EADS LAB 3

BST and AVL Tree

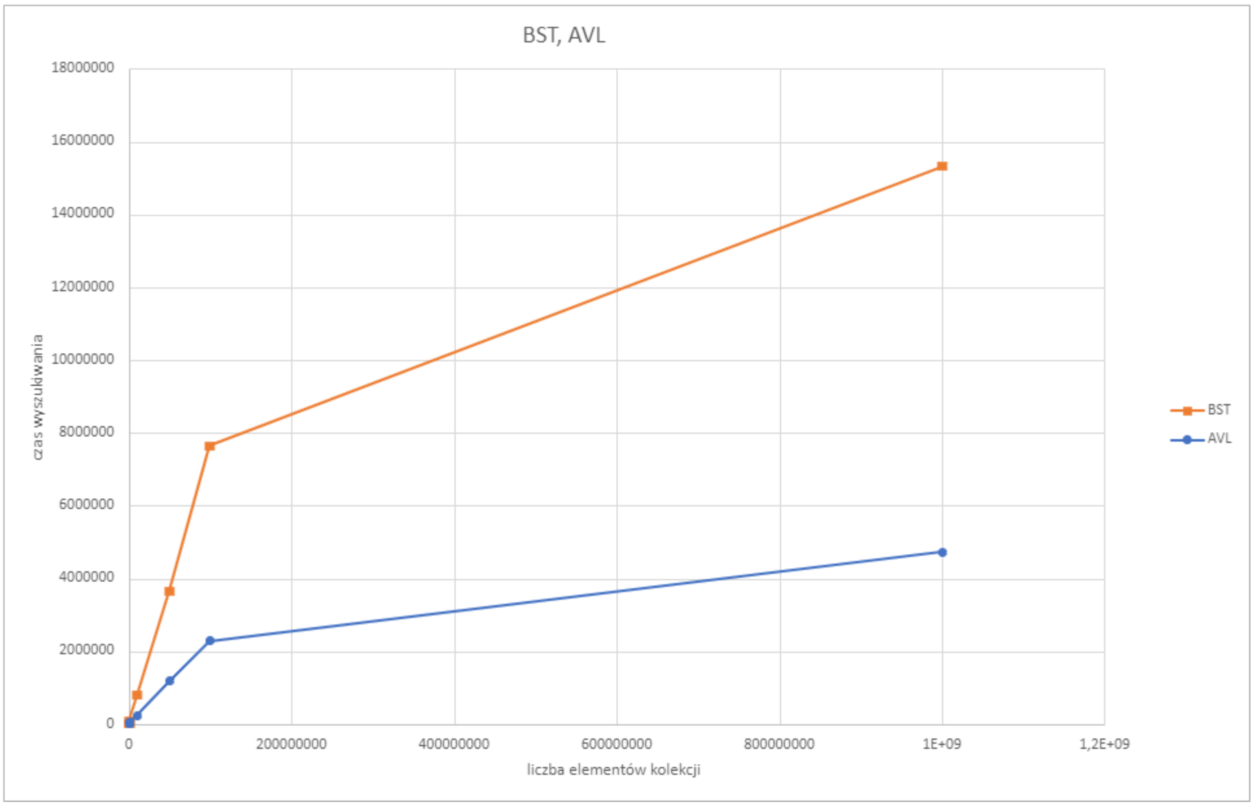
LAKSMAN SIVARAM SENTHILKUMAR
317289
26/01/2022

COMPUTATIONAL COMPLEXITY

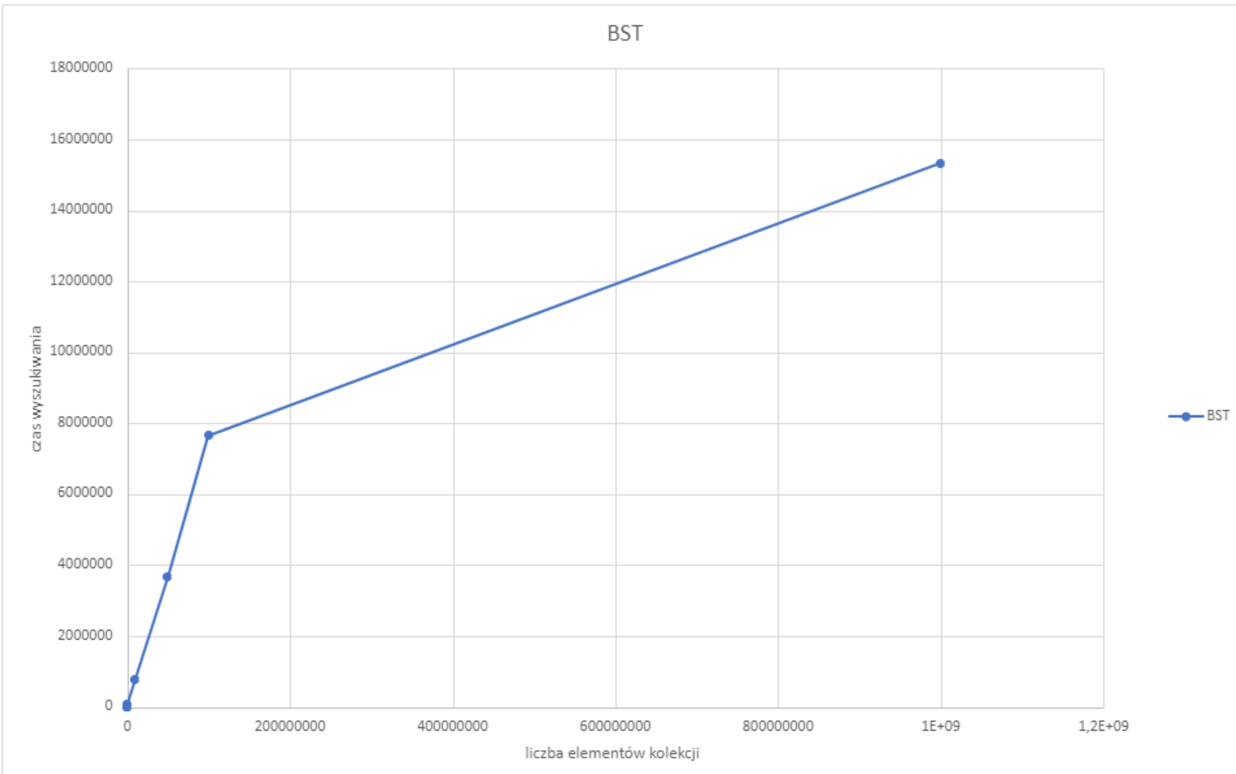
Collection Size	AVL Search Time(ms)	BST Search Time(ms)
1000	9	30
10000	111	308
100000	2000	6430
1000000	37986	96464
10000000	233992	786543
50000000	1190259	3664340
100000000	2294269	7653323
1000000000	4732732	15323875

GRAPHS

AVL vs BST

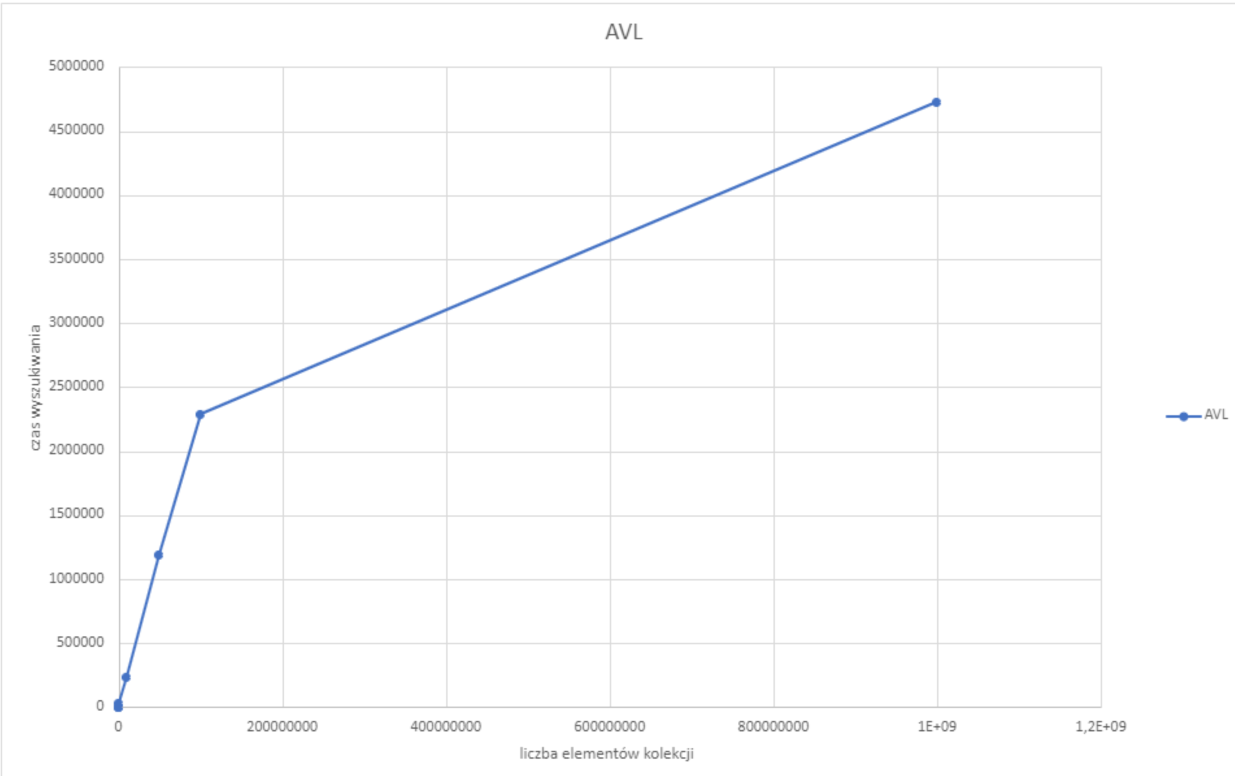


BST



BST
18000000
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
czas wyszukiwania
0
200000000
400000000
600000000
800000000
1E+09
1,2E+09
liczba elementów kolekcji
BST

AVL


AVL
5000000
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
czas wyszukiwania
0
200000000
400000000
600000000
800000000
1E+09
1,2E+09
liczba elementów kolekcji
AVL

CONCLUSION

BST Complexity

Operation	Best Case Complexity	Average Case Complexity	Worst Case Complexity
Search	O(log n)	O(log n)	O(n)
Insertion	O(log n)	O(log n)	O(n)
Deletion	O(log n)	O(log n)	O(n)

AVL Complexity

Insertion	Deletion	Search
O(log n)	O(log n)	O(log n)

AVL tree is a self-balancing binary search tree in which each node maintains extra information called a balance factor whose value is either -1, 0 or +1.

Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree)

From the obtained results, BST is 3 times slower than AVL on average. But then AVL will consume more memory (each node has to remember its balance factor) and each operation can be slower (because we need to maintain the balance factor and sometimes perform rotations).

But for lower collection count, both the tree data structures perform similarly in searching operator, once the collection count starts to increase, AVL benefits from its low height.