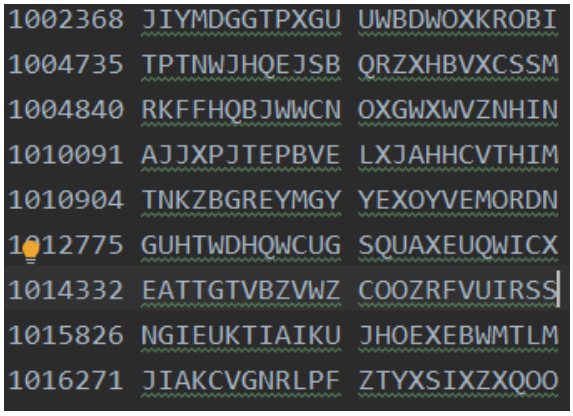
Project report		Academic year:
Subject: **Algorithms and data structures**		**2020/21**
Project name: **Analysis of dynamic lists and binary search trees**		Subject takes place at: **N/A**
Faculty, field of study, semester: **FC, AI, II**	Full name: **Jan Gruszczyński**	Grade:

Project 2

Performance evaluation of add, find and delete operations on ordered dynamic linked list, binary search tree and self-balancing binary search tree.

At each measuring point, time to perform operation was measured 15 times, then an average was taken as representative of obtained values. This was done in order to get more accurate data.

Structure of data:



Data consist of one-row of 7-digit unique indexes, and two rows of corresponding 12-chars names and surnames. Data was generated randomly and saved in data.txt file.

Datapoints used in operation where chosen randomly from the dataset.

Dynamic linked list is implemented in such a way, that it stores datapoints in increasing order (based on the value of indexes).

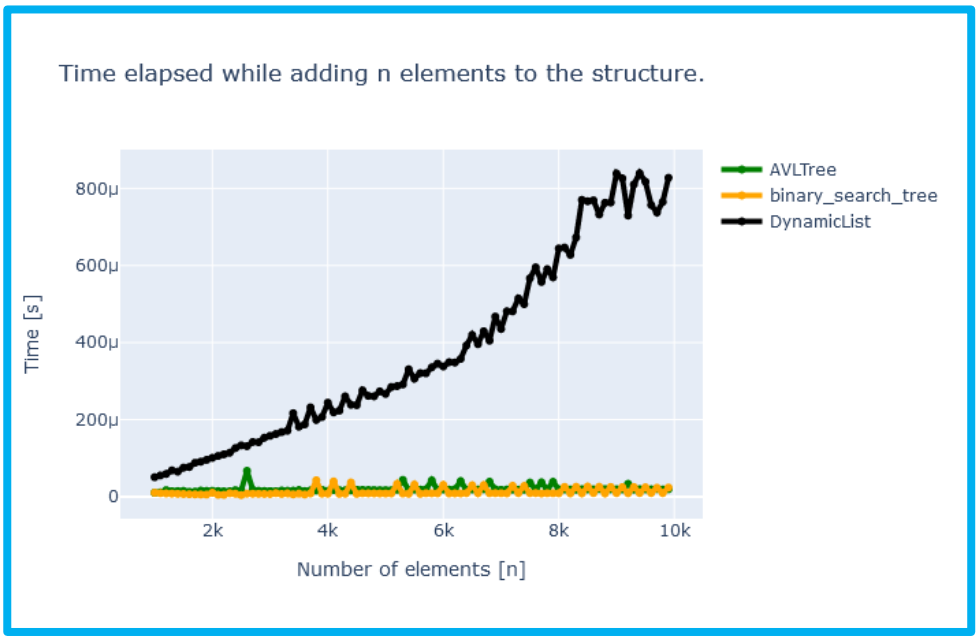


Figure 1: Average time to add element to the structure.

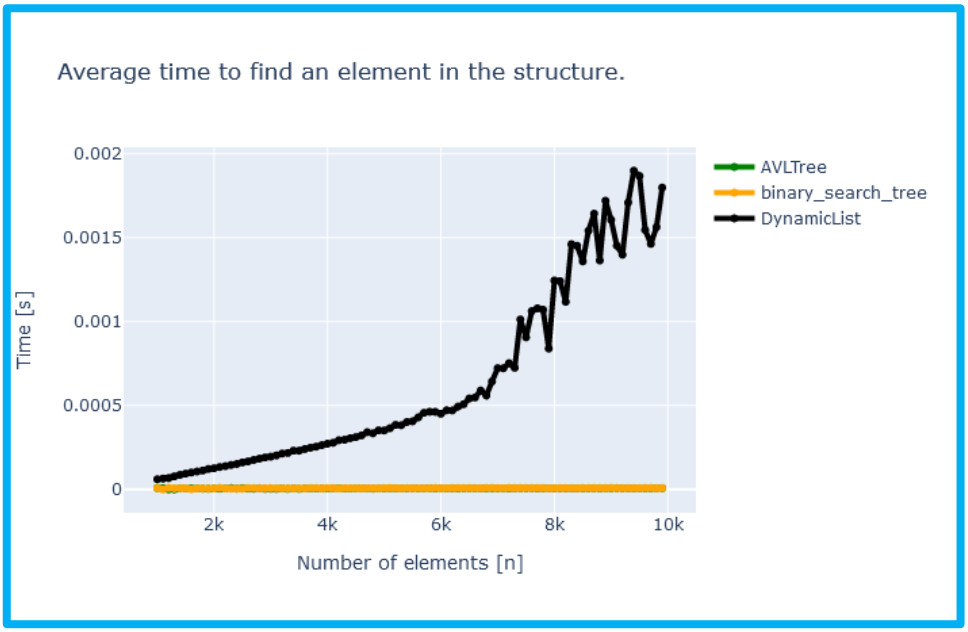


Figure 2: Average time to find random element in the structure.

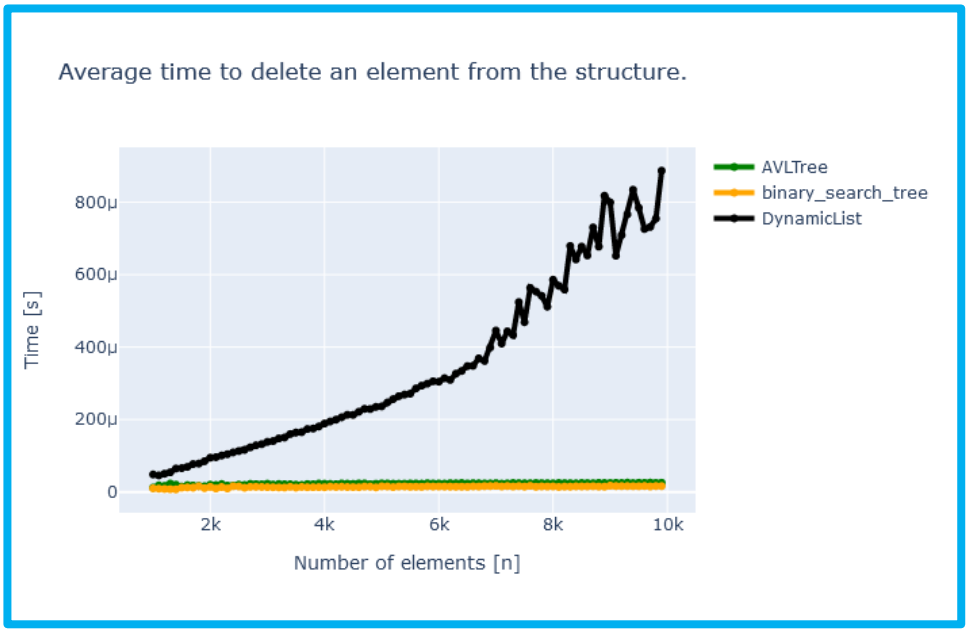


Figure 3: Average time to delete element from the structure.

As expected, dynamic list performs the worst in every case. When, we add new elements to the ordered dynamic list, the time complexity is around $O(n^2)$. On the other hand, time complexity for insertion, search and deletion for binary search trees and self-balancing search trees is on average equal to O(log n).

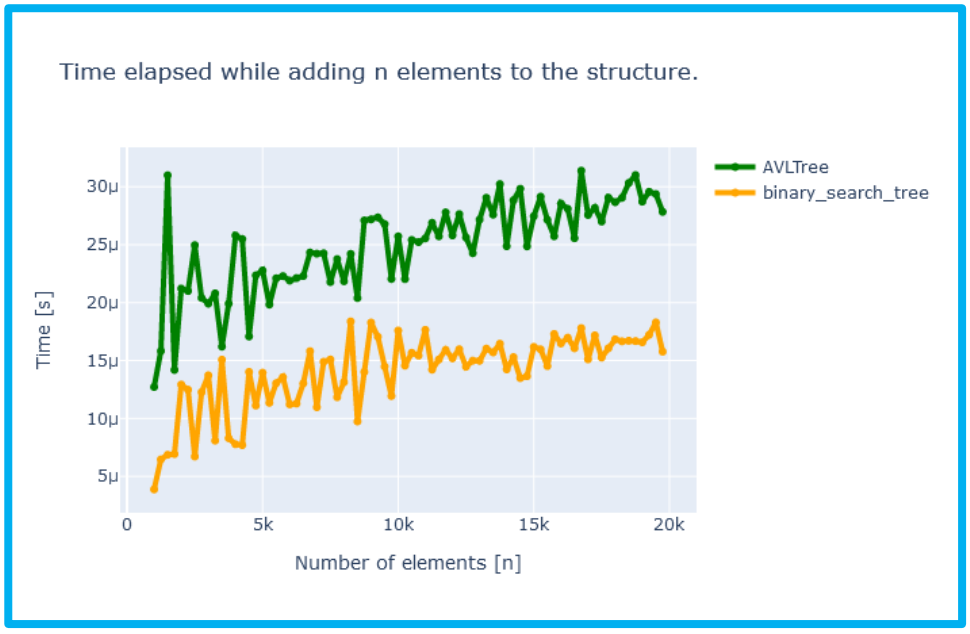


Figure 4: Average time to add element to the structure.

After every insertion and deletion, self-balancing binary search tree needs to rebalance itself (if it became unbalanced), thus it takes more time to perform those operations when compared to standard BSTs.

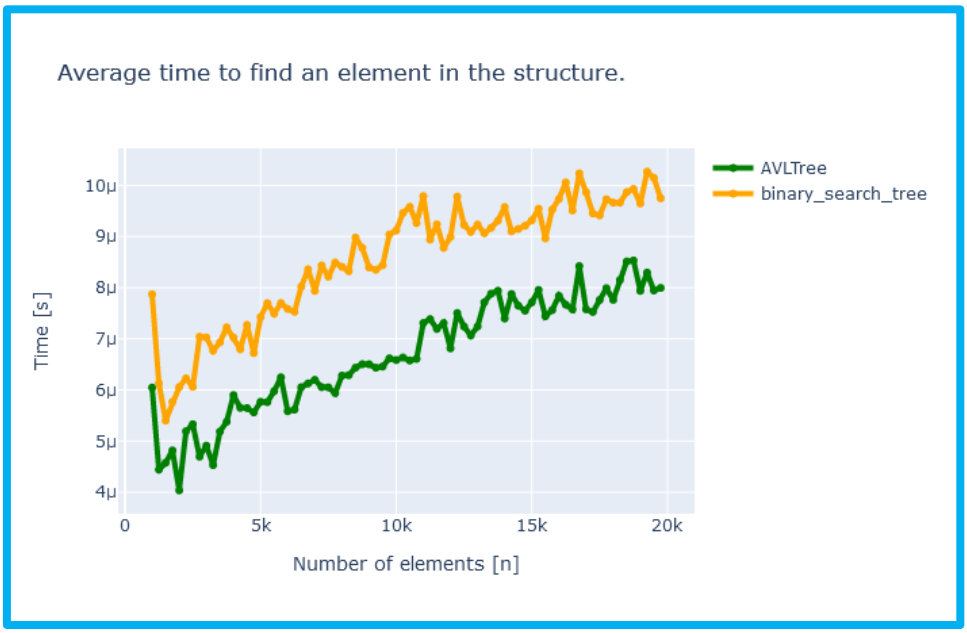


Figure 5: Average time to find random element in the structure.

Search operations on a binary search tree take time directly proportional to the height of the tree. Because self-balancing binary search trees (AVL) automatically keeps it height small (definitely smaller than standard BST), it performs better than normal non-balanced BST.

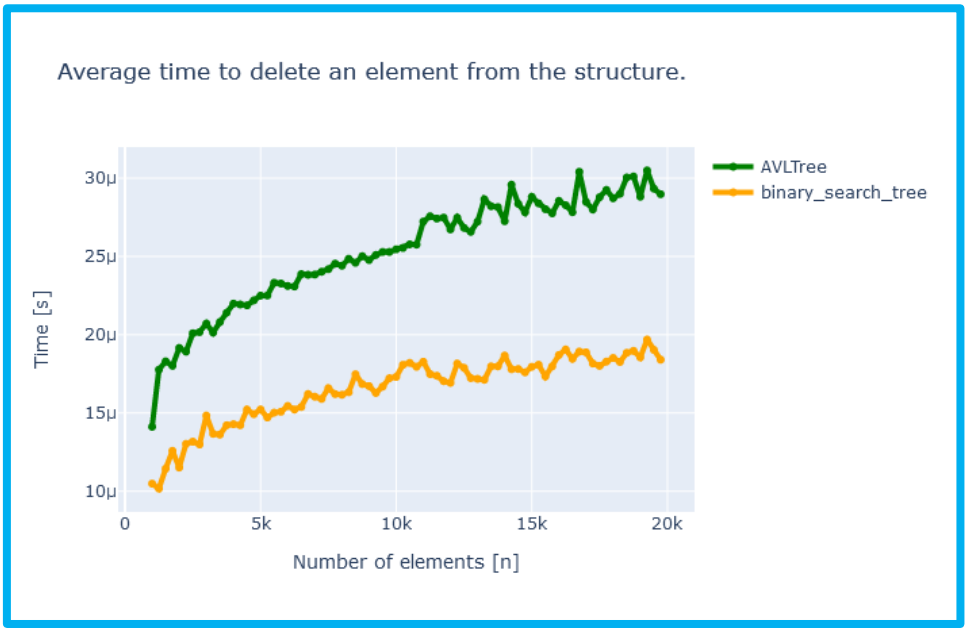


Figure 6: Average time to delete element from the structure.

Similar situation as in figure 4.

Final remarks:

- Ordered dynamic lists are highly ineffective if compared to BSTs. (if it comes to insertion, deletion and search operations)
- As self-balancing binary search trees keep smaller height relative to the number of stored elements, they are more effective when searching elements than standard BSTs.
- What's worth mentioning, the performance gap between AVLs and BSTs, would be much bigger, if inserted datapoints wouldn't be chosen randomly but for example in order.