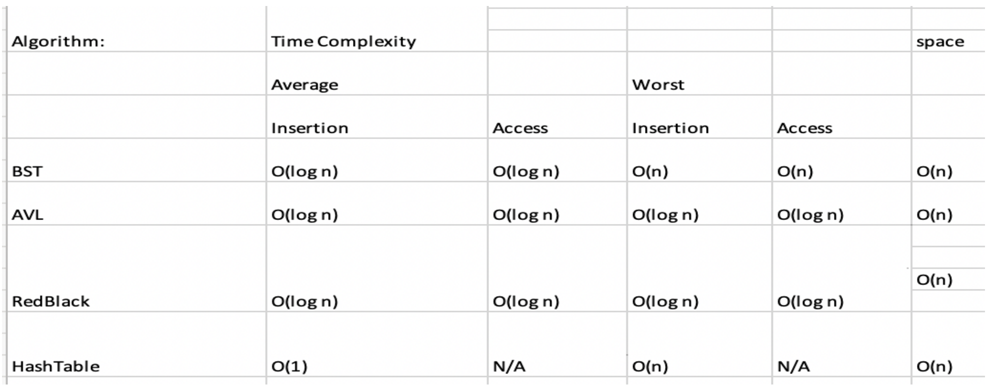
# Assignment 4 Report

Chang Yan (Charlotte)

May 31 2019



| Algorithm: | Time Complexity | | | | space |
|---|---|---|---|---|---|
| | Average | | Worst | | |
| | Insertion | Access | Insertion | Access | |
| BST | O(log n) | O(log n) | O(n) | O(n) | O(n) |
| AVL | O(log n) | O(log n) | O(log n) | O(log n) | O(n) |
| RedBlack | O(log n) | O(log n) | O(log n) | O(log n) | O(n) |
| HashTable | O(1) | N/A | O(n) | N/A | O(n) |

Run Times for BST, AVL, Red-Black and Hash Table

## 1 Sorted Insert

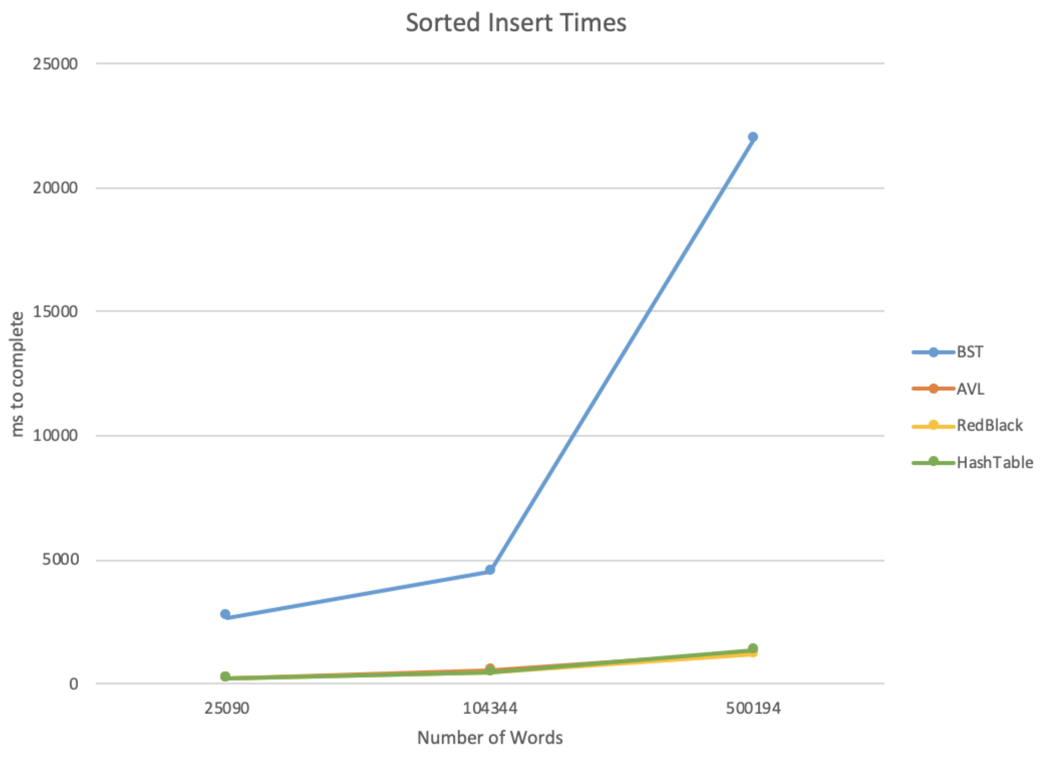


Figure 1: Sorted Insert Times

Figure 1 shows the insert times for a sorted text. We can see that the insert times for a BST takes the longest time. This is because the worst case for insertion in a BST is $O(n)$. If we want to insert a sorted list of values, it chains them into a linked list with no left subtrees. For example, put([1, 2, 3, 4, 5]) yields the tree (1 (2 (3 (4 (5))))). The worst case time complexity of insertion for BST can be improved by using a self-balancing binary search tree like Red Black Tree, and AVL Tree. Using self-balancing binary tree Tree will take $O(logn)$ time to insert data in worst case. The run time for insertion in a Hash Table is constant. According to Figure 1, AVL tree, Red Black tree, and Hash Table spent almost equal amount of time. Hence, these three data structures are more desirable when inserting a list of sorted data comparing to BST.

## 2 Sorted Print

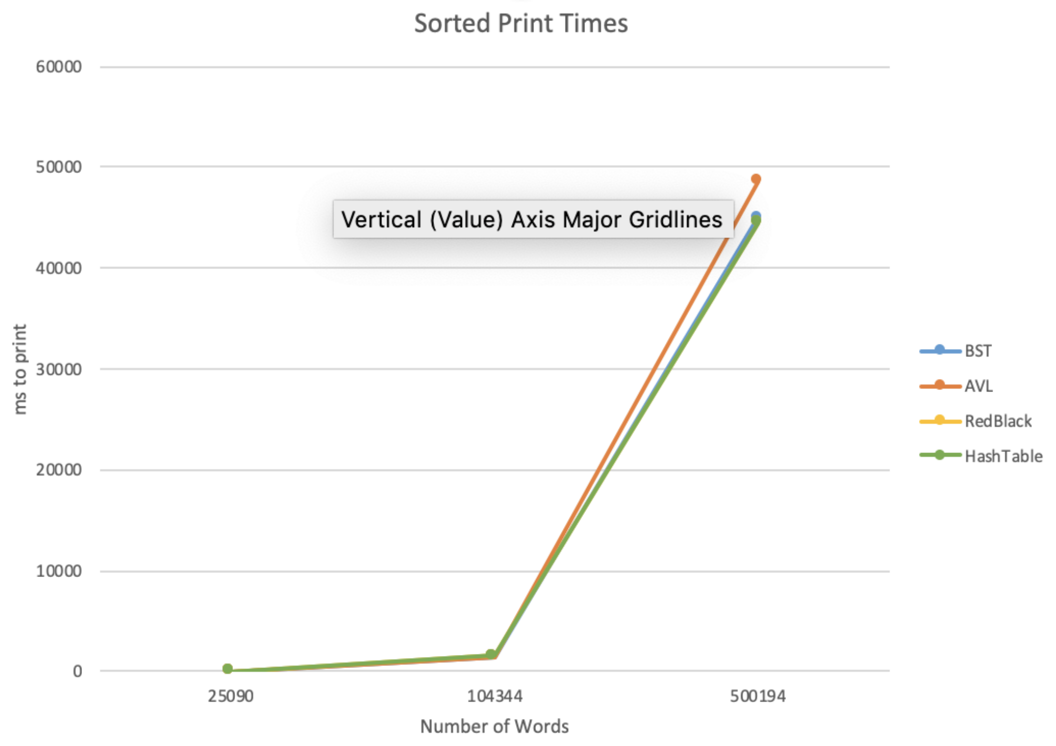


Figure 2: Sorted Print Times

Figure 2 shows the print times for a sorted text. We can see that the print times for BST, AVL tree, Red Black tree and Hash Table take almost the same time. This is because the toString() method for BST, AVL tree and Red Black tree used recursion, which takes $O(n)$ time. In toString() method, we visited every node once and do one amount of work. If we denote this amount of work by $a$, the calculation for run time will be $O(a \cdot n)$. Because we ignore constant factors, a simpler answer would be $O(n)$. Even in a worst case time scenario, the complexity is still linear. Only the constant is higher. The run time for toString() method in Hash Table is also linear, because we iterate the whole array in toString() method. Hence, the run time for BST, AVL tree, Red Black tree and Hash Table are all linear. Therefore, their print times do not have a big difference.

## 3 Standard English Text Insert

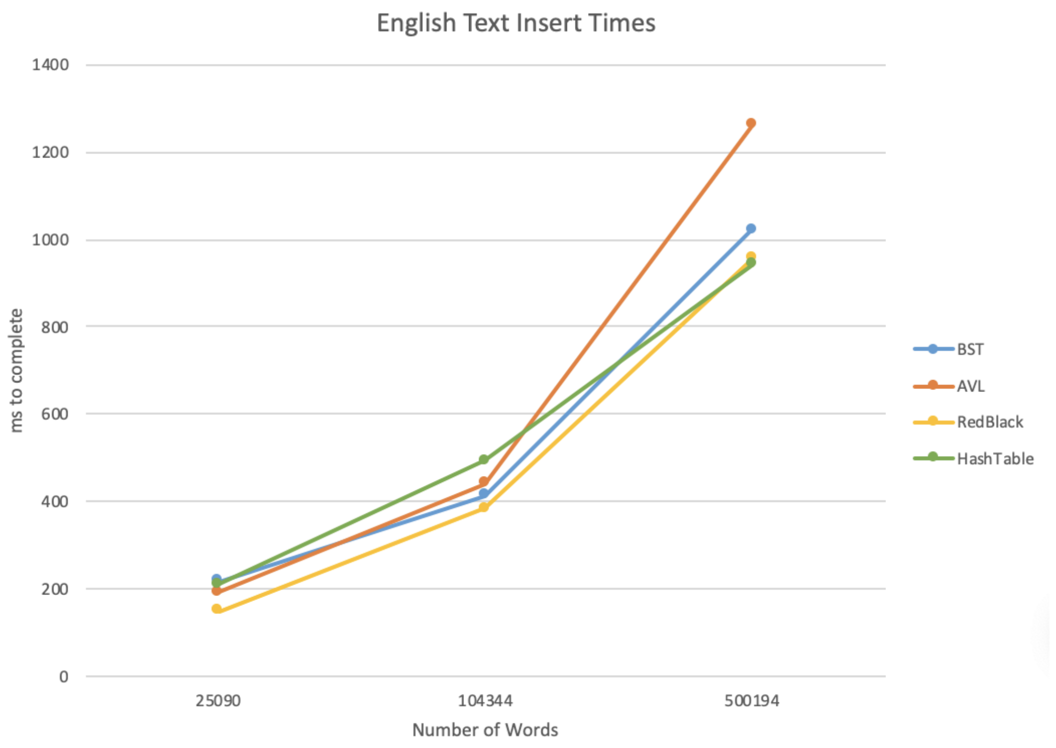


Figure 3: English Text Insert Times

Figure 3 shows the insert times for a standard English text. According the Figure 3, we can see that the insertion performances are almost identical for BST, AVL tree, Red Black tree and Hash Table. Notice that the insert times are linearly incremented by the word size of the file. In Figure 1, we have showed the worst case scenario for BST is the insertion for a list of sorted data. Now, we are inserting a list of unsorted English words. This means we have the average insertion case for BST, which is $O(log(n))$. Hence, in this case, the insertion performances for these four data structures are performing almost identically.

## 4 Standard English Text Print

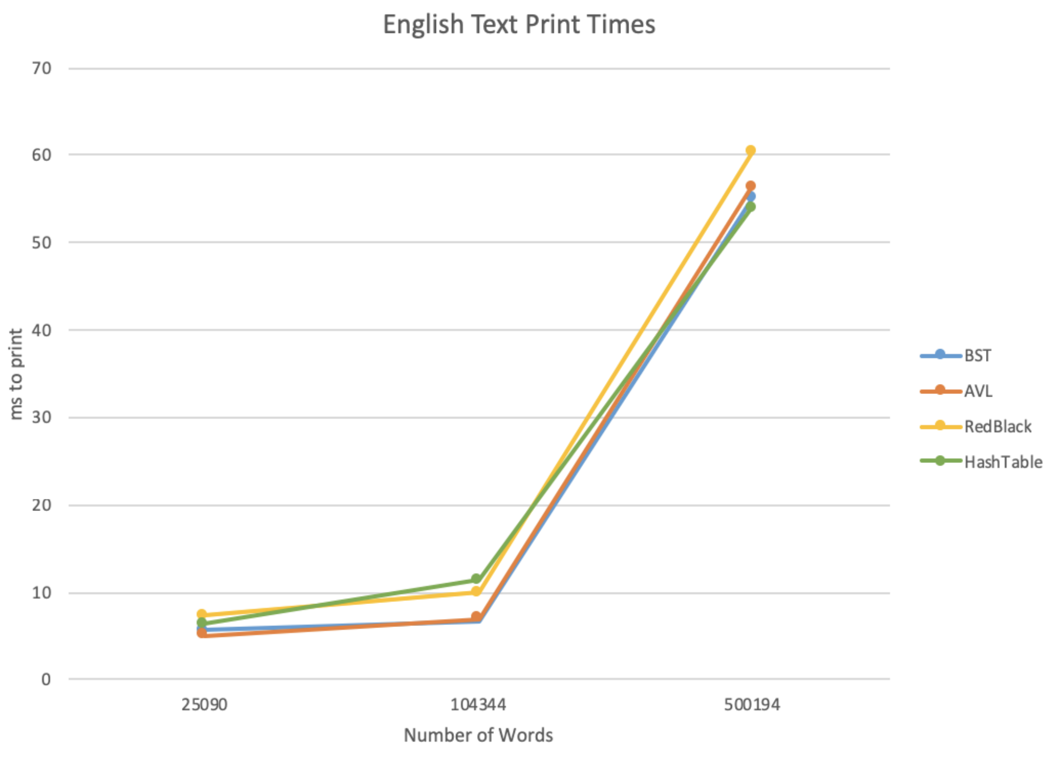


Figure 4: English Text Print Times

Figure 4 shows the print times for a standard English text. According the Figure 4, we can see that the print performances are almost identical for BST, AVL tree, Red Black tree and Hash Table. The print times are linearly incremented by the word size of the file as well. There is no big difference because the run time for printing in BST, AVL tree and Red Black tree are all $O(n)$. This is because the use of recursion, inorder tree traversal, and a single StringBuilder instance in my toString() method. In Hash Table I also linearly iterated my whole table which causes a linear run time as well.

## 5 Conclusion

In conclusion, BST is easy to implement compared to Hash Table, we can easily implement our own customized BST. With Self-Balancing BST like AVT tree and Red Black tree, all operations are guaranteed to work in $O(log(n))$ time. But with Hashing, $O(1)$ is average time and some particular operations may be costly, especially when table resizing happens.

## References

[1] https://www.geeksforgeeks.org/advantages-of-bst-over-hash-table/