Table 1 Random Insert

size \op	BST insert	BST find	AVL insert	AVL find
1000	0.000401	0.000476	0.001012	0.000201
10000	0.008485	0.013263	0.014348	0.008385
100000	0.114199	0.128272	0.178888	0.073246

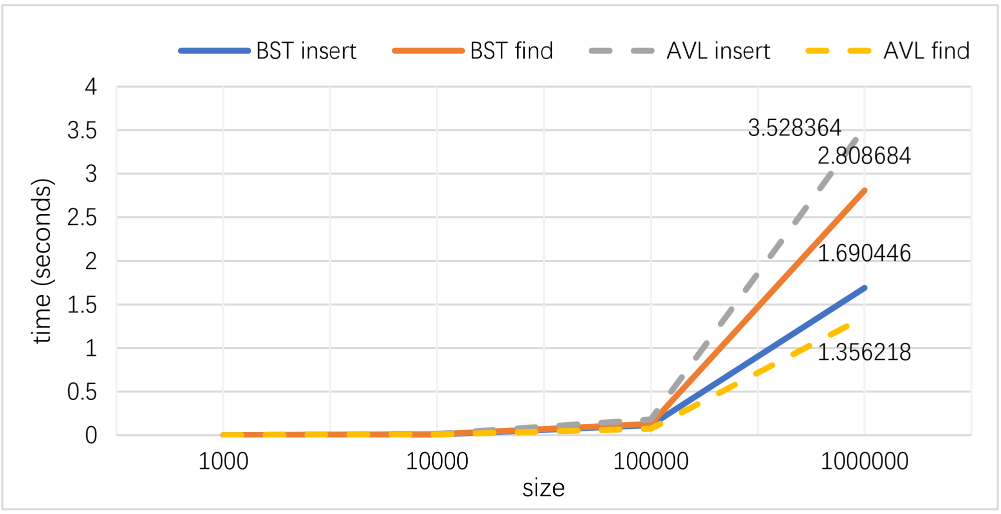

Figure 1 Random Insert

First, a random number generator is used to generate 1000 to 1,000,000 pieces of data for the insert operation. BST and AVL were tested separately, and the results are shown above. From the results, we can see that the insertion time of BST is less than that of AVL with random data, because AVL will dynamically adjust during the insertion time. On the other hand, the lookup time is better for AVL than BST, because AVL always guarantees that the tree is balanced and the lookup overhead is closer to O(logn). However, BST is more dependent on the distribution of data and often its tree structure is not as balanced as AVL.

Table 2 Skewed Insert

size \ op	BST insert	BST find	AVL insert	AVL find
1000	0.003862	0.009365	0.000719	0.00081
10000	0.443012	0.971319	0.006682	0.002102
100000	82.84609	105.0991	0.050596	0.016344

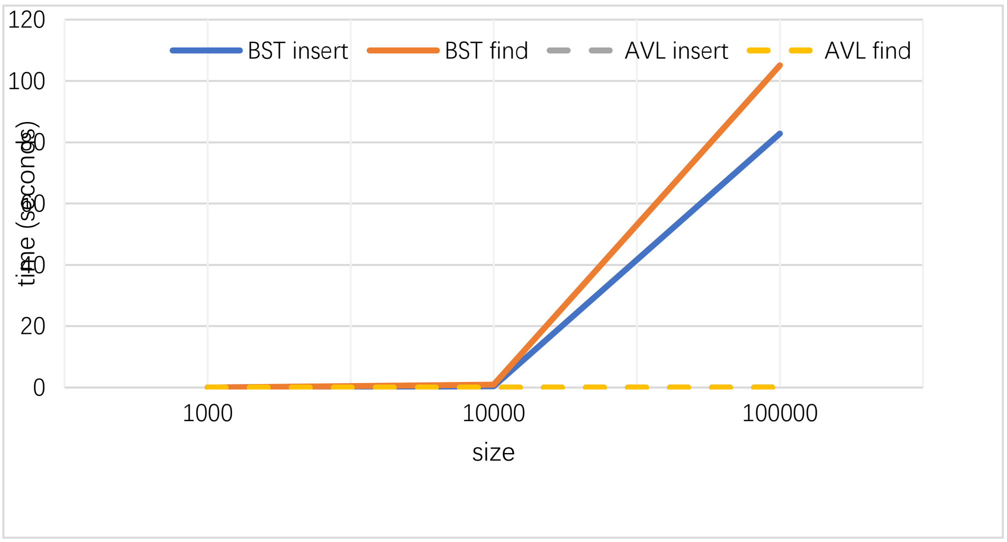

Figure 2 Skewed Insert

Then, the performance of AVL and BST was tested by generating three data sequences of size 1000, 10000 and 100000 by skewed order. From the above results, we can see that the insertion and lookup overhead of BST is much higher than that of AVL with skewed order data, because AVL can dynamically adjust the structure of the tree during insertion, making subsequent insertion and subsequent lookup operations much easier. For skewed order datasets, this causes the BST insertion lookup overhead to increase from O(logn) to O(n), so its performance is severely degraded compared to random datasets.