Experiment 1

Inserting 5,000 elements in increasing order into my AVL tree, C# SortedDictionary and a simple binary search tree.

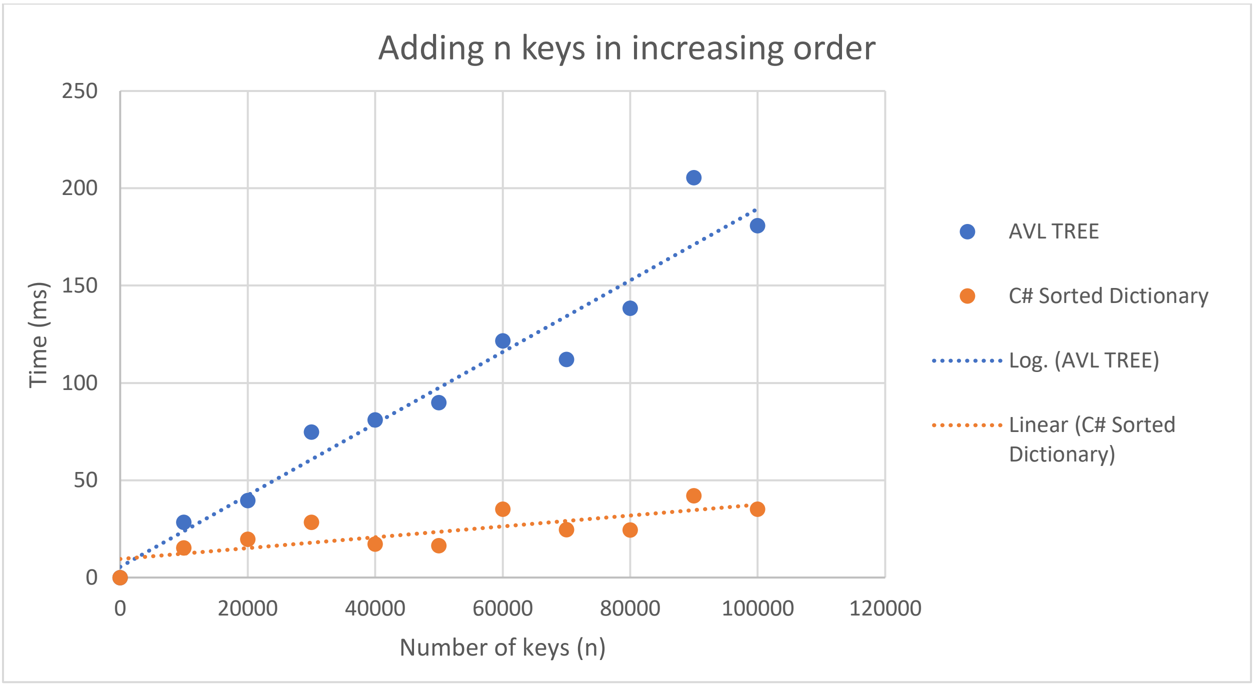
Data Structure	Time (ms)
AVL Tree	24.1001
C# SortedDictionary	11.220
Simple BST	1521.8641

Observations and Analysis:

As expected, the simple binary tree performs really poor when keys are added in order because a linked list is created and when it contains n elements, Add has θ(n) complexity. My AVL tree and the built in C# SortedDictionary perform very similarly because they are both self balancing binary trees. The C# SortedDictionary persorms 2 times faster, this might be due to micro-optimizations or because Add in the C# SortedDictionary might be iterative while it is recursive in my AVL tree.

Experiment 1.1

Adding 10,000 to 100,000 keys in increasing order into my AVL tree, C# SortedDictionary.



Observation and Analysis:

Both data structures exhibit linear running time for adding n ordered keys which means that individual calls to Add run in approximately constant time. They should run in O(log n) time however, even for

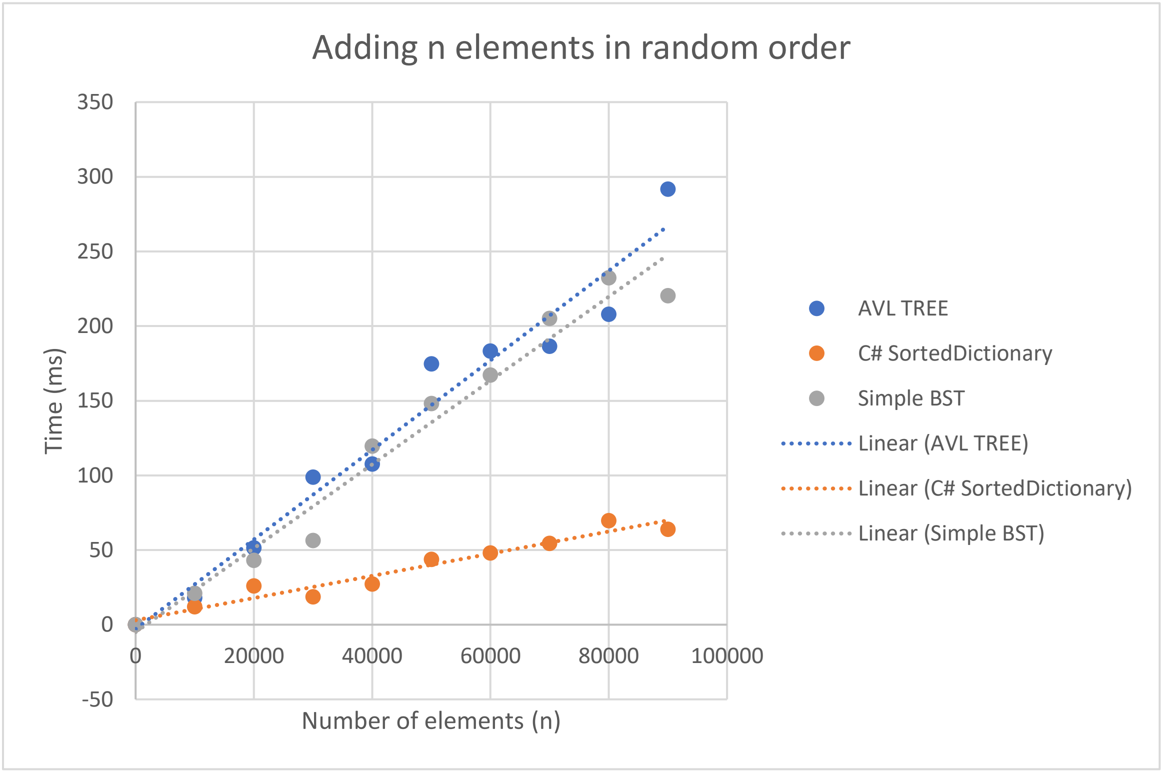
100,000 keys the depth of a balanced binary tree will be around 16. That is why (log n) appears as a constant on the graph.

Possible reasons for C# SortedDictionary being faster than the AVL tree:

- It uses red-black tree which performs less rotations when elements are added
- It uses iterative logic instead of recursive
- It used micro-optimizations

Experiment 2

Adding 10,000 – 100,000 keys in random order into my AVL tree, C# SortedDictionary and a simple binary search tree.

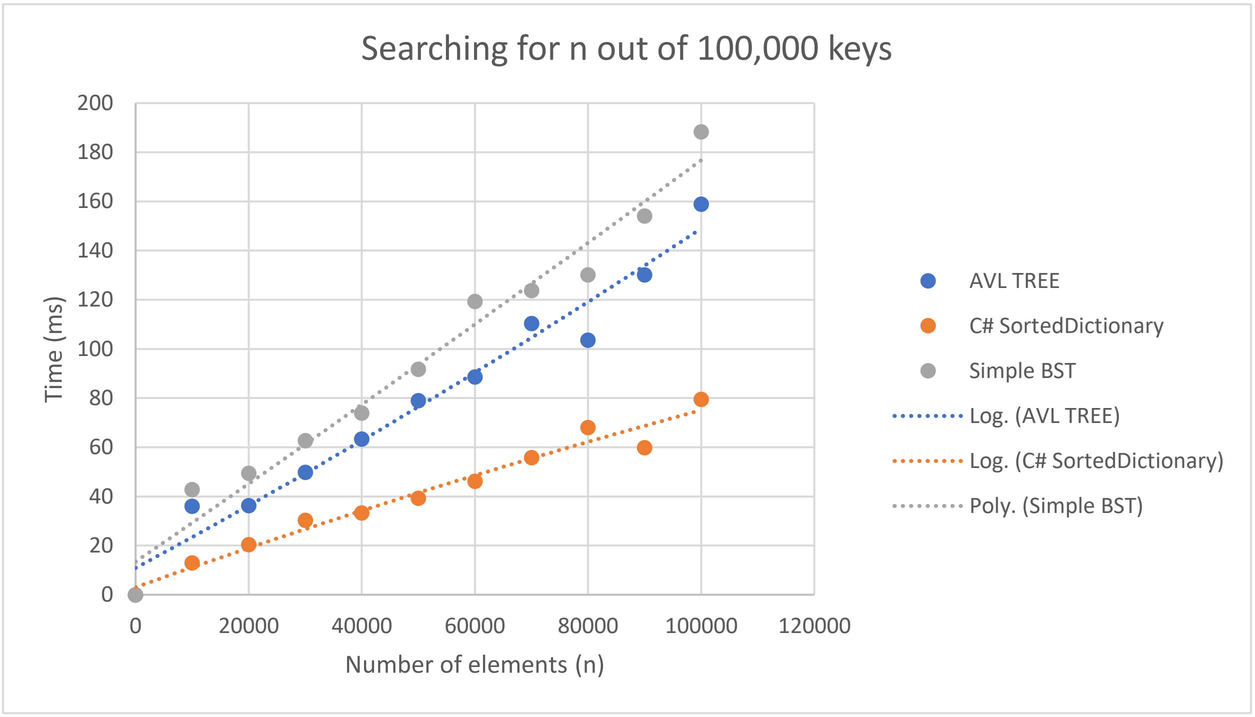


Observation and Analysis:

The comparison between the AVL tree and the C# SortedDictionary is similar to the previous graph. The comparison between the AVL tree and the simple BST looks interesting. They seem to have very similar performance. This might be because, the AVL tree has a lower height but it spends some time doing the rotations with each Add. While the simple BST might not be as balanced but still balanced enough to display O(log n) performance on average. This shows that using a self balancing binary tree is not necessary when the input data is random.

Experiment 3

Searching for 10,000 – 100,000 out of 100,000 random keys in my AVL tree, C# SortedDictionary and a simple binary search tree.



Observation and Analysis:

Comparing the AVL tree and the simple BST, the lower height of the AVL tree shows up on the better performance of it in this experiment. However, it is not significantly faster. The C# SortedDictionary is closer in performance to the other two data structures because not many micro-optimizations can be used. And I believe its faster because my trees use recursion which slows them down.