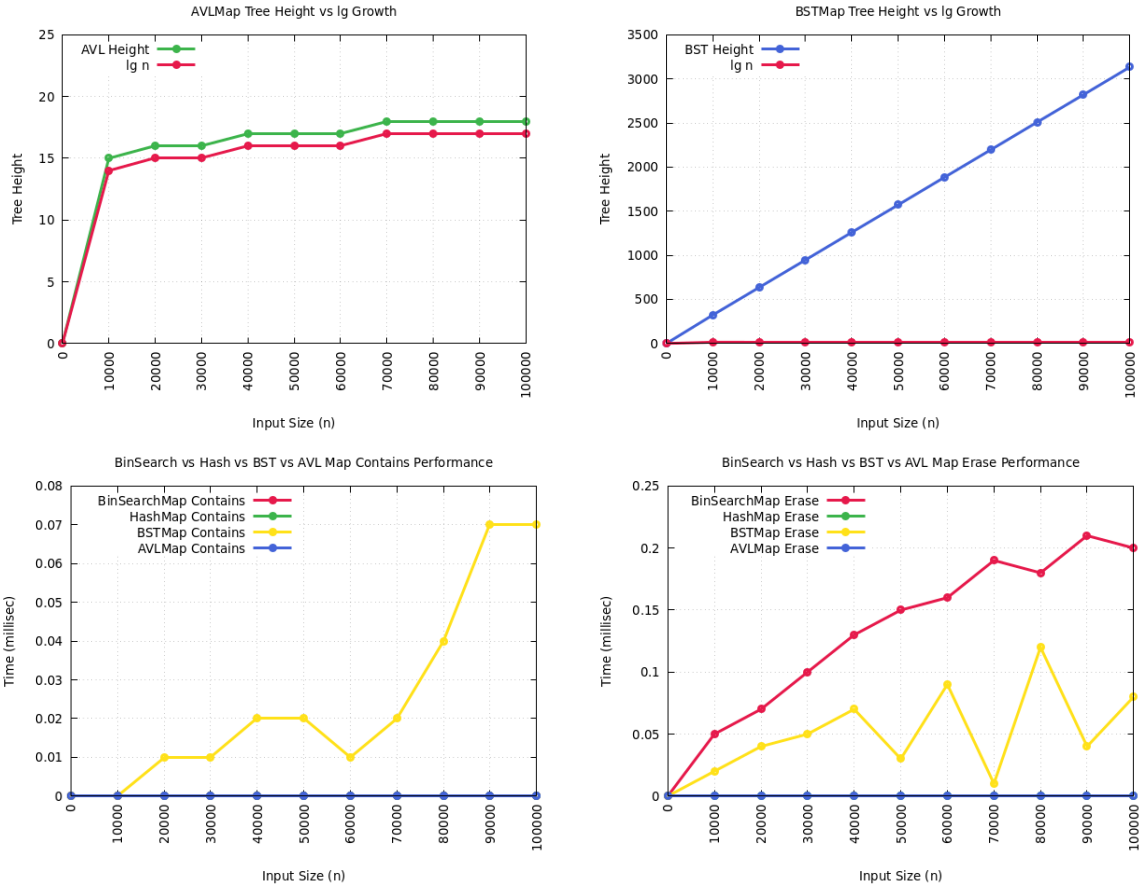
HW9 Writeup (Spring 2022)

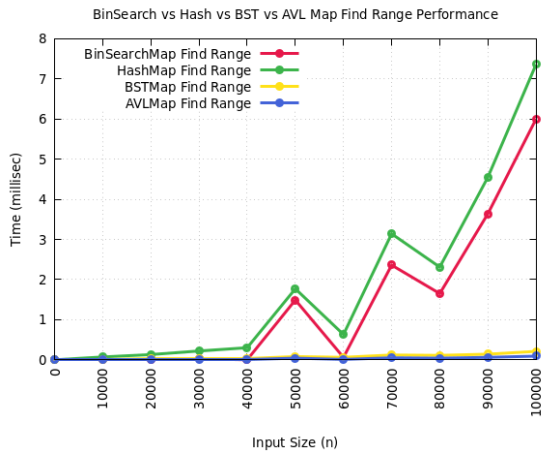
Plots:

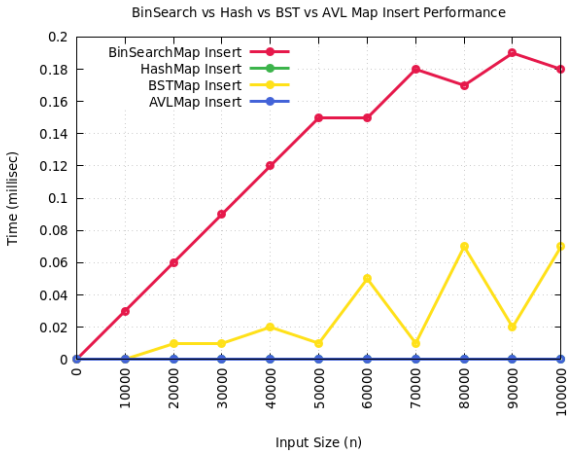


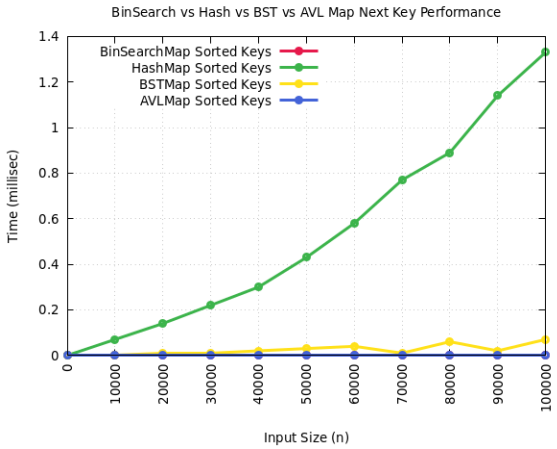
The AVLMap Height performance shows a similar trend to that of lg growth. The AVL height function does seem to return heights 1 level greater than the lg growth. As my height function in the AVLMap implementation simply returns to current height of the tree, it would stand to reason that the rebalance mechanism does a height change that adds 1 onto the end of the height. However, the AVLMap height maintenance is far better than in the BSTMap height. This can be attributed to the added rebalance method mentioned above.

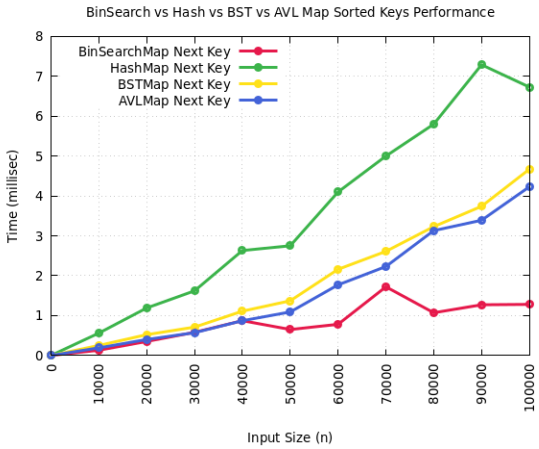
The contains comparison graphs appears to only show a stark comparison between BSTMap and AVLMap. As AVLMap can travel shorter paths to find a node due to rebalancing on insert and erase, it makes sense that AVLMap contains would perform better.

Erase in AVLMap appears to be constant in time complexity when compared to other map implementations. Erase in AVLMap utilizes recursion to navigate to the node that will be removed. Then it calls rebalance which is constant time. Thus, erase really has a complexity of O(log(n)) whereas BSTMap has an erase with O(n) time complexity.









The find range method of AVLMap, while identical to BSTMap in implementation, has a quicker time complexity due to keeping a balanced tree on hand. Thus, searching for keys within a range searches root-to-leaf paths with less depth than in BSTMap.

The insert method of AVLMap is also O(log(n)) due to the recursive calls to insert and the constant rebalance on the current root node. The graph is correct in depicting BSTMap as more complex since insert is also O(n) for BSTMap.

Next and previous key for AVLMap appear quicker than BSTMap for the same reason as with insert, a tree with smaller heights in its subtrees will produce quicker searching for nodes. So even though the implementation remains the same, BSTMap is more complex in time to find the in-order successor of the node requested.

Finally, the sorted keys performance comparison graph remains in line with the other graphs in showing AVLMap as quicker than BSTMap. However, BinSearchMap still holds the lowest time complexity as a sorted list in maintained throughout all methods used in that data structure.

Challenges:

Implementing insert, erase, rebalance, and the rotation helpers was less challenging than I thought it would be. But my attempts to fix memory leaks proved to replace that success with distress and difficulty. I still have not found where I could be having memory leaks. From my search for answers, the copy constructor check, erase check, and erase rebalance check are tests where there seem to be memory leaks. This leads me to believe it may be an error with erase and/or rebalance (it could totally be insert too as one cannot very well erase unless key-value pairs have been inserted first).