Duration** |
| **Ascending** | **10000** | 0 | 10000 | 218 |
| **20000** | 0 | 20000 | 1000 |
| **30000** | 0 | 30000 | 2812 |
| **40000** | 0 | 40000 | 6309 |
| **50000** | 0 | 50000 | 10159 |
| **65000** | 0 | 65000 | 20707 |
| **80000** | 0 | 80000 | 36342 |
| **95000** | 0 | 95000 | 54227 |
| **100000** | 0 | 100000 | 61730 |
| **110000** | 0 | 110000 | 67275 |
| **115000** | 0 | 115000 | 76793 |
| **125000** | 0 | 125000 | 94278 |
| **Descending** | **10000** | 0 | 10000 | 229 |
| **20000** | 0 | 20000 | 946 |
| **30000** | 0 | 30000 | 2734 |
| **40000** | 0 | 40000 | 5895 |
| **50000** | 0 | 50000 | 10677 |
| **65000** | 0 | 65000 | 20598 |
| **80000** | 0 | 80000 | 36747 |
| **95000** | 0 | 95000 | 52486 |
| **100000** | 0 | 100000 | 60209 |
| **110000** | 0 | 110000 | 74860 |
| **115000** | 0 | 115000 | 76943 |
| **125000** | 0 | 125000 | 93714 |
| **Random** | **10000** | 0 | 10000 | 0 |
| **20000** | 0 | 20000 | 9 |
| **30000** | 0.1 | 29999.9 | 10 |
| **40000** | 0 | 40000 | 20 |
| **50000** | 0.4 | 49999.6 | 30 |
| **65000** | 0.9 | 64999.1 | 40 |
| **80000** | 0.4 | 79999.6 | 74 |
| **95000** | 0.4 | 94999.6 | 89 |
| **100000** | 0.9 | 99999.1 | 90 |
| **110000** | 1.4 | 109998.6 | 120 |
| **115000** | 2.2 | 114997.8 | 106 |
| **125000** | 1.4 | 124998.6 | 120 |

**Table 1.** Performance measurements for binary search tree build and walk.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data** | **Nodes** | **Duplicates** | **"New Size"** | **Duration** | **BH** | **BH measure** | **Case 1** | **Case 2** | **Case 3** | **Left Rotations** | **Right Rotations** |
| **Ascending** | **10000** | 0 | 10000 | 0 | 13 | 0 | 9971 | 0 | 9976 | 9976 | 0 |
| **20000** | 0 | 20000 | 0 | 14 | 0 | 19969 | 0 | 19974 | 19974 | 0 |
| **30000** | 0 | 30000 | 10 | 15 | 0 | 29967 | 0 | 29973 | 29973 | 0 |
| **40000** | 0 | 40000 | 10 | 15 | 0 | 39967 | 0 | 39972 | 39972 | 0 |
| **50000** | 0 | 50000 | 10 | 16 | 6 | 49966 | 0 | 49971 | 49971 | 0 |
| **65000** | 0 | 65000 | 30 | 16 | 0 | 64961 | 0 | 64971 | 64971 | 0 |
| **80000** | 0 | 80000 | 40 | 16 | 0 | 79965 | 0 | 79970 | 79970 | 0 |
| **95000** | 0 | 95000 | 40 | 16 | 0 | 94962 | 0 | 94970 | 94970 | 0 |
| **100000** | 0 | 100000 | 47 | 17 | 3 | 99964 | 0 | 99969 | 99969 | 0 |
| **110000** | 0 | 110000 | 60 | 17 | 2 | 109961 | 0 | 109969 | 109969 | 0 |
| **115000** | 0 | 115000 | 53 | 17 | 7 | 114963 | 0 | 114969 | 114969 | 0 |
| **125000** | 0 | 125000 | 60 | 17 | 10 | 124963 | 0 | 124969 | 124969 | 0 |
| **Descending** | **10000** | 0 | 10000 | 0 | 13 | 0 | 9971 | 0 | 9976 | 0 | 9976 |
| **20000** | 0 | 20000 | 0 | 14 | 0 | 19969 | 0 | 19974 | 0 | 19974 |
| **30000** | 0 | 30000 | 10 | 15 | 0 | 29967 | 0 | 29973 | 0 | 29973 |
| **40000** | 0 | 40000 | 10 | 15 | 0 | 39967 | 0 | 39972 | 0 | 39972 |
| **50000** | 0 | 50000 | 10 | 16 | 3 | 49966 | 0 | 49971 | 0 | 49971 |
| **65000** | 0 | 65000 | 20 | 16 | 0 | 64961 | 0 | 64971 | 0 | 64971 |
| **80000** | 0 | 80000 | 40 | 16 | 0 | 79965 | 0 | 79970 | 0 | 79970 |
| **95000** | 0 | 95000 | 50 | 16 | 0 | 94962 | 0 | 94970 | 0 | 94970 |
| **100000** | 0 | 100000 | 40 | 17 | 0 | 99964 | 0 | 99969 | 0 | 99969 |
| **110000** | 0 | 110000 | 60 | 17 | 0 | 109961 | 0 | 109969 | 0 | 109969 |
| **115000** | 0 | 115000 | 50 | 17 | 1 | 114963 | 0 | 114969 | 0 | 114969 |
| **125000** | 0 | 125000 | 60 | 17 | 0 | 124963 | 0 | 124969 | 0 | 124969 |
| **Random** | **10000** | 0 | 10000 | 0 | 9 | 0 | 5124.9 | 1938.7 | 3901.7 | 2920.4 | 2920 |
| **20000** | 0 | 20000 | 10 | 10 | 0 | 10262.9 | 3889.4 | 7761.1 | 5825.2 | 5825.3 |
| **30000** | 0 | 30000 | 10 | 10.1 | 1 | 15382.1 | 5824.4 | 11650.6 | 8743.3 | 8731.7 |
| **40000** | 0.2 | 39999.8 | 20 | 10.8 | 0 | 20549.8 | 7773.3 | 15493.9 | 11638.6 | 11628.6 |
| **50000** | 0.3 | 49999.7 | 30 | 11 | 0 | 25680.2 | 9714.2 | 19429 | 14576.6 | 14566.6 |
| **65000** | 0.6 | 64999.4 | 40 | 11 | 7 | 33367.3 | 12572.6 | 25206.4 | 18912.2 | 18866.8 |
| **80000** | 0.6 | 79999.4 | 80 | 11 | 5 | 41020.8 | 15486.3 | 31090.9 | 23290.8 | 23286.4 |
| **95000** | 1.1 | 94998.9 | 84 | 11 | 6 | 48770 | 18528.4 | 36958.5 | 27736 | 27750.9 |
| **100000** | 1.4 | 99998.6 | 91 | 11.3 | 9 | 51333.8 | 19380.2 | 38865.9 | 29114 | 29132.1 |
| **110000** | 1.8 | 109998.2 | 126 | 11.3 | 5 | 56495.5 | 21378.4 | 42707.3 | 32067.9 | 32017.8 |
| **115000** | 1.3 | 114998.7 | 110 | 11.6 | 8 | 58973.1 | 22338.8 | 44719.8 | 33537.5 | 33521.1 |
| **125000** | 1.4 | 124998.6 | 124 | 11.9 | 8 | 64130.4 | 24233.4 | 48524.3 | 36400.2 | 36357.5 |

**Table 2**. Performance measurements for red-black tree build and walk.

Due to the nature of the data generator, there are no duplicates in the ascending and descending data sets. Given the large domain of integers we are generating from, the random dataset produces very few duplicates as well. Note, both the duplicates and “new size” columns are the average of ten iterations, resulting in their averages being non-integer values.

1. **Analysis**

The most straightforward analysis is a comparison so build and sort time between the two trees. First, the duration to build and sort random data sets is plotted below.

Chart, line chart

Description automatically generated

**Figure 1.** Sort time of random data for binary search tree (blue) and red-black tree (red).

While there is clearly a bit of noise in the experiment’s measurements (particularly a small spike at 115,000 nodes), the data is fairly clean. In addition to the two traces plotted belonging to the two trees, a reference c\*nlog(n) line is also plotted in black. The first observation one can make, is both tree sort algorithms perform similarly for this type of input data. This matches the theory, as the the binary search tree will be somewhat balanced with truly random inputs. A second observation is both algorithms appear to behave as O(nlg(n)) given these inputs, again matching the theoretical performance.

Next, the performance for ascending and descending input datasets is plotted.

Chart, line chart

Description automatically generated

**Figure 2.** Sort time of ascending and descending data sets both binary search tree and red-black tree.

With ascending and descending datasets, the two algorithms no longer perform similarly. Looking at the plotted cn2 reference line, the binary search tree appears to be behaving at O(n2). The red-black tree significantly outperforms the BST, behaving similarly to the random input dataset. The major degradation in the BST algorithm is due to major tree imbalance. With ascending and descending datasets, we have a maximally unbalanced tree, and each of the n tree insertions costs O(h). Once again, this matched the theoretical performance of the BST.

Next, we can analyze the ‘meta-data’ of the red-black tree. An interesting first plot shows, for the random data inputs, how input dimension affects each of the red-black tree counters.

Chart, line chart

Description automatically generated

**Figure 3.** Red-black tree meta-data for varying input data sizes.

As we would expect, this data has a linear relationship with the input dimension. This matches our expectation; for every node we insert, we either do nothing, or some combination of case 1, 2, and 3. The frequency of each individual case is directly linearly proportional to the input data size. As expected, case 1 occurs at the highest frequency, at roughly 50% the data input size. This matches the theoretical expectation, as roughly half of our insertions will have a red uncle/aunt. It’s also shown that the number of left rotations and right rotations are roughly equal, which is expected given the symmetry of the tree with random data inputs. Given the linear relationship between the frequency of the rotations/cases and the input data dimension, the random data set will have an O(nlg(n)) relationship to the run time.

Not plotting for brevity, the proportions of case 1, 2, and 3 differ for ascending and descending data inputs. For ascending datasets, we hit cases 1 and 3 close to 100% of the insertions. This is because all insertions will end up in the right subtree. The balancing of the tree will thus always be case 3, involving a left rotation. The same applies for the descending data input, but we see case 1 and case 2, and balancing using a right rotation. Once again, given the linear relationship between the frequency of the cases/rotations and the input data dimension, ascending and descending data sets will have an O(nlg(n)) relationship to the run time.

Finally, the black height of the trees are examined. Due to the speed of the black height measurement, very little can be learned. The black height is measured so quickly, it is well below the noise floor, with the largest of trees taking 10ms to measure. However, we can verify that the black height of these trees matches the theoretical O(lg(n)). For a tree of size 125,000, this gives us an order of magnitude black height on the order of 16.9. The ascending and descending trees at 125,000 nodes both have a black height of 17. The random input red-black tree at 125,000 nodes has a black height of 12.

1. **Conclusion**

This experiment shows that this implementation of both the binary search tree and the red-black tree performs very closely to the theoretical expectations. For a random input dataset, both tree sort algorithms perform at O(nlg(n)). While both on the same order of magnitude, we can note that the binary search tree slightly out-performs the red-black tree, due to the red-black tree having to run an insert-fixup routine for every insert. This fixup makes the red-black tree slightly more balanced, but the cost slightly outweighs the benefit for this balanced dataset. However, for sorted input datasets, the binary search tree sort performs at O(n2). This is a worst case scenario for this algorithm; for this same input, the red-black tree that still performs at O(nlg(n)). Finally, we note that the red-black tree meta-properties (case 1/2/3 and left/right rotation) have a linear relationship to the input data dimension. The particular ratio of each of these frequencies to the data input dimension is dependent on “how sorted” the input data is, with fully ascending and fully descending data inputs being the extremes.