CS590 – Algorithms Assignment 3

Name: Sushant Bhat

CWID: 10474365

Report

Abstract:

This report discusses and provides findings various operations performed on Binary Search Trees and Red Black Trees. We are given an array of n integers and we sort them into an empty BST and RBT using a modified in-order-tree-walk algorithm. We

Algorithms:

We discuss the following algorithm:

1. **BST Insertion:**
   Binary Search Tree elements must be comparable so that we can order them inside the tree. When inserting an element we want to compare its value to the value stored in its current node we're considering to decide on one of the following:
   i. Recurse down left subtree (< Case)
   ii. Recurse down right subtree (> Case)
   iii. Handle finding a duplicate value (= Case)
   iv. Create a new node (found a null value)

2. **In-order Traversal:**
   In-order traversal is a systematic way of visiting nodes in a tree. In-order tree traversal is similar to Depth-First Search in a way that we use stack for both Cases. In-order traversal is outlined by 3 steps:
   i. Traverse Left
   ii. Visit Node
   iii. Traverse Right

   The pseudocode for the same is as follows:

```
inorder(node)
    if node == null then return
    inorder(node.left)
    visit(node)
    inorder(node.right)
```

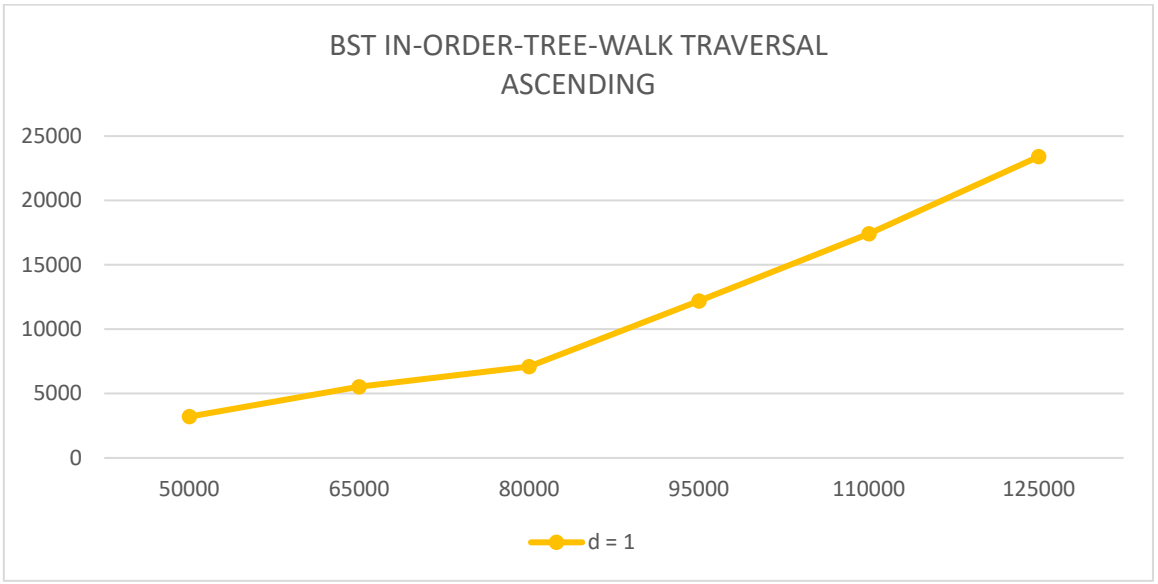
Following are the average running times and graph plots of BST and RBT traversals algorithms:

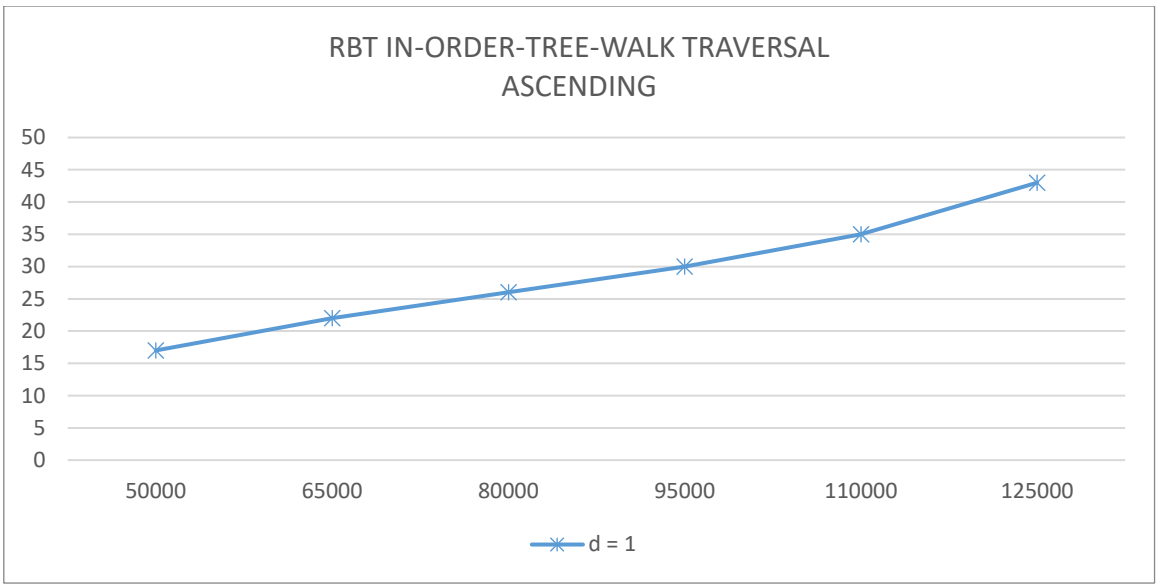
BST IN-ORDER-TREE-WALK TRAVERSAL:

BST IN-ORDER-TREE-WALK TRAVERSAL			
Input Values (n)	**d = -1 (Descending Order)**	**d = 0 (Random Order)**	**d = 1 (Ascending Order)**
50000	3039	9	3208
65000	6024	13	5527
80000	9474	16	7094
95000	13233	23	12192
110000	19247	26	17419
125000	25177	34	23404

RBT IN-ORDER-TREE-WALK TRAVERSAL:

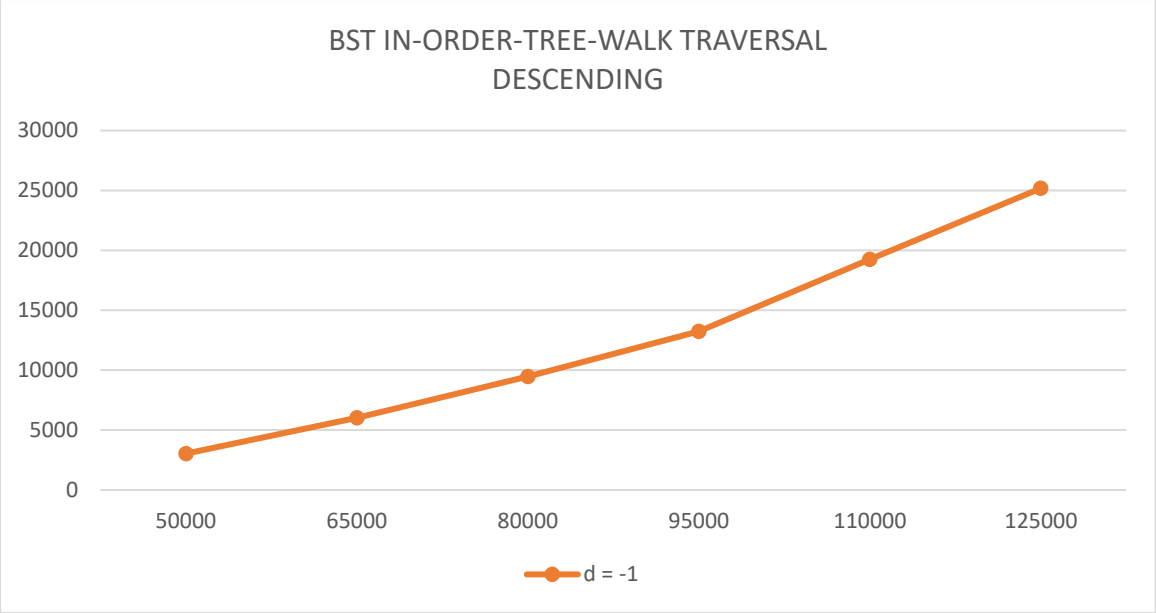
RBT IN-ORDER-TREE-WALK TRAVERSAL			
Input Values (n)	**d = -1 (Descending Order)**	**d = 0 (Random Order)**	**d = 1 (Ascending Order)**
50000	13	19	17
65000	16	30	22
80000	22	34	26
95000	25	37	30
110000	28	53	35
125000	33	65	43

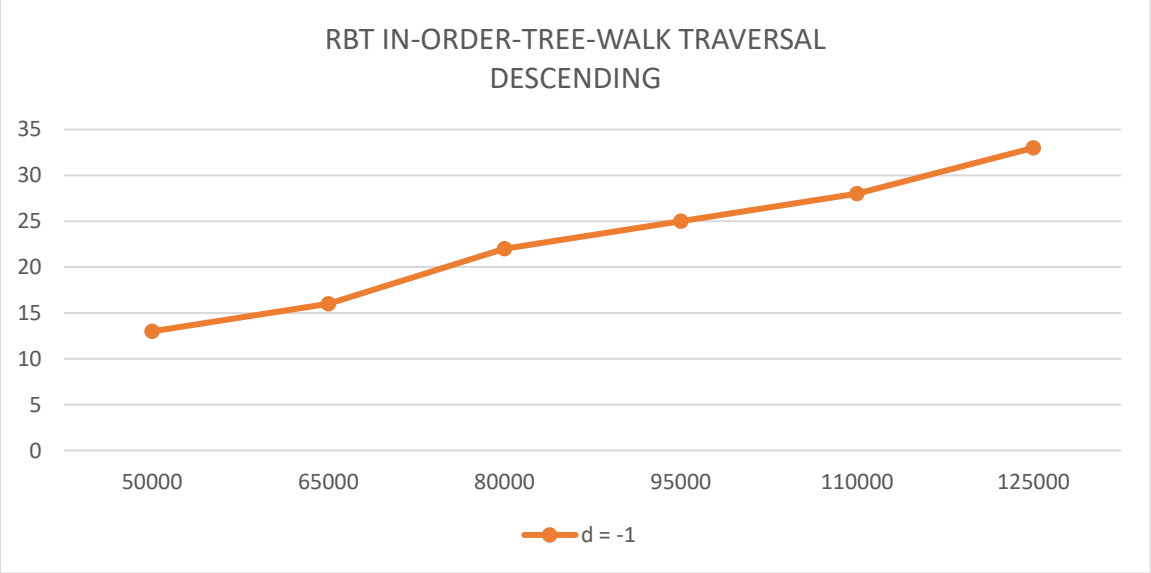




Comparison 1: Ascending Order

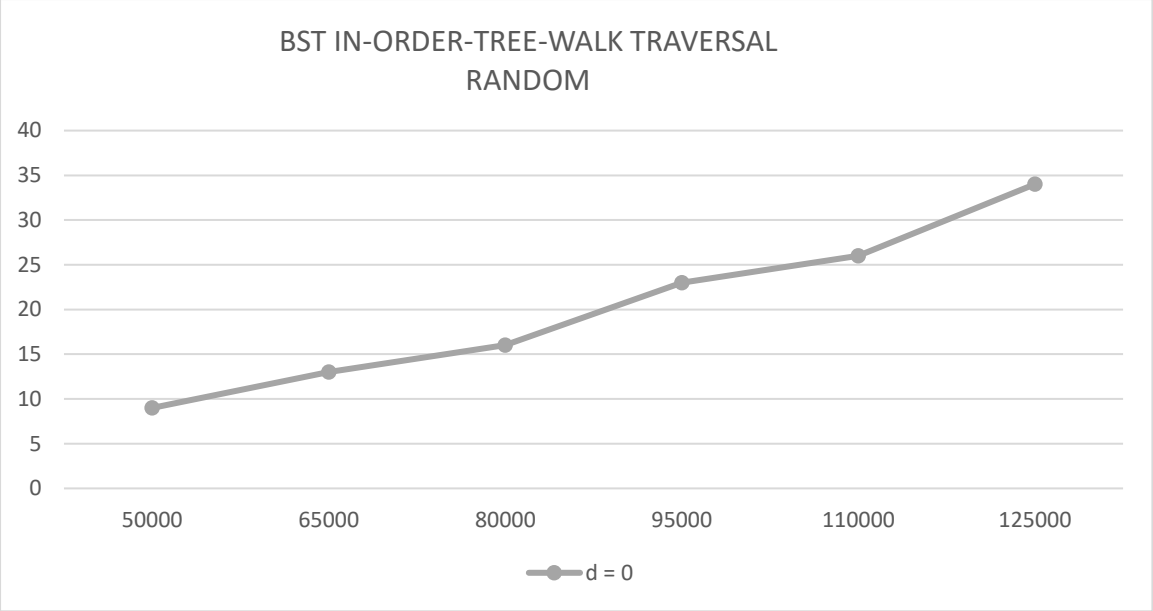
According to the above figure RBT traversal for ascending order takes much lesser time as compared to BST.

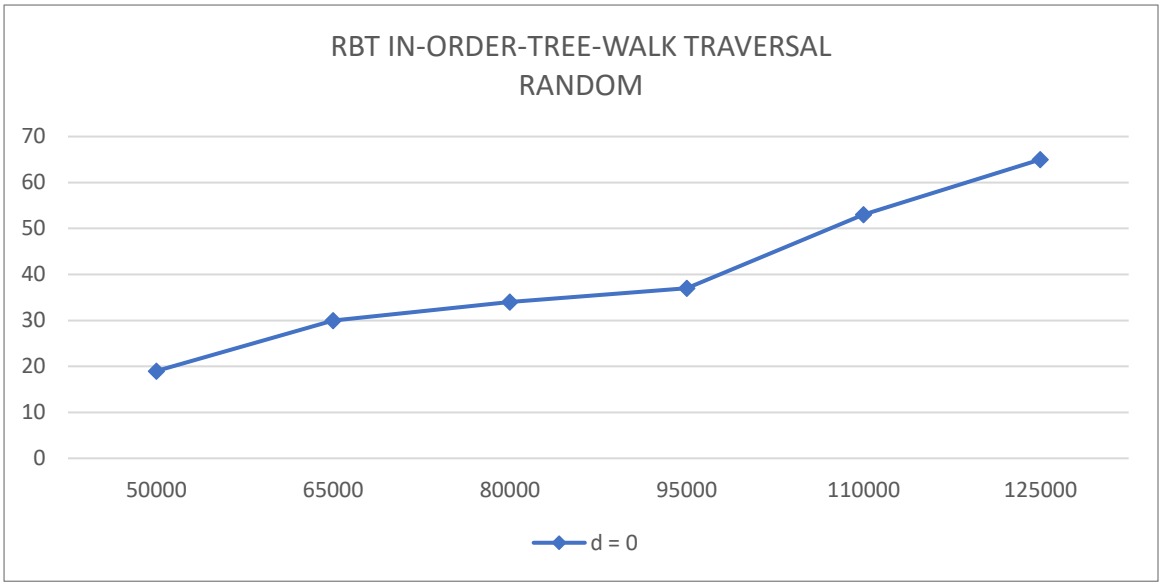




Comparison 2: Descending Order

Again, RBT traversal for descending order takes much lesser time as compared to BST.





Comparison 3: Random Order

Here, the performance of RBT and BST is roughly the same.

Average Duplicate Values and Counters for BST and RBT:

DUPLICATES IN BST IN-ORDER-TREE-WALK TRAVERSAL			
Input Values (m)	**d = -1 (Descending Order)**	**d = 0 (Random Order)**	**d = 1 (Ascending Order)**
50000	0	3	0
65000	0	4	0
80000	0	4	0
95000	0	3	0
110000	0	5	0
125000	0	7	0

DUPLICATES IN RBT IN-ORDER-TREE-WALK TRAVERSAL			
Input Values (m)	**d = -1 (Descending Order)**	**d = 0 (Random Order)**	**d = 1 (Ascending Order)**
50000	Case 1: 49966 Case 2: 0 Case 3: 49971 Left Rotate: 0 Right Rotate: 49971 Duplicates: 0	Case 1: 25657 Case 2: 9805 Case 3: 19561 Left Rotate: 14693 Right Rotate: 14673 Duplicates: 1	Case 1: 49966 Case 2: 0 Case 3: 49971 Left Rotate: 0 Right Rotate: 49971 Duplicates: 0
65000	Case 1: 64961 Case 2: 0 Case 3: 64971 Left Rotate: 0 Right Rotate: 64971 Duplicates: 0	Case 1: 33325 Case 2: 12609 Case 3: 25260 Left Rotate: 18867 Right Rotate: 19002 Duplicates: 1	Case 1: 64961 Case 2: 0 Case 3: 64971 Left Rotate: 0 Right Rotate: 64971 Duplicates: 0
80000	Case 1: 79965 Case 2: 0 Case 3: 79970 Left Rotate: 0 Right Rotate: 79970 Duplicates: 0	Case 1: 41042 Case 2: 15517 Case 3: 30891 Left Rotate: 23233 Right Rotate: 23175 Duplicates: 1	Case 1: 79965 Case 2: 0 Case 3: 79970 Left Rotate: 0 Right Rotate: 79970 Duplicates: 0
95000	Case 1: 94962 Case 2: 0 Case 3: 94970 Left Rotate: 0 Right Rotate: 94970 Duplicates: 0	Case 1: 48765 Case 2: 18484 Case 3: 36948 Left Rotate: 27745 Right Rotate: 27687 Duplicates: 3	Case 1: 94962 Case 2: 0 Case 3: 94970 Left Rotate: 0 Right Rotate: 94970 Duplicates: 0
110000	Case 1: 109961 Case 2: 0 Case 3: 109969 Left Rotate: 0 Right Rotate: 109969 Duplicates: 0	Case 1: 56621 Case 2: 21473 Case 3: 42834 Left Rotate: 32094 Right Rotate: 32213 Duplicates: 3	Case 1: 109961 Case 2: 0 Case 3: 109969 Left Rotate: 0 Right Rotate: 109969 Duplicates: 0
125000	Case 1: 124963 Case 2: 0 Case 3: 124969 Left Rotate: 0 Right Rotate: 124969 Duplicates: 0	Case 1: 64211 Case 2: 24315 Case 3: 48606 Left Rotate: 36421 Right Rotate: 36500 Duplicates: 4	Case 1: 124963 Case 2: 0 Case 3: 124969 Left Rotate: 0 Right Rotate: 124969 Duplicates: 0

Conclusion:

From the above measurements, graphs, and plots, we come to the following conclusion that reverse sorted and sorted arrays are the worst-Case scenarios for Binary Search Trees as the tree is unbalanced. The in-order-tree-walk has a complexity of O(n) where n is the number of nodes in the BST.

Randomly sorted arrays, BST and RBT show roughly the same performance as a randomly created BST with n nodes has the same height as an RBT that is O(logn).

The sorting time complexity for the reverse sorted and sorted RBT would remain O(lgn) since RBT is balanced. The sort time for BST rises exponentially as the input size is increased whereas for RBT there is little change in sort times.