Comparing BST and AVL trees

Ayna Sultanova

25 January 2023

1 Introduction

In this paper, we investigate BST and AVL insertion and search times. The comparison will be based on the time algorithms take to insert and search randomized strings with different sizes in trees.

2 BST

A Binary Search Tree (BST) is a data structure that is used to store data in a hierarchical, ordered manner. Each node in a BST contains a value, a left child, and a right child. The left child of a node contains a value that is less than the value of the node, and the right child contains a value that is greater than the value of the node. The value of the root node of a BST is the largest in the tree, and the values of the leaf nodes are the smallest. The data in a BST is organized in such a way that it can be quickly searched, inserted, and deleted. The time complexity of the basic operations (search, insert, delete) in a BST is $O(\log n)$, making it a very efficient data structure for many applications. Additionally, a BST can also be used to implement a variety of other data structures such as sets and maps.

Insertion: $O(h)$, where h is the height of the tree. In the worst case, the tree may be skewed and the height may be equal to n (number of nodes), resulting in a time complexity of $O(n)$. However, on average, the height of a well-balanced BST is $O(\log n)$, resulting in an average time complexity of $O(\log n)$ for insertion.

Search: $O(h)$, with the same explanation as above. The worst-case time complexity is O(n) and the average time complexity is $O(\log n)$.

Deletion: $O(h)$, with the same explanation as above. The worst-case time complexity is $O(n)$ and the average time complexity is $O(\log n)$.

3 AVL tree

A balanced binary search tree, or AVL tree, is a type of binary search tree that guarantees a height of $O\log(n)$ for n nodes, where n is the number of nodes in the tree. This is in contrast to a regular binary search tree, where the height can be as large as n, leading to poor performance for searching and insertion operations.

An AVL tree achieves this balance by enforcing a height difference of at most 1 between the left and right subtrees of any node. If the height difference exceeds 1, the tree is rotated to restore balance. This process is called AVL rotations.

AVL trees are particularly useful for real-time applications that require fast search and insertion operations, such as databases and operating systems. They are also used in situations where memory is limited and the tree must be as compact as possible.

The time complexity of insertion and search are the same - $O(\log(n))$. The worst case - $O(n)$.

4 Methodology

The algorithms were implemented using C++. In this experiment, insertion and search were tested 100 times, up to 1000 strings. Sizes of strings for random data are from 1 to 1000. For ordered data is fixed size - 5. The results are presented in nanoseconds and represent the average time each algorithm takes to insert and search each element.

5 Results

Graphs of the results of the experiments are represented in Figures 1, 2,3, and 4.

5.1 Random data

According to Figure 1, insertion to the AVL tree takes longer time than the BST. Whereas BST takes less time. However, the BST search takes more time than the AVL tree search.

Figure 3 shows the average time to insert one random string 100 times. All functions show constant time performance. However, AVL tree insertion performs worse than BST insertion, and BST search shows worse results than the AVL tree search.

5.2 Ordered data

According to Figure 2, ordered data AVL search performs better than any other function. Ordered data BST search performs the worst. AVL tree insertion performs better than the BST insertion.

Figure 4 shows the average time to insert and search one ordered string 100 times. Here, the performance of BST search is the worst among the all functions. AVL search is the fastest. AVL tree insertion performs better than BST insertion.

6 Conclusion

For the random data set BST insertion works better. For the same case, but for searching element - AVL search works better than the BST. For ordered data set AVL tree insertion and search shows the fastest perfomance.

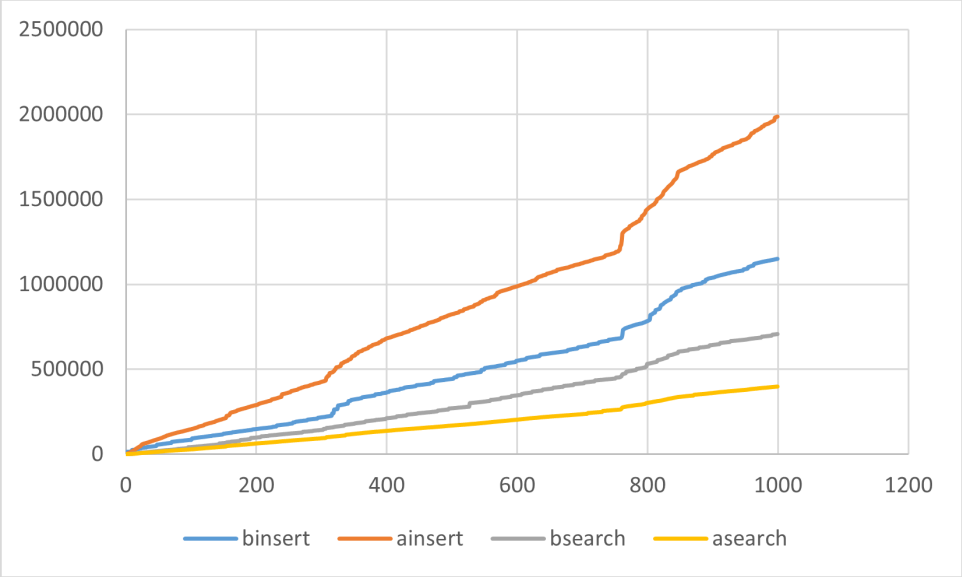

Figure 1: Time to insert and search random single string in BST and AVL tree

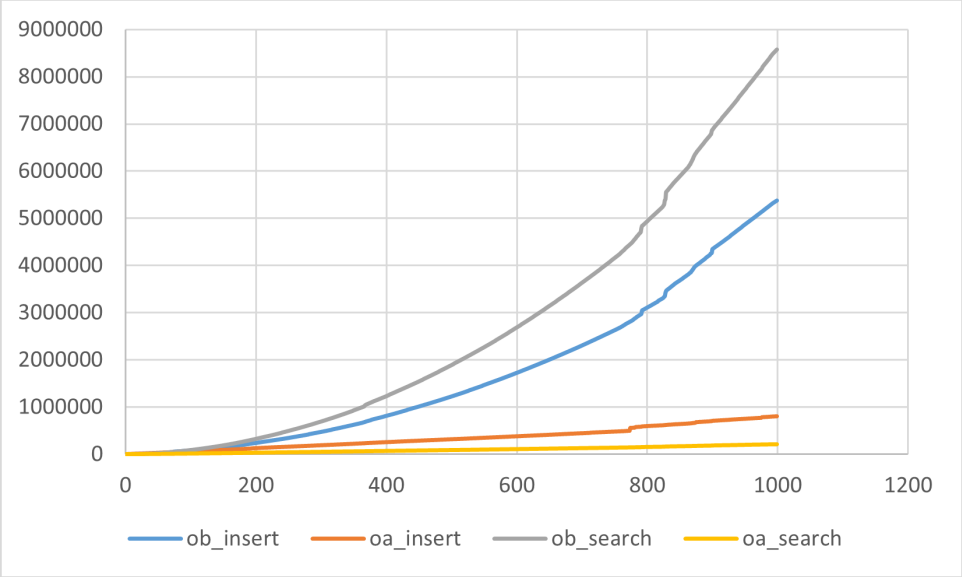

Figure 2: Time to insert and search ordered strings in BST and AVL trees

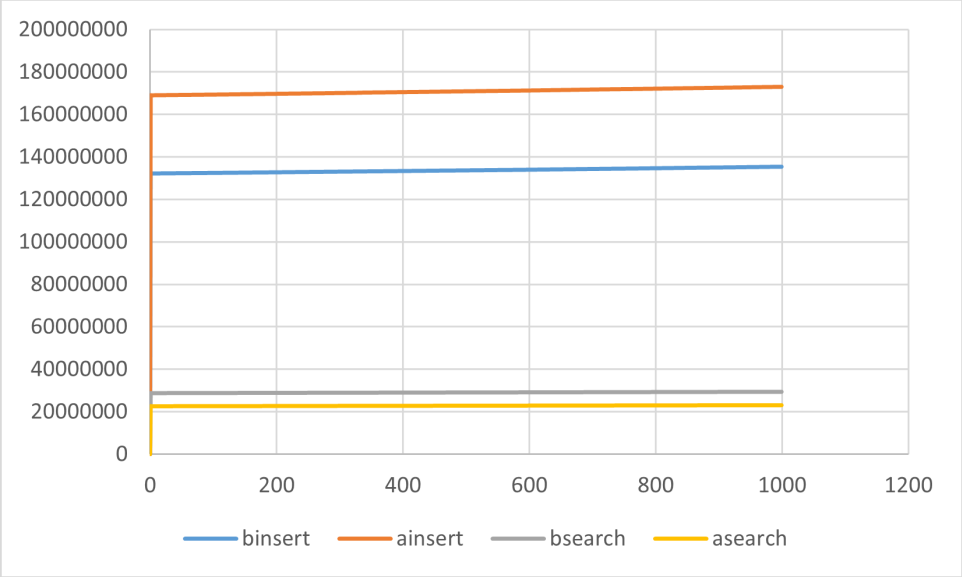

Figure 3: Average time to insert and search one random string 100 times

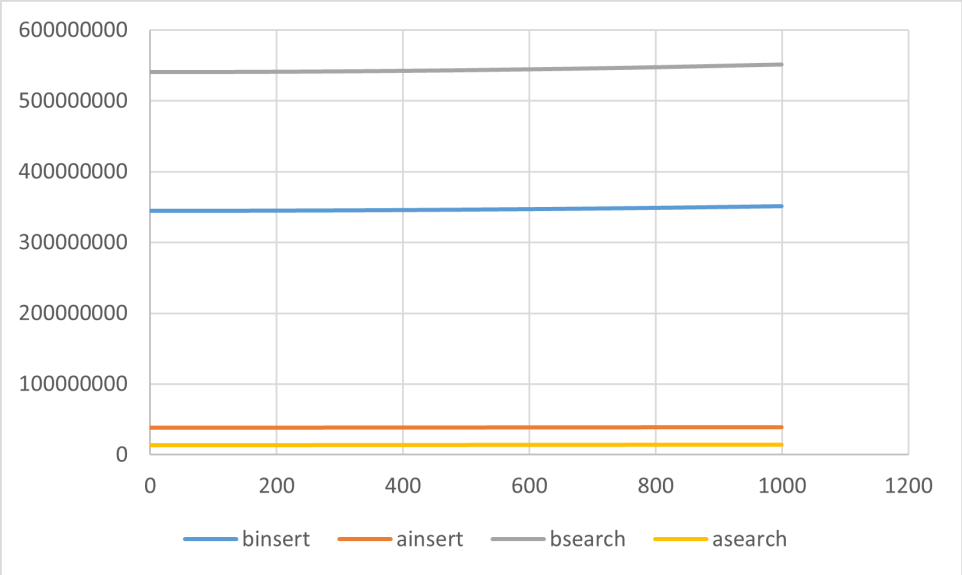

Figure 4: Average time to insert and search one ordered string 100 times