CMPT225 homework 2

DLtest.cpp

- *Void displayLinks()* display shape of the tree and also the corresponding address of nodes and its right and left pointers.
- Bst T1 and AVL tree T2 are inserted with the same keys in order and called displaylinks()
- The function works on both BinarySearchTree and AvlTree. We can confirm that the links are correct by matching the address of a node's left/right pointer to its corresponding child's address.

Htest.cpp

- *Int Height()* returns the height of the tree which is the longest path from root to its descendent.
- Both BinarySearchTree and AvlTree tree are inserted with the same numbers and called height().
- The output of height function is correct and is reexamined visually by output the trees with displayTree().

Dtest.cpp

- *Int Depth(comparable &x)* returns the length from node x to root.
- Both BinarySearchTree T1 and AvlTree T2 are inserted with the same keys 1 to 16 in order
- In this test, key 9 is the target key we want to know the depth of. The output depth is right and rechecked it using displayTree().

Exp1.cpp

- Compare two BST of same keys where as T1 is tall and skinny and T2 is short and bushy. T2 2 is inserted with *void bushy(int start, int end,Comparable a[])*. This function allows the program to insert an ordered array first from its median which makes the root of the tree. Then the median of the two subtrees of the root and so on so that every parents is the median of their descendants.
- The graphs below show the <u>average search depth</u> and <u>average search time</u> for existing and non-existing keys range for n=1,000 to n=30,000.
- Based on observations, the data trend of T1's average depth is similar to the trend of T1's search time. T2 exhibits the same phenomenon. This makes sense because the search of the member depends on the height of tree. Considering the wort case, taller tree need more time traversing down to leaf. Because T1 is tall and skinny, its height grows linearly based on number of nodes in T1. For a short bushy tree T2 which is close to shape of perfect binary tree, the height of tree will be upper bounded by O(logn).
- Using the trendline equation in excel, the equation of T1 is linear whereas equation of T2 is logarithmic and much less than T1. As number of node increase, search time needed for tall and skinny tree grows faster than the search time for short and bushy tree. Same applies for the average search depth.

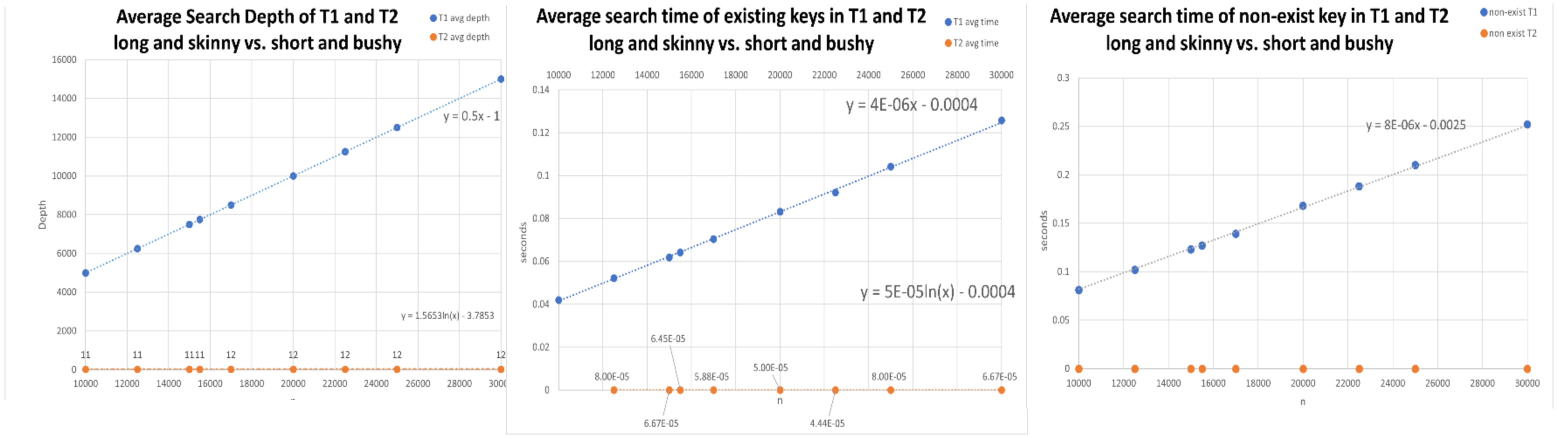
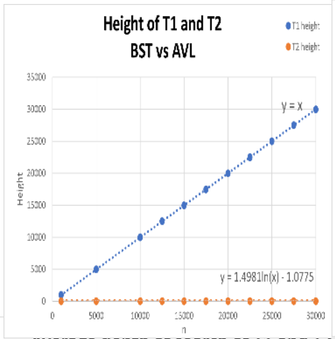
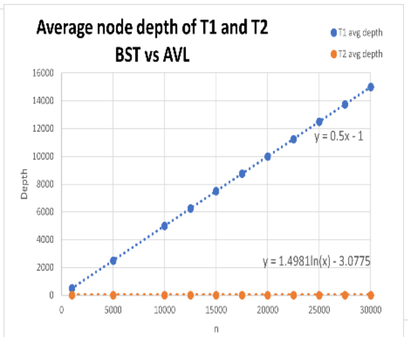


Figure 1: Avg search depth(left), Avg search time of existing keys(middle), avg search time of non-existing keys(right)

Exp2.cpp

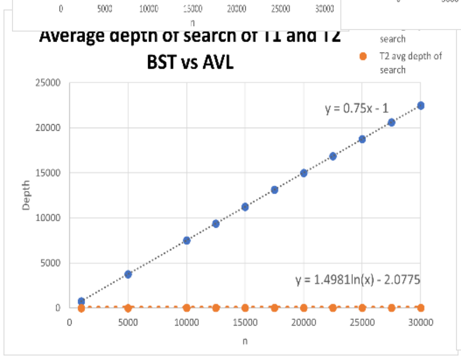
- AVL tree has properties height balance invarinace which allows height to be upper bounded by O(log n). And since the worst case of traversal is equivalent to the height of the tree, the operations are O(log n).
- The graphs below show the average insertion time, height of tree, average depth of tree, average search depth and average search time with respect to n. n ranges for n=1,000 to n=30,000.
- Half the keys used for checking search depth are within the tree and half the keys are not included in the tree, so we take account of the depth when user search for non-existing key.

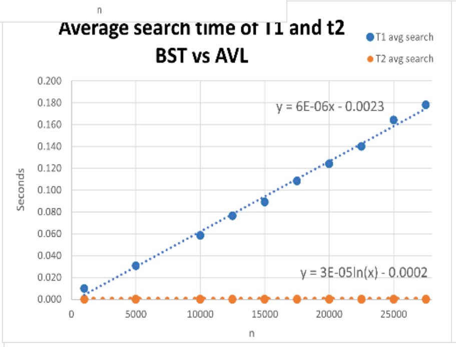


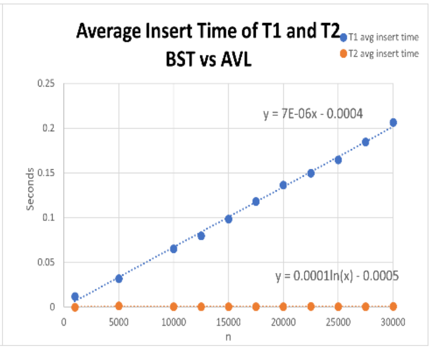


The graph of height shows that the height of the BST tree has a linear growth whereas the height of AVL tree are lower and bounded as n increases. Since AVL tree is height balance invariant, the height is at most 2log$_2$n which is O(log n).

Similar patterns are also observed in average depth of search, average search time and average insert time.







T1 n	min checking time	Max checking time	min search depth	max search depth
1000	0	1	0	999
5000	0	2	0	4999
10000	0	2	0	9999
12500	0	2	0	12499
15000	0	2	0	14999
17500	0	2	0	17499
20000	0	3	0	19999
22500	0	2	0	22499
25000	0	2	0	24999
27500	0	2	0	27499
30000	0	2	0	29999

T2 n	min membership time	Max checking time	min search depth	Max search depth
1000	0	0	0	9
5000	0	1	0	12
10000	0	1	0	13
12500	0	1	0	13
15000	0	1	0	13
17500	0	1	0	14
20000	0	1	0	14
22500	0	1	0	14
25000	0	1	0	14
27500	0	1	0	14
30000	0	1	0	14

Min/MaxTime for Membership check and depth of search for T1(left) and T2(right)

- The above chart show the min/max time for membership check and min/max for depth of seach.
- It shows that the minimum checkig time and depth of search does not differ too much in our slelected range of n this is because the minimum check and search happens at the root which does not depend on the number of nodes.
- However, the maximum membership checking time is about twice as maximum checking time for AVL tree. And maximum depth of search has consideringly great increase as n increases.

By change the varuable NUMS in these test can change the number of keys in the tree for testing purpose.