2XB3 Lab 6

Casey Doede – 001223493 – doedecd@mcmaster.ca – L01

Amaan Ahmad Khan – 400230523 – khana251@mcmaster.ca – L01

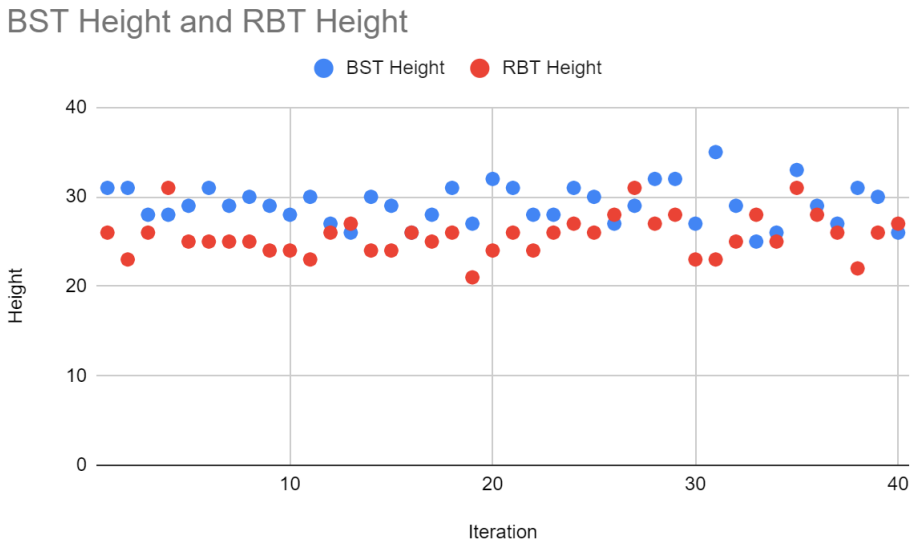
Alex Axenti – 400268807 – axentia@mcmaster.ca - L04

Implementation of RBTree:

When inserting the numbers 1 to 10000 in ascending order the height of a red black tree should remain constant no matter how many times one calls this creation loop. This shrinkage in the Tree height after the third call is due to a bug in the insert method code. The insert method checks if the value to be inserted is less than the node value and then only has an else statement to insert the value. Thus if the value is the same as the node value it will still be inserted. This violates one of the principal properties of binary trees.

One can fix this error by changing the else statement to an elif to check that the value to insert is greater than the node value. Our thought process behind the reduction in height by the addition of multiple duplicates is that after a certain number of duplicates are added the tree may have to perform a rebalancing to reduce the overall height.

Average Height Comparison:

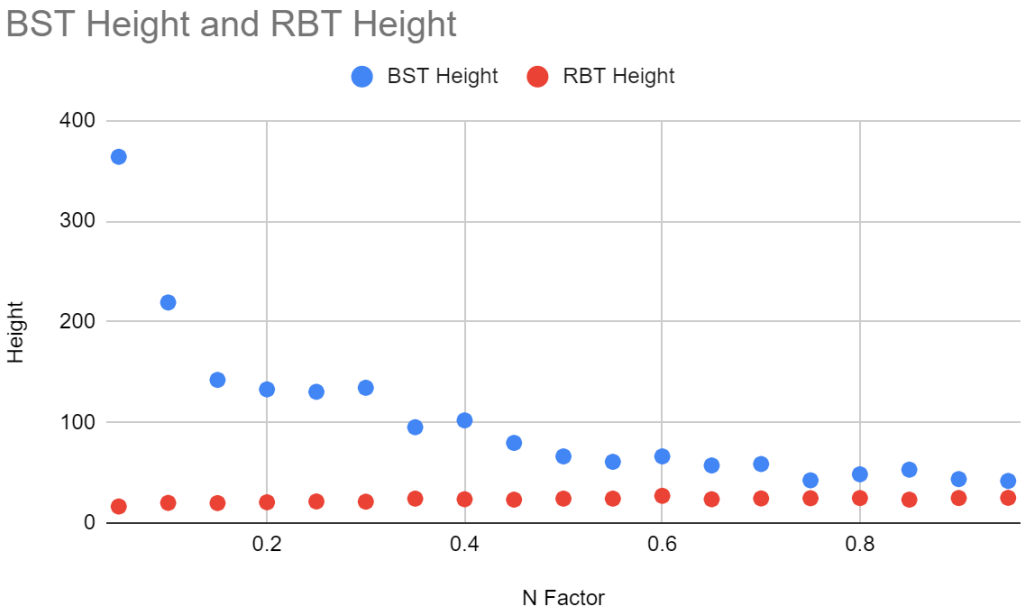


The exact average height difference is 3.475. BSTs have very poor performance when the list is near sorted (as will be seen later in the report), while having similar performance to a RBT when the list is randomized. Therefore, cases where an BST would be better would be when the inputted list is completely randomized. This is because the height would be practically the same as for a RBT, but constructing the tree would require a much smaller number of comparisons, resulting in faster operations.

The implementation of sort is where the performance will be marginally different. At first looking at the code for the two implementations one would assume that as the code is identical but BST is a more simple method that it would have improved performance, but due to the nature of the lack of balancing

it can suffer due to this fact. As each added level of depth reduces performance. Though both should be in the realm of O(logn) optimally.

Near Sorted Comparison:



Clearly the height of a BST is extremely large when the inputted list in nearly sorted. This is because every element is continuously added down and to the right of the tree, without any rebalancing occurring. Although as the inputted list became less sorted, the BST approached similar values to the RBT. This connects to the previous question, by showing that RBTs are much better for near sorted lists, but the performance for randomized lists is very similar and thus not worth the added performance hit required for the balancing of red black trees.