Comparing two binary trees:
AVL search tree, Binary search tree

Khashimov Akmalkhon

February 19, 2021

1. Goal

Our goal: find out the more efficient search tree (AVL vs Binary search tree) for various input sizes. CRITERIA: Time complexity of search and insertion.

2. Description

AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.

Binary search tree, also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree.

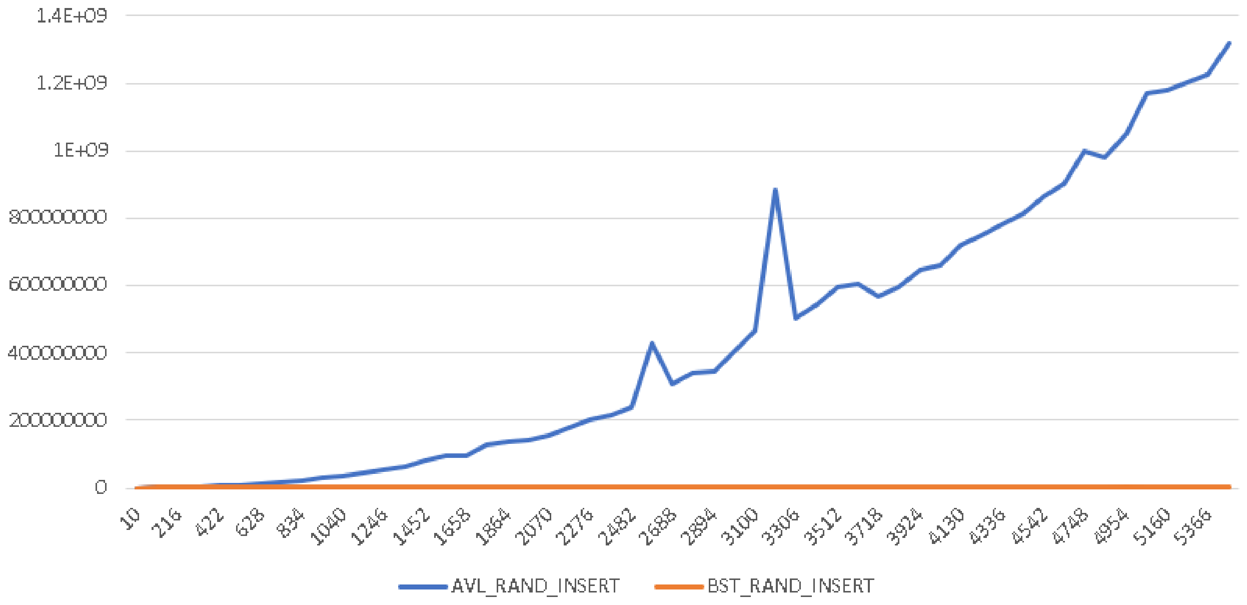
3. Methodology

We compare insertion time for random elements, slightly random, and ordered elements.

In addition, search time for existing and non-existent element in all of the above cases.

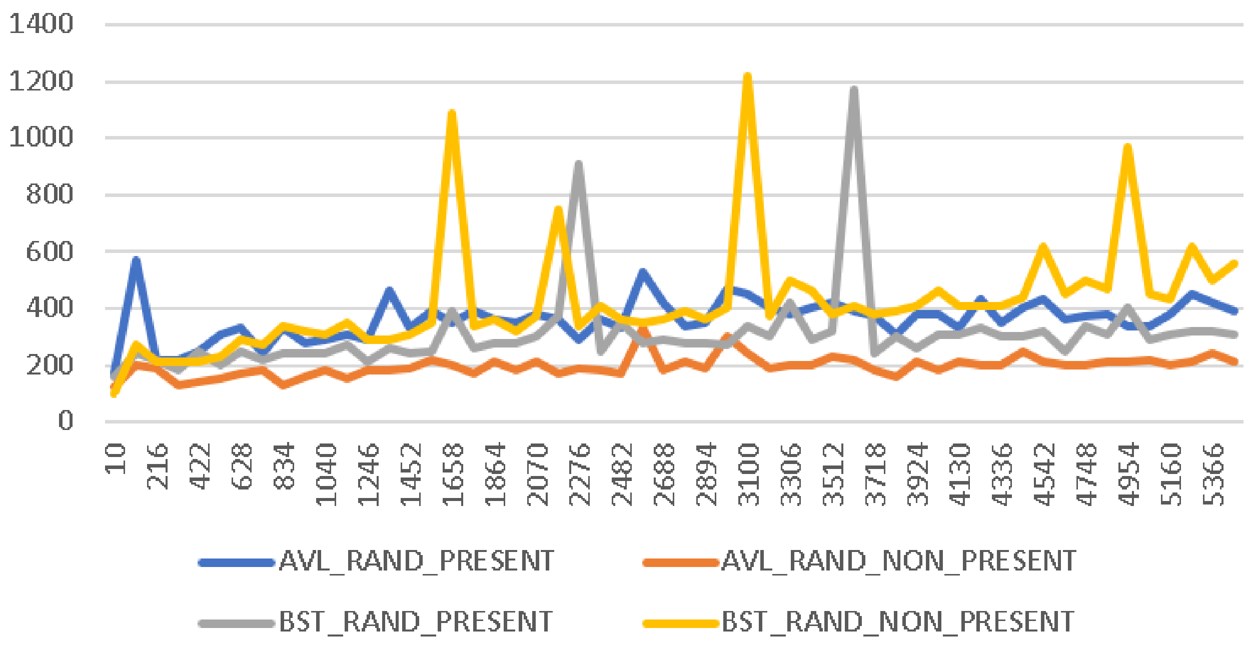
4. Results

AVL vs BST insertion time (RANDOM elements):



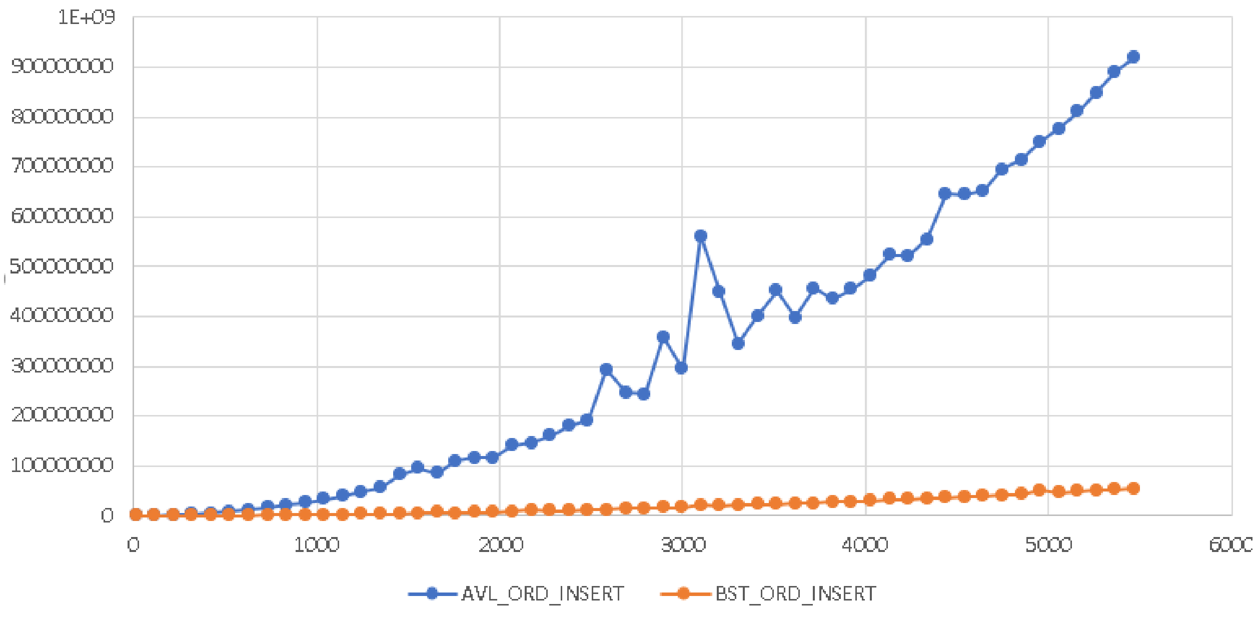
Obviously AVL time is dependent on size of input.

Present and non-present element search time RANDOM

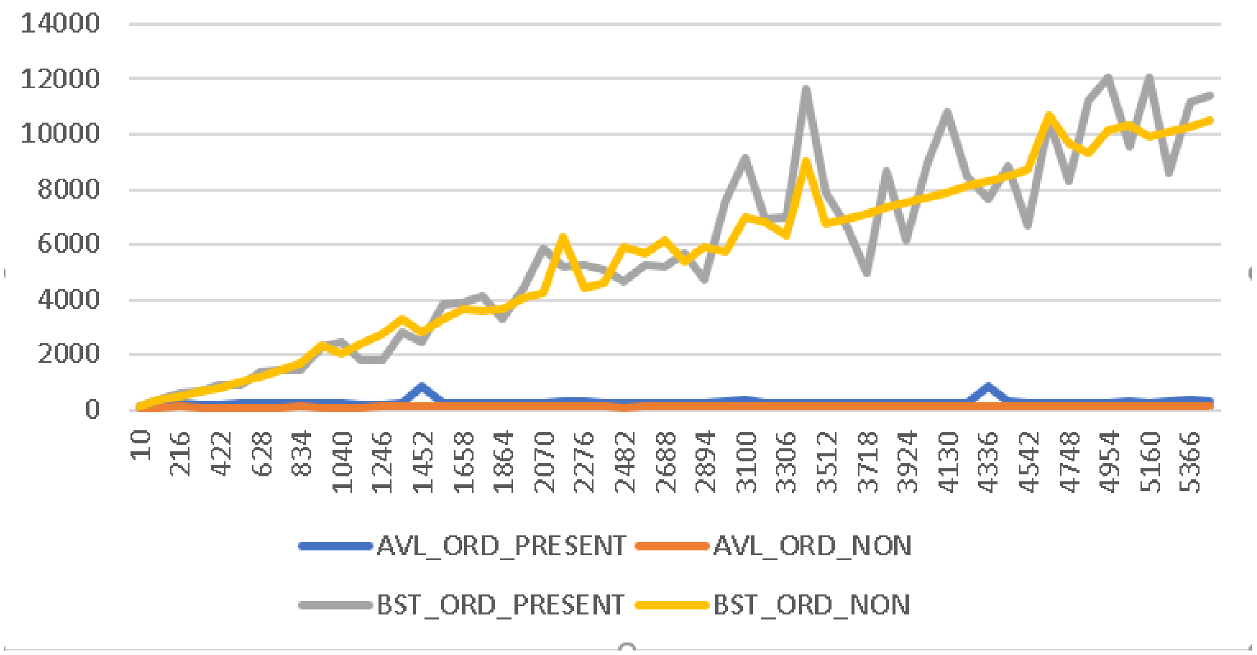


There are outliers, but we can clearly see the trend: Searching for non-existent element in AVL is the most efficient and searching for existing element in BST is more efficient than searching for existing element in AVL, however there are worst cases, which make everything much worse, compared to AVL searching for existing element.

AVL vs BST insertion time (ORDERED elements):

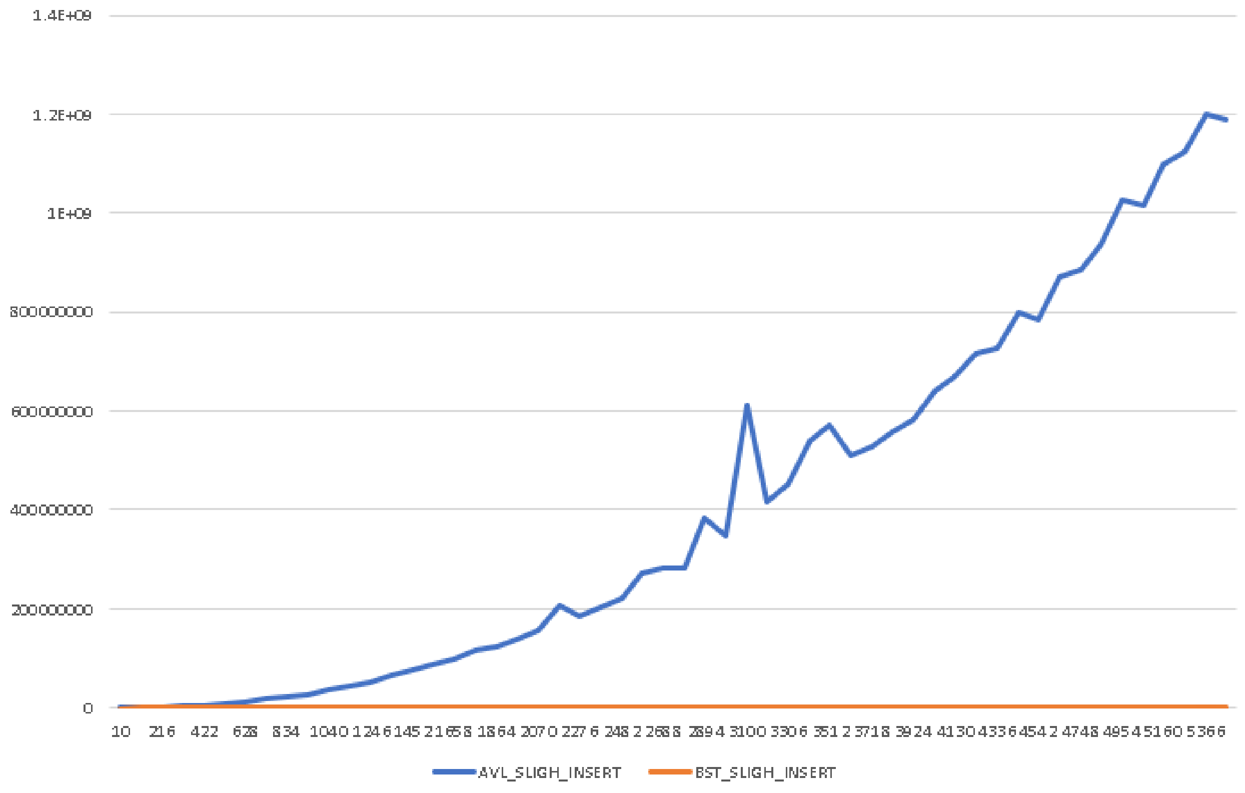


Present and non-present element search time ORDERED

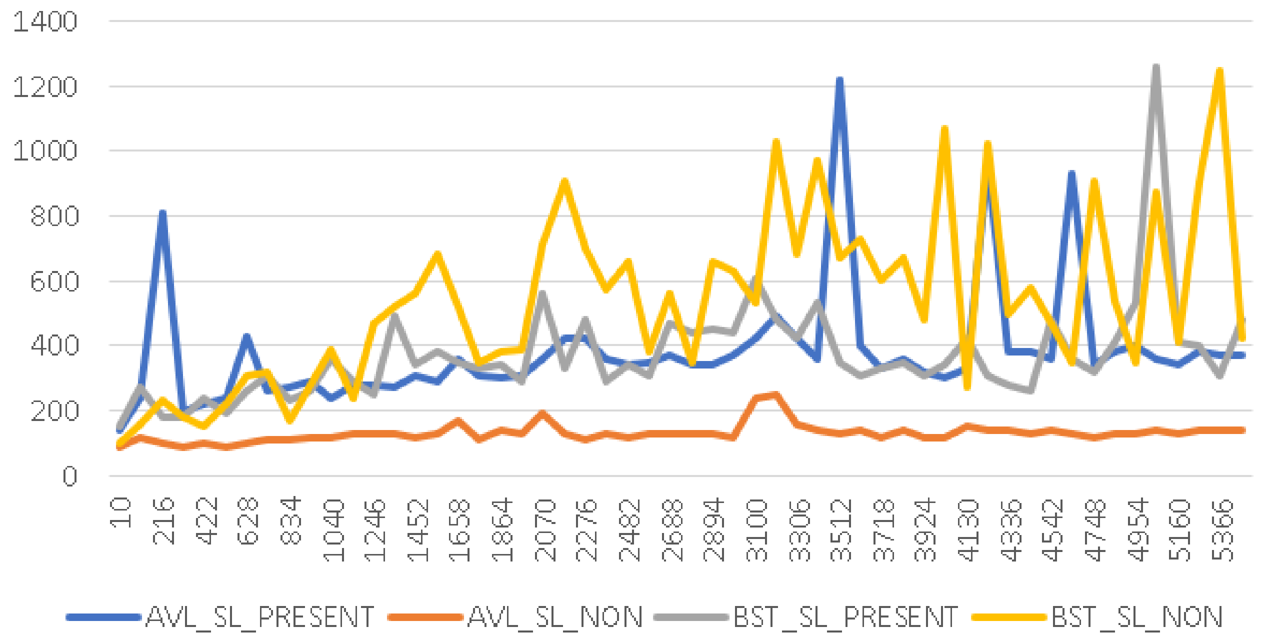


Clearly, in this case, AVL is much better than BST.

AVL vs BST insertion time (SLIGHTLY ORDERED elements):



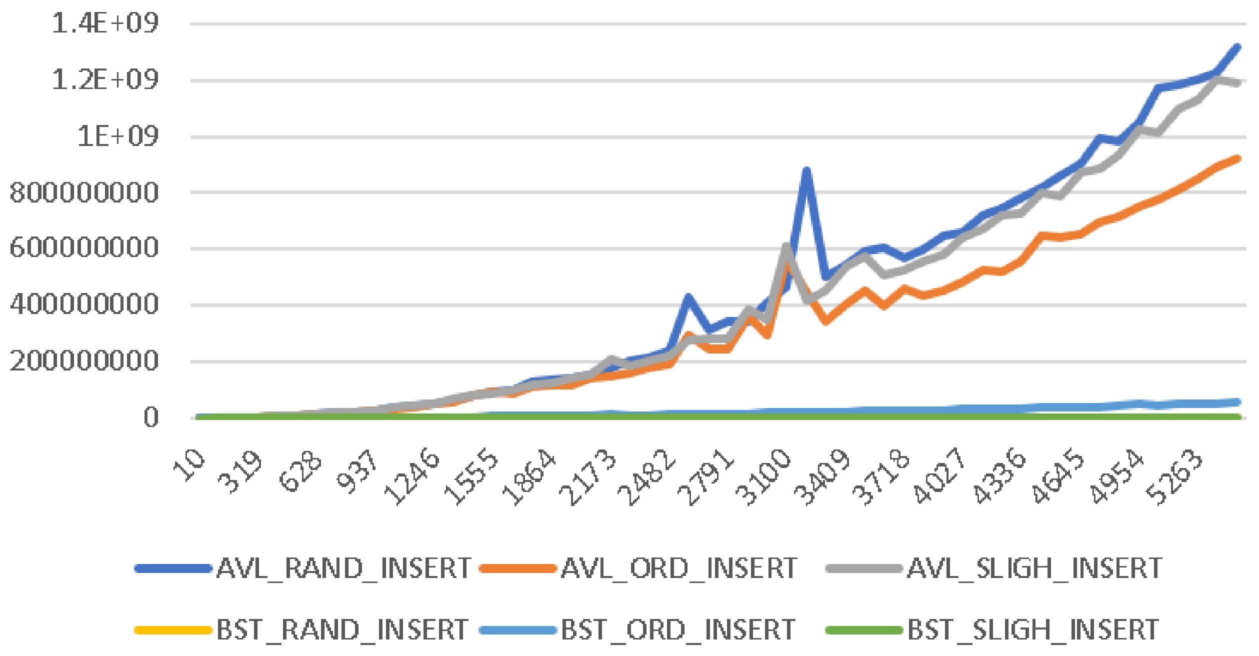
Present and non-present element search time SLIGHTLY ORDERED



Even though the graph has outliers and does not show the clear trend, but what we are interested is which one is better, and obviously we can see that AVL is much better for non-existent elements and AVL is approximately the same as BST when it comes to the present element.

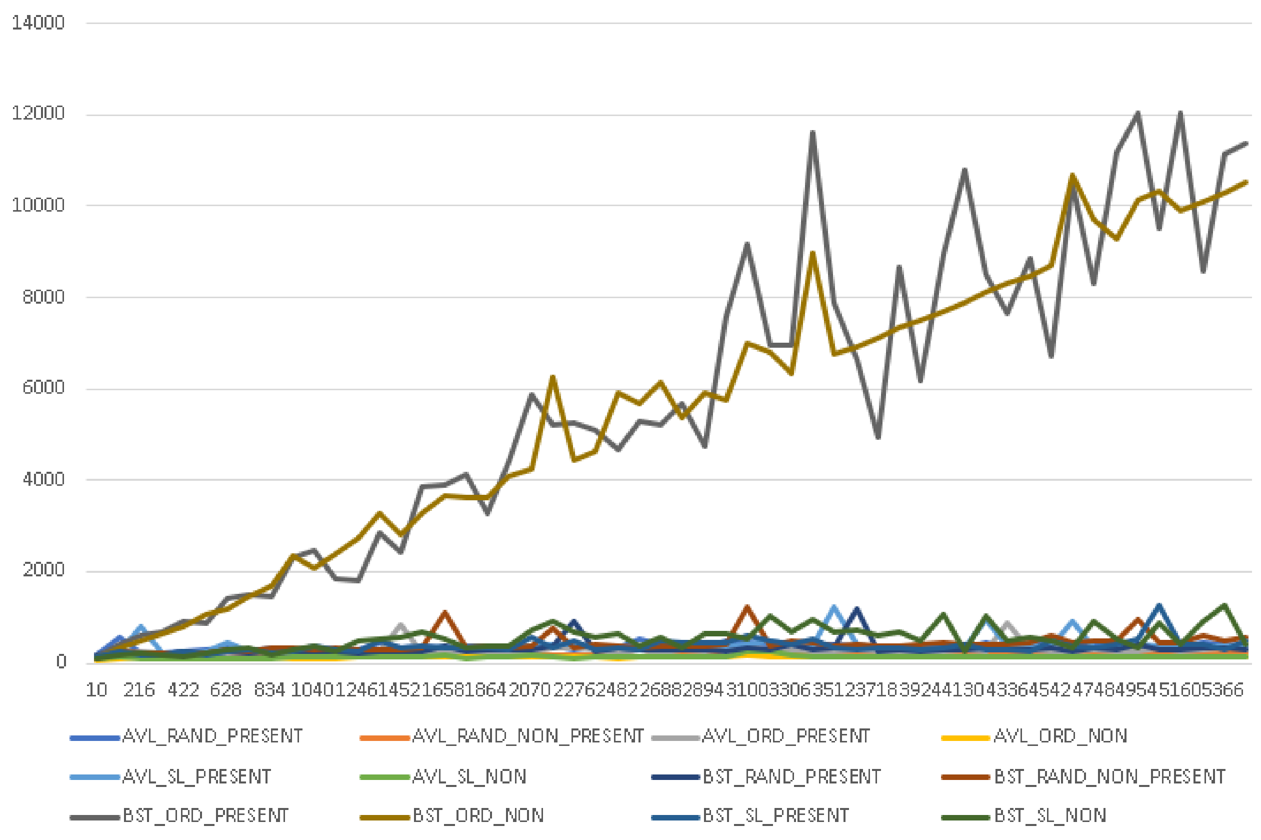
5. Conclusions

Insertion times comparison:

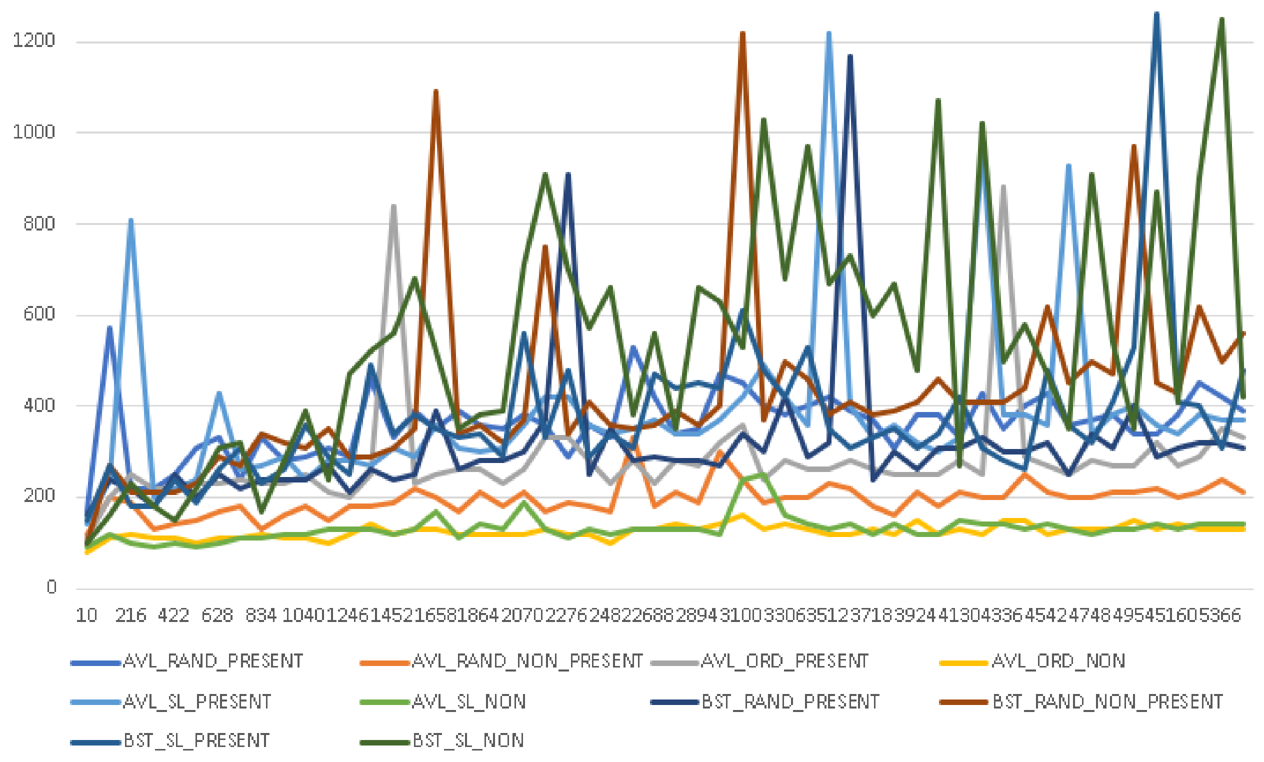


Clearly insertion time for AVL growth much faster than BST.

Search time comparison:



BST is worst for ordered both existing and non-existent elements. Let us take a look at the graph without them. (next page)



Obviously AVL search is best for ordered existent/non-existent elements, slightly ordered existing/nonexistent elements.

In summary, AVL insertion time grows much faster than BST, however search time for most cases is much more time optimized for AVL, while for BST there is substantial probability of facing the worst case.