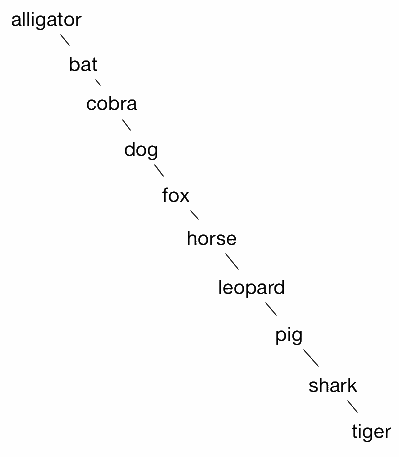
Andrew Tilisky

Ap. 4, 2012

Data Structs 316 Dr. Duan

AVL Search Tree

 The two search trees are easily comparable because they are by definition the same data structure. As such their space complexity will always be O(n); the structures requires only enough memory to store each new node. The best case search, insert, and delete complexities are all the same for this structure with or without AVL behavior. It will take O(logn) time to complete these operations in a balanced tree. However, the rules of AVL trees were devised to prevent the worst-case O(n) running time for binary search trees. When a sorted list is fed into an algorithm that adds in sorted order a binary search tree will actually be a linked list. G.M. Adelson-Velskii and E.M. Landis incorporated a rotation operation into the addition process of each node. In this program that operation is conducted for a child after a recursive addition to that child. Adding a new node to the right child of a node implies that the right child may be imbalanced. After checking the heights of the left and right nodes the subtree or its children are rotated to restore balance. This ensures to O(logn) running time for AVL trees and guarantees quicker search time than simple BSTs outside scenraios where nodes are added to the simple BST so specifically that the tree is balanced. The addition iterator for program two was implemented to add the middle node of a data subset and then the remaining halves were passed recursively to have their middle nodes added etc.

The build times for AVL trees increase proportionally to the dataset. Increasing the number of random additions in an average case necessitates more rotations. Another obstacle for simpler implementations like this one is that the recursive height method is called so numerously that given its O(logn) time it still adds considerable latnecy when the dataset is large enough. For tremendous datasets a lookup table would need to be maintained and used. This idea was attempted here but wasn’t ultimately necessary. Other approaches might also be necessary. Even with such a small data sample the increase could be logarithmic although figure two is not very helpful for drawing conclusions. Search times for the AVL were better, which is expected. The hashmap achieves constant time for get/put operations so it obviously was not beaten.

AVL tree Build times

Nodes|times-> avg

19976 15150 16103 15648 15829 15719 15689.8

9988 3344 1885 2029 2469 2238 2393

4994 471 436 330 315 330 376.4

2497 175 80 85 75 80 99

AVL tree Search times

avg

44 25 25 30 20 28.8

23 30 35 158 57 60.6

25 13 10 15 15 15.4

10 5 10 10 80 23

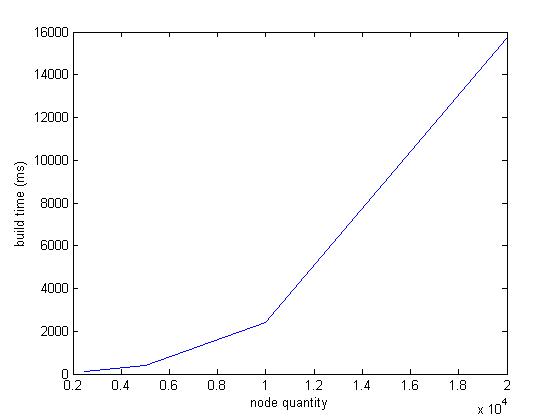
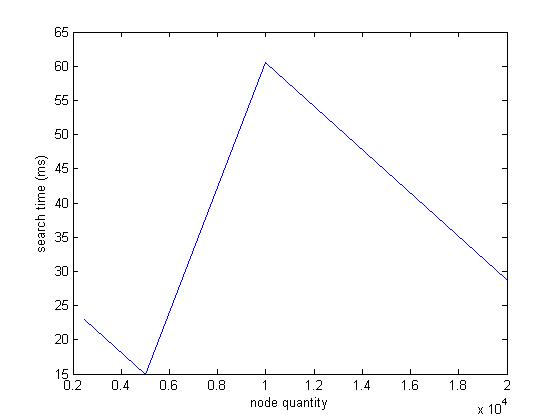


Figure build times

Figure 2 search times

BST search times

Nodes empirical times

10 92 322

19977 93

Hashmap average search times

Nodes avg time

10 25

9988 39

19977 27