Samuel Fu, Xingxing Geng
Professor Divy Agrawal
CS130A
3/16/18

Method	BST	Worst case time complexity of BST	Hash Table	Worst case time complexity of HT	AVL	Worst case time complexity of AVL	2-5	Worst case time complexity of AVL
Search	9*10^(-4)	O(n)	2.8*10^(-3)	O(n)	1.7*10^(-5)	O(logn)	2*10^(-5)	O(logn)
Insert	1.7*10^(-3)	O(n)	3*10^(-4)	O(n)	2.1*10^(-5)	O(logn)	5*10^(-5)	O(logn)
Delete	1.7*10^(-13)	O(n)	7*10^(-6)	O(n)	2.1*10^(-5)	O(logn)	4*10^(-5)	O(logn)
Sort	0.41165	O(n)	1.473	O(nlogn)	0.00284	O(n)	0.007574	O(n)
Range query (n=10)	7*10^(-5)	O(n)	0.007334	O(n)	7*10^(-6)	O(n)	1.22*10^(-4)	O(n)
Range query (n=100)	0.000567	O(n)	0.00744	O(n)	2.4*10^(-5)	O(n)	1.9*10^(-4)	O(n)
Range query (n=1000)	0.004753	O(n)	0.005233	O(n)	9*10^(-4)	O(n)	2*10^(-4)	O(n)

Figure 1: BST vs. Hash Table vs. AVL vs. 2-5

For BST, the worst case time complexity always converges to O(n), as the worst shape of a BST looks similar to a LinkedList, and therefore you cannot perform a binary search but only traverse through every node. Even though Hash Table does not have concerns as of its shape, it does have another major setback—collisions. Collisions caused by linear probing are difficult to keep track of and therefore it may require an algorithm to traverse through the whole array in the worse case, which is O(n) for all methods but *Sort*. Hash Table's *Sort* is bounded under O(nlogn) as it requires O(n) time complexity to insert each word in a Vector<string>, and O(nlogn) time complexity to perform *std::sort*. The AVL Tree and the 2-5 both outperformed the previous two data structures due to their special property of "self balancing." Therefore, the time complexity of these two are influenced more by the height of the tree than number of the nodes. On top of that, the AVL avoid the situation of having a "Linked List" that BST may face in worst cases because it is also self-height balanced.

To put into a nutshell, AVL Tree and 2-5 Tree have better performances on each methods because of their unique balancing property. It even becomes more superior as number of data grows, the running times of both BST and Hash Table make less differences as the complexity relates heavily on the height of the tree for BST and number of keys for Hash Table. Nonetheless, the AVL tree and 2-5 still provide with efficient performances.