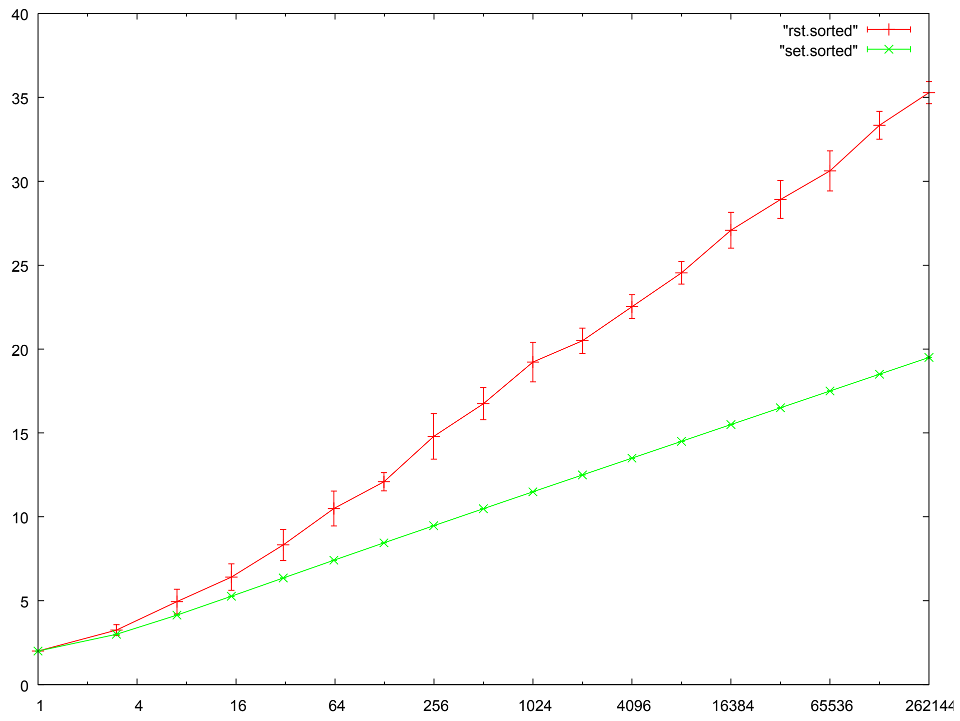
Bench Tree Analysis

The three data structures BST, RST, and Set are all implementations of binary search tree. The BST is just a simple implementation with no added features. RST is almost identical BST except that it introduces randomness in the structure so it is independent of how data is inserted. Set uses more complicated algorithm to make sure the tree structure is more rigidly balanced which speeds up find operation. In fact, set is the fastest data structure out of the three in terms of data retrieval. BST and RST is identical in retrieval efficiency if the data was originally inserted in a random order. BST becomes astonishingly inefficient when the data was inserted in a sorted order.

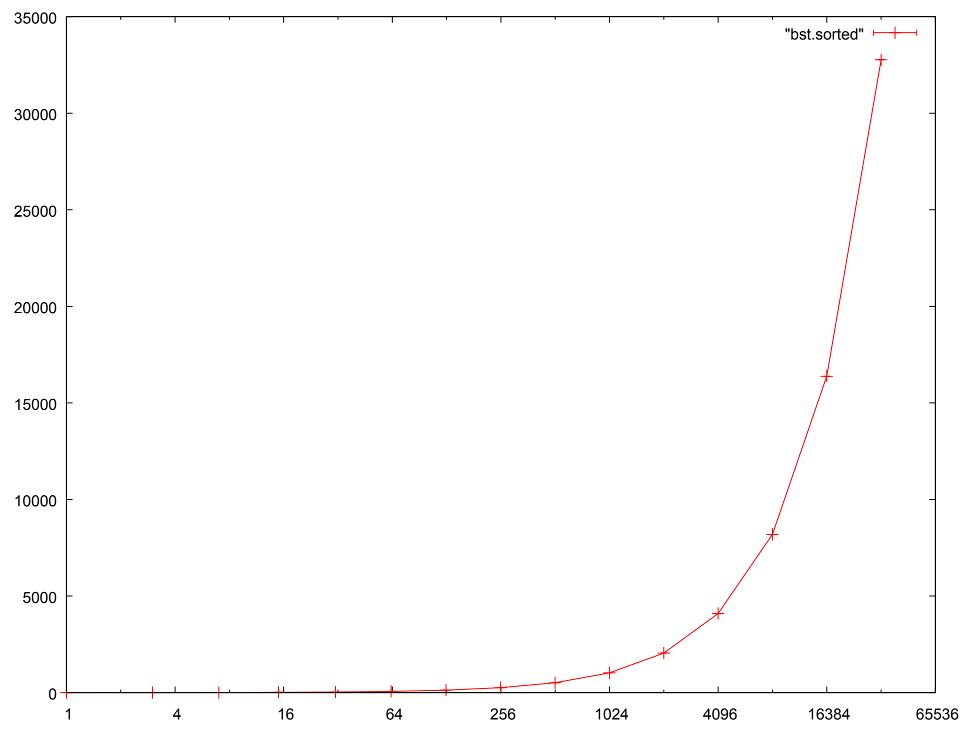
Below is a graph that shows the number of comparison taken by using RST and Set when data was originally inserted in sorted sequence.



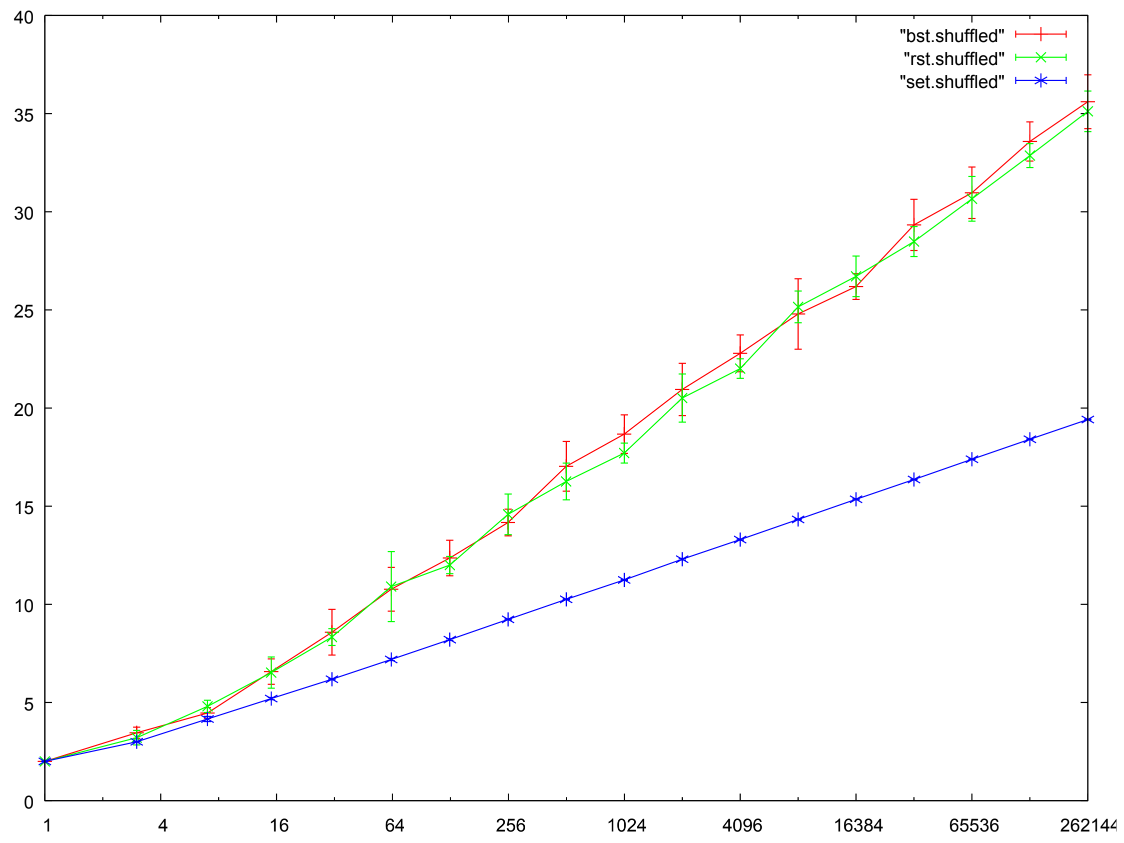
Since the X axis is in log scale, and the plot for both RST and Set is roughly a straight line, we can be sure that the time complexity for "Find" operation is in big-O log(n). RST is slightly less efficient than Set since RST plot has a bigger slope, suggesting that it takes more comparison than Set does for the same amount of Data.

Now, let's take a look at BST in sorted insertion.
The plot has X axis in log scale as well, but the BST plot shows an exponential growth instead of a straight line. This indicates that the time complexity for "Find" operation in BST is not big-O log(n). In fact, by observing the data, BST uses big-O (n) on average to find a particular data. This is on the same level as arrays, linked lists, etc..

One interesting thing about find in BST is that if the data were originally inserted from small to big, and there are N data to insert. Then BST takes N comparisons on average to find a particular data. But if the original data were inserted from large to small, then it only takes only $((1+n)*\frac{N}{2}+n)/n \ = \frac{(1+n)}{2}+1$ comparisons (roughly n/2) to find a particular data. This is due to the fact that in a BST, the data is first compared to see if it's smaller than current node, if not then compare to see if data is bigger than current node. So if we insert in ascending order, each data is the right child of its previous data, and so two comparisons are needed to go one level below. Whereas in descending order, the first comparison (if data < node) directly goes into the lower level.



For shuffled insertions, the time complexity for “Find” operations is big-O log(n) for all three data structures. Which is the expected result. The plot of RST and BST are nearly identical. This also expected since the only advantage of RST over BST is that it ensures that the tree will have a random structure. Since our data was inserted randomly, the structure of a BST is also random which of course requires the same amount of comparison as RST. Set on the other hand is immune to insertion sequence and apparently faster in retrieval than BST and RST. Its more rigidly balanced structure is the reason for this superiority.



Conclusion:

For shuffled insertions, RST performs the same as BST, Set performs slightly better than both.

For sorted insertions, BST has big-O (n) for find, yet RST and Set have big-O log(n). Set still performs better than both BST and RST.