Comparison of Time Efficiency of Balanced Binary Tree, Ordered List and Unbalanced Binary Tree

Mateusz Tabaszewski 151945

Poznań, April 22

Introduction and Methods of Research

This paper is meant to formalize the conclusions of the experiment which tested the time efficiency of three separate structures. Structures, as well as experiments, have been written using C++ language. Time was measured with help of GetProcessTimes(), GetCurrentProcess() functions included via <processthreadapi.h> header. During the experiments, the program was run for all n elements of the array. This means that that addition was performed in such a way that the structure was built from no initial elements and at the end of the operation the structure was filled with n randomly chosen elements. The same process was used for searching (performing search operation for n randomly chosen elements) and deletion (essentially erasing the whole structure one element at a time). However, the graphs illustrating the time of addition, deletion, and search operations, show the average time of a single operation, meaning that the previously calculated time was divided by n. The exception to it being Figure 3 which shows the total time spent performing search operations for n elements. Each element of data is a structure containing name, surname and index-treated as the only key. Data was generated randomly in C++, shuffled and inserted into a file, from which it was later read for the experiments.

Addition time and deletion time were measured only for Unbalanced BST and Ordered List while search time was measured for all 3 structures. Balanced BST was created by finding the median of values of to-be-added elements and assigning it as root, and then performing the same operation recursively for all sub-trees.

Graphs and Tables

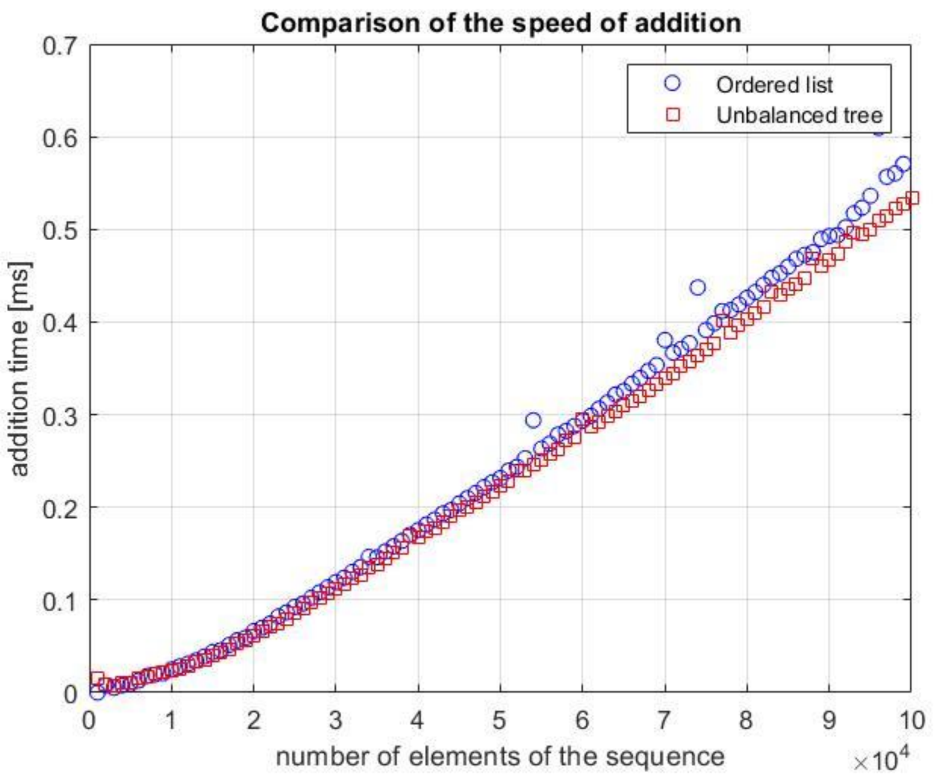

Fig. 1. Average time of addition of one element out of n for BST and Ordered List.

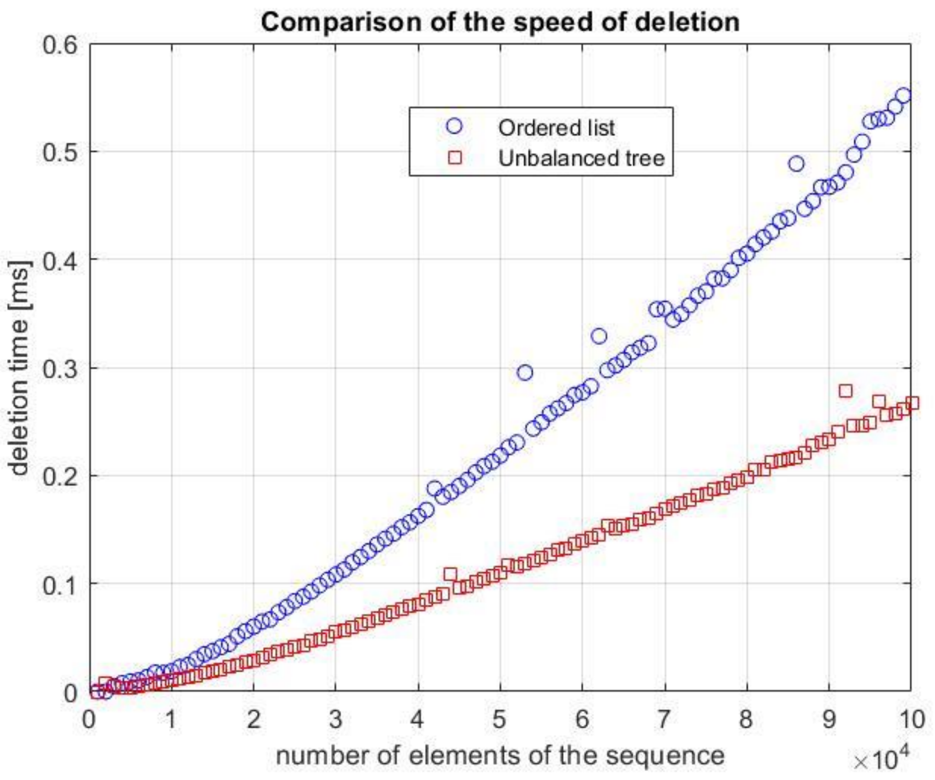

Fig. 2. Average time of deletion of one element out of n for BST and Ordered List.

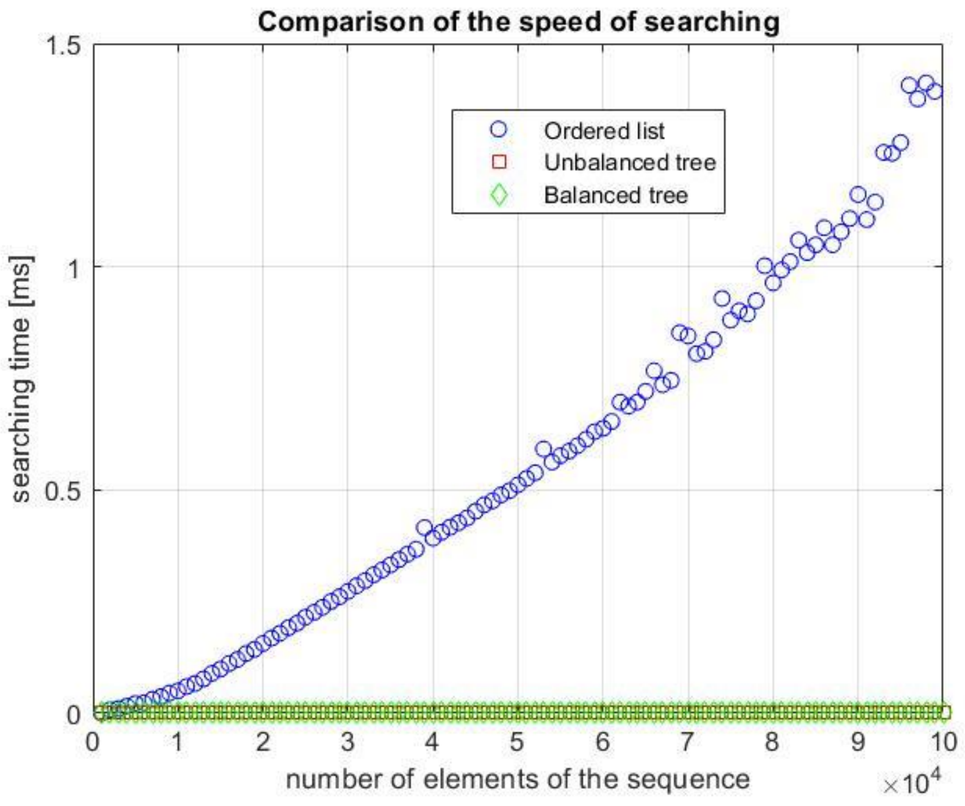

Fig. 3. Total time of searching for all elements in n-sized structures-BST, Ordered List, Balanced Tree.

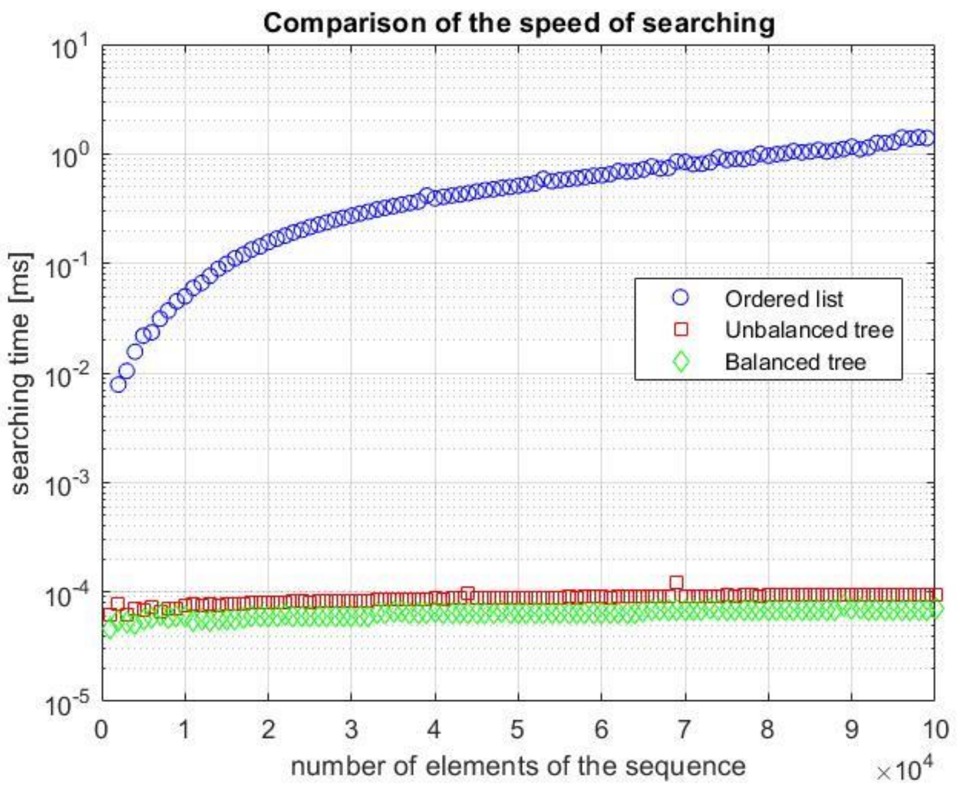

Fig.4. Average search time for a single element in n-sized structures-BST, Ordered List, Balanced Tree presented with semi-logarithmic scale.

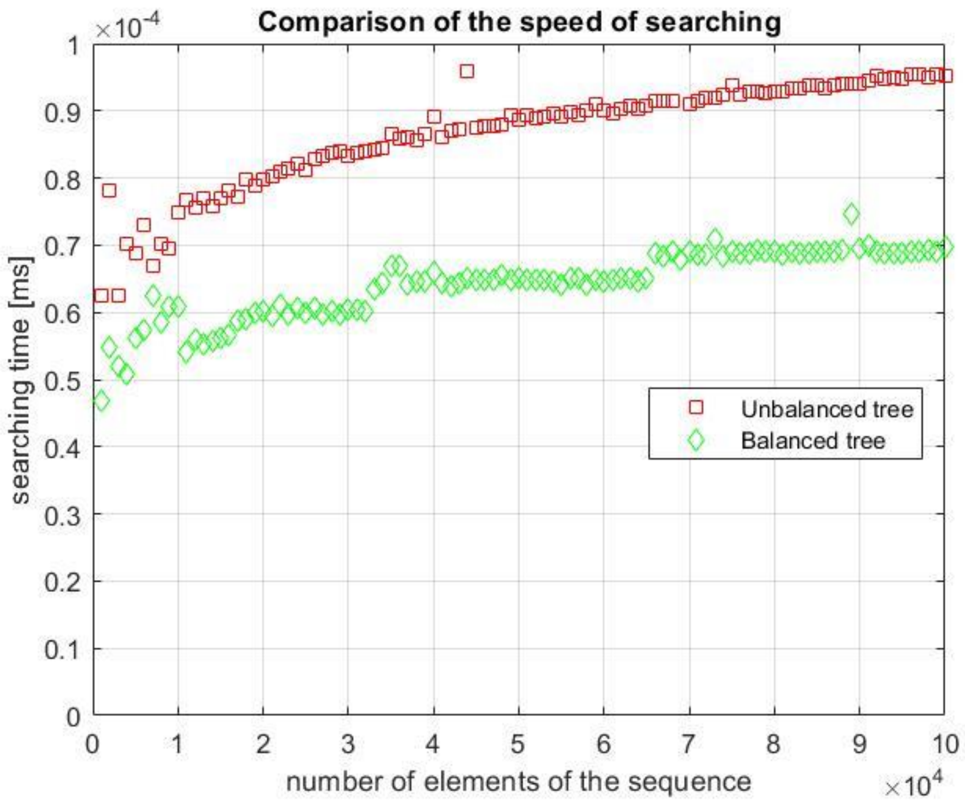

Fig. 5. Comparison of searching time for a single element in n-sized structure between Unbalanced BST and Balanced BST.

Table of Experiment's Results				
	Addition		Deletion	
N	List add. [ns]	U.Tree add [ns]	List del. [ns]	U.Tree del [ns]
1000	0	15625.0	0	0
10000	25000.0	24062.5	18750.0	10937.5
20000	66406.0	61250.0	60156.0	28906.3
30000	119270.7	112500.0	108333.3	54687.3
40000	175390.5	167031.3	162500.0	80859.5
50000	231562.0	223124.0	218438.0	109687.6
60000	293230.0	295833.3	276823.3	138802.0
70000	380580.0	339374.3	354241.4	169197.1
80000	426172.5	403827.5	405273.8	198437.5
90000	492882.2	466840.0	467013.3	233506.7
100000	-	533594.0	-	266719.0

Tab. 1. Comparing average time for addition and deletion of one element into n sized structure.

Table of Experiment's Results			
	Search		
N	List search [ns]	U.Tree search [ns]	Balanced BST search [ns]
1000	0.0	62.5	47.0
10000	50000.0	75.0	61.0
20000	156250.0	79.7	60.2
30000	272916.7	83.3	60.4
40000	392187.5	89.1	66.0
50000	511562.0	88.8	65.0
60000	638020.0	90.1	64.6
70000	845088.6	91.1	69.0
80000	963867.5	93.0	69.0
90000	1161800.0	94.1	69.6
100000	-	95.3	69.8

Tab . 2. Comparison of average time of searching for one element in n sized structure.

*Note: In Both Table 1 and Table 2 there were originally more records however the number has been cut down to only about 1/10 of original number of records to help with readability. The time shown in tables is measured in nanoseconds while time shown on the graphs was measured in milliseconds.

Conclusions

There exists a huge difference between times for the same operations across different structures. Ordered List is the slowest out of presented structures for all operations. Although it's the easiest structure to implement the shortcomings of this method are apparent, especially when comparing the time necessary to perform the search operation (Figure 3 and Figure 4). On top of that, the worst possible case for all operations on the Ordered List for all n elements may require an algorithm to go through all n elements of the structure n times.

The structure that during experiments showed a greater speed of performing tested operations is the Unbalanced Binary Search Tree. The greatest improvement can be seen when comparing the time necessary to perform the search operation. The speed of addition and deletion operations is of course also greater than for Ordered List with addition time also taking longer than deletion, most likely due to the way the algorithm was implemented (addition requires copying the data to the structure, as opposed to deletion requiring only freeing of memory). The biggest negative of this structure is the risk of the worst-case scenario, because there is no rebalancing process, depending on starting data, the tree may end up having a structure comparable to the Ordered List and in turn, the similar time required for performing operations. On top of that even in cases other than worst-case Unbalanced Tree will in most cases not utilize Binary Search Tree's full potential, as it is very unlikely that without any balancing the tree will be constructed in such a way that all data can be found in logarithmic time complexity.

The most optimized structure for performing search operations is the Balanced Binary Search Tree, which is capable of finding an element in logarithmic time complexity while also being completely immune to the worst-case scenario as the Balanced Tree will always take full advantage of the structure that is BST.