Runtime Analysis of Binary Search Trees and AVL Trees

Esteban Fernando Gonzalez

UC Santa Barbara
CS 130A

efg@umail.ucsb.edu

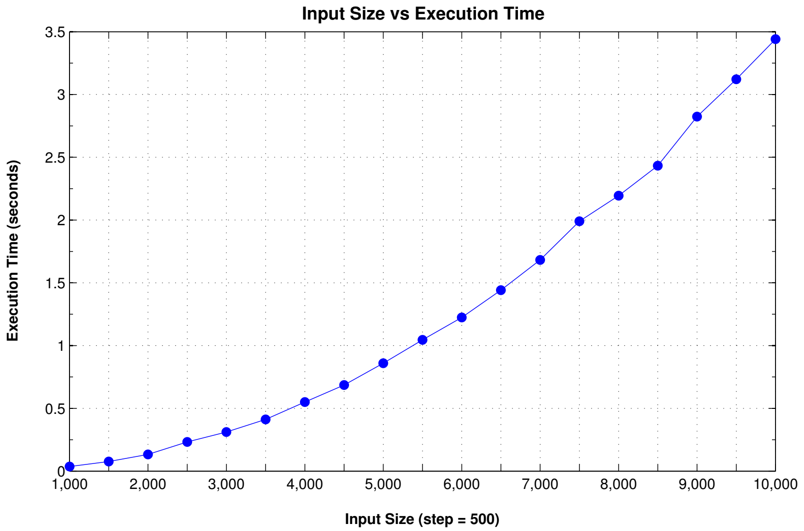
May 15, 2014

Overview

1. Binary Search Trees
   - Insert, Access, and Delete in Increasing Order
   - Insert/Access in Increasing Order, Delete in Reverse Order
   - Insert, Access, and Delete in Random Order

2. AVL Trees
   - Insert, Access, and Delete in Increasing Order
   - Insert/Access in Increasing Order, Delete in Reverse Order
   - Insert, Access, and Delete in Random Order

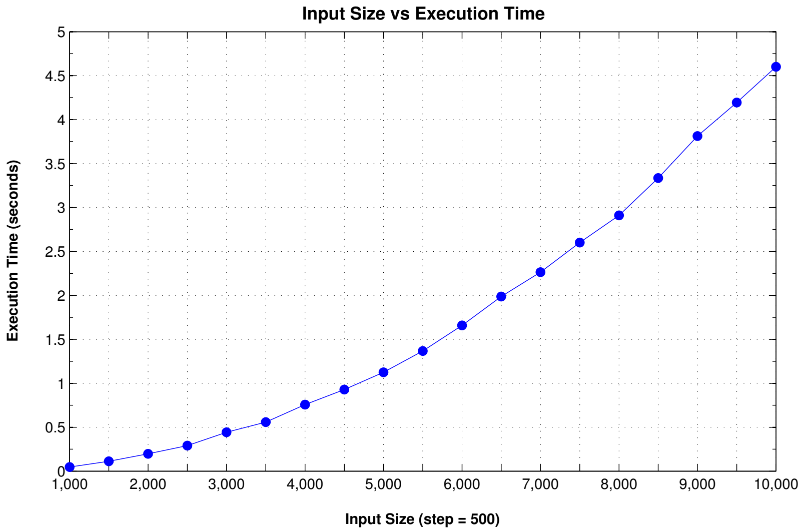
Insert, Access, and Delete in Increasing Order



Analysis of Operations in Increasing Order

The operations performed on the BST were N insertions in increasing order, then N accesses in increasing order, followed by N deletions in increasing order. The execution time was measured for this full set of operations at each N. This means that for insertions, we have the worst case where our BST is really just a linked list. So inserting and accessing are $\mathcal{O}(N)$. For deletions, we are always deleting the root node, so we have the best case of $\mathcal{O}(1)$. From this small dataset, we find that input size is roughly proportional to execution time.

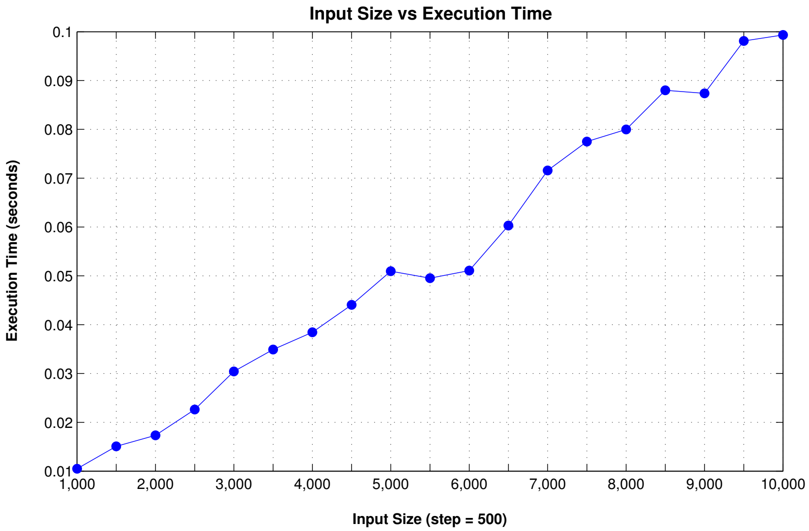
Insert/Access in Increasing Order, Delete in Reverse Order



Analysis of Operations in Increasing/Reverse Order

The operations performed on the BST were N insertions in increasing order, then N accesses in increasing order, followed by N deletions in reverse order. The execution time was measured for this full set of operations at each N. This means that for insertions, we have the worst case where our BST is really just a linked list. So inserting and accessing are $\mathcal{O}(N)$. For deletions, we are always deleting the rightmost leaf node, so we also have the worst case of $\mathcal{O}(N)$. From this small dataset, we find that input size is roughly proportional to execution time.

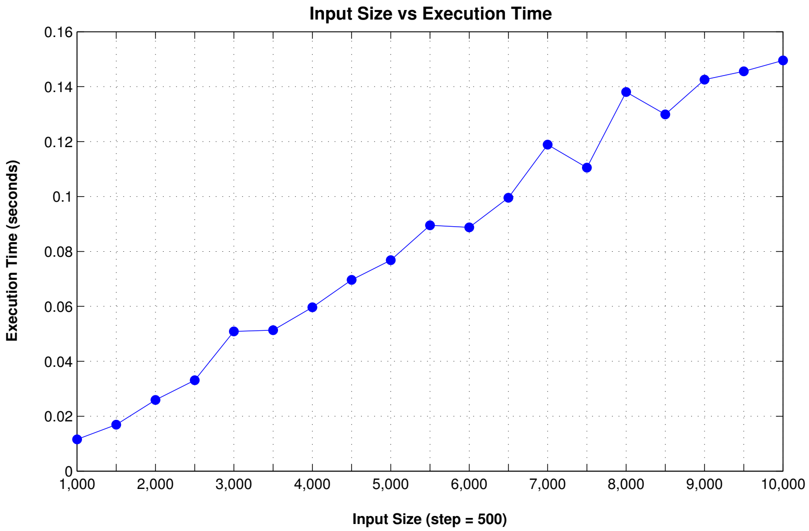
Insert, Access, and Delete in Random Order



Analysis of Operations in Random Order

The operations performed on the BST were N insertions in random order, then N accesses in a different random order, followed by N deletions in another random order. The execution time was measured for this full set of operations at each N. This means that for insertions, accesses, and deletions, we expect to be closer to the average case of $\mathcal{O}(\log N)$. However, from this small dataset, we find that our plot is very much linear.

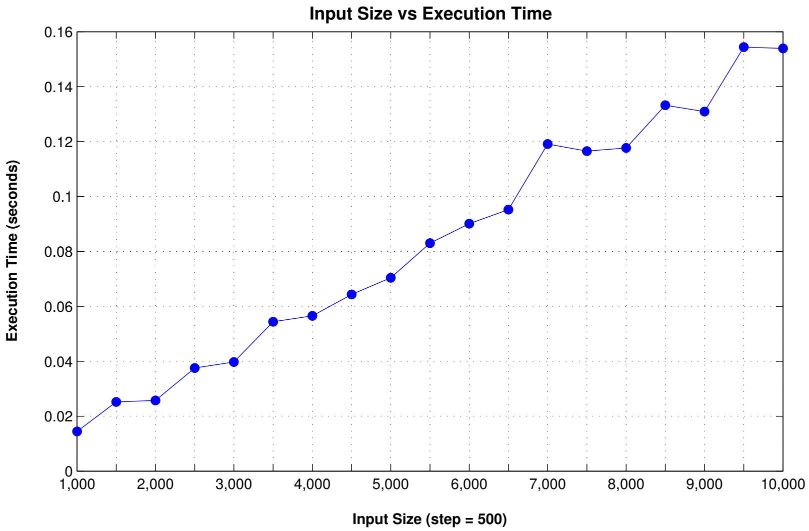
Insert, Access, and Delete in Increasing Order



Analysis of Operations in Increasing Order

All AVL operations are $\mathcal{O}(\log N)$ but from this small dataset, we find that input size is roughly proportional to execution time. These results were however much better than the BST with the same operations.

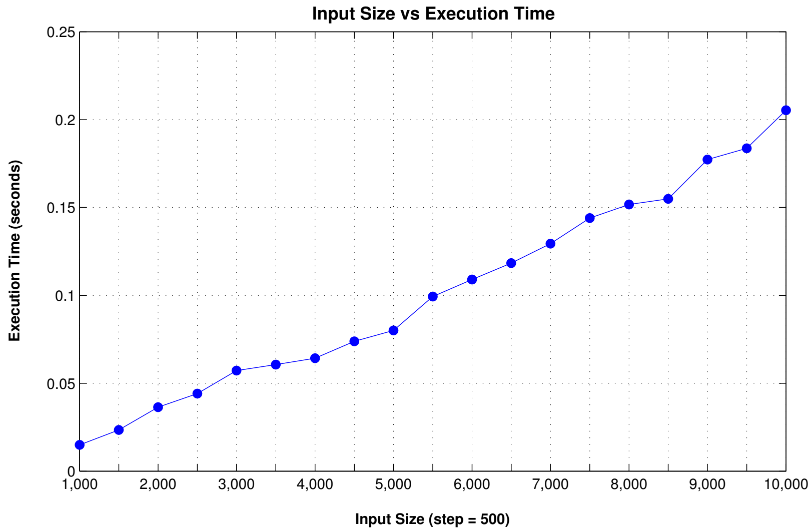
Insert/Access in Increasing Order, Delete in Reverse Order



Analysis of Operations in Increasing/Reverse Order

All AVL operations are $\mathcal{O}(\log N)$ but from this small dataset, we find that input size is roughly proportional to execution time. These results were however much better than the BST with the same operations. Another interesting note is that these results were almost identical to the last trial.

Insert, Access, and Delete in Random Order



Analysis of Operations in Random Order

All AVL operations are $\mathcal{O}(\log N)$ but from this small dataset, we find that input size is roughly proportional to execution time. These results were however much better than the BST with the same operations. I expected all trials for the AVL to be about the same and that is the case, however, the random operations had the slowest runtime.

The End