STEVENS INSTITUTE OF TECHNOLOGY

Computer Science (Master's)

CS 590 A - ALGORITHMS
ASSIGNMENT 3

Submission to:
Professor In Suk Jang

Submission By:
Chetan Goyal
CWID: 20005334

Date: November 11, 2021

OBJECTIVES

- To implement a binary-search tree with the corresponding functionality. To modify the insertion routine of the binary-search and red-black tree such that it does not allow duplicate values making sure the insertion functions should not insert a value if the value is already in the tree

- To modify the INORDER-TREE-WALK algorithm for binary-search and red-black trees such that it traverses the tree in order to copy its elements back to an array, in a sorted ascending order where the number of elements in the tree might be less than n due to the elimination of key duplicates. The function should therefore return the number n' of elements that were copied into the array (number of tree elements)

- To modify your insertion routine for binary-search and red-black trees such that it counts the following occurrences over the sequence of insertions
  - Counter for the number of duplicates
  - Counter for each of the insertion cases (case 1, case 2, and case 3) (red-black tree only)
  - Counter for left rotate and for right rotate (red-black tree only)

- Develop a test function for red-black trees such that, given a node of the red-black tree, traverses to each of the accessible leaves and counts the number of black nodes on the path to the leave

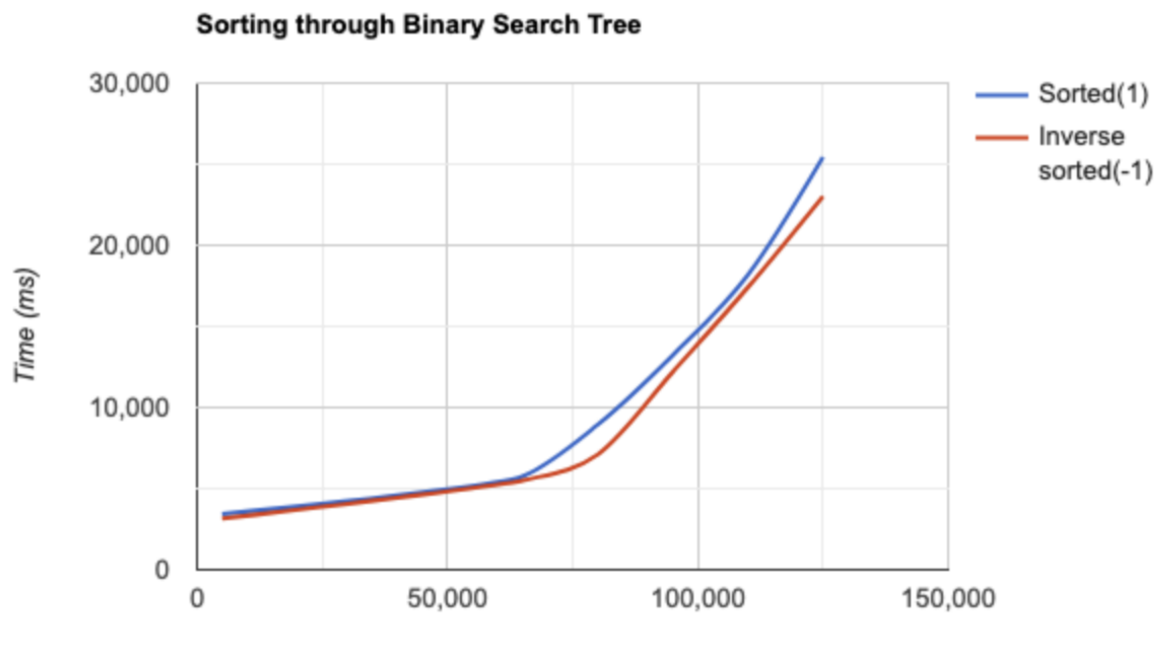
- Measure the runtime performance of your "Binary-Search Tree Sort" and "Red-Black Tree Sort" for random, sorted, and inverse sorted inputs of size n = 50000; 65000; 80000; 95000; 110000; 125000

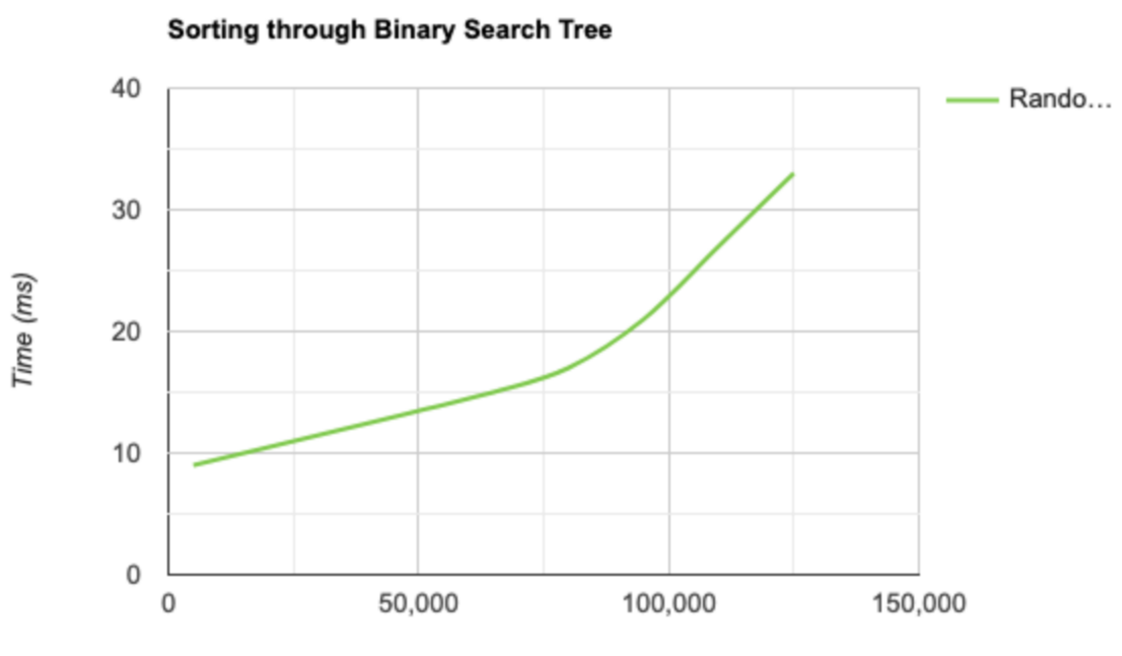
Binary Search Tree Sorting

Below table is the result of modifying INORDER-TREE-WALK algorithm for sorting the binary search tree:

	Order	Sorted(1)	Random(0)	Inverse sorted(-1)
n (Input Values)	Time (ms)			
5000		3423	9	3157
65000		5748	15	5506
80000		8935	17	7094
95000		13223	21	12192
11000		18184	27	17408
125000		25449	33	23027

	Order	Sorted(1)	Random(0)	Inverse sorted(-1)
n (Input Values)	Duplicates			
5000		0	1	0
65000		0	3	0
80000		0	4	0
95000		0	3	0
11000		0	5	0
125000		0	8	0


Sorting through Binary Search Tree
30,000
20,000
10,000
0
Time (ms)
0
50,000
100,000
150,000
Sorted(1)
Inverse sorted(-1)


Sorting through Binary Search Tree
40
30
20
10
0
Time (ms)
0
50,000
100,000
150,000
Rando…

Red Black Tree Sorting

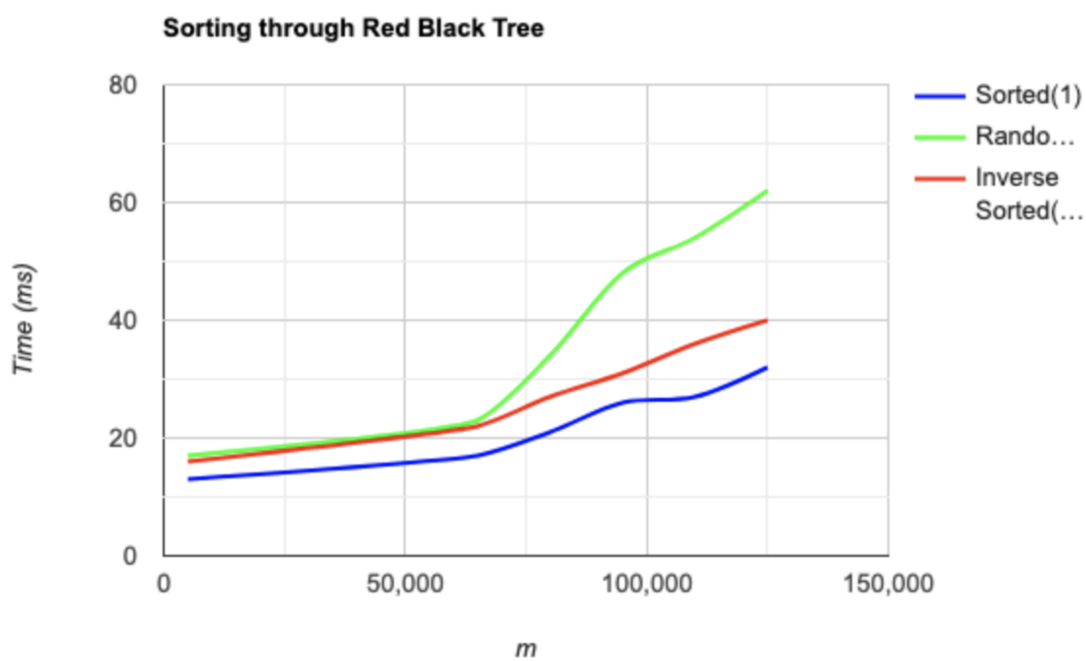
Below table is the result of modifying INORDER-TREE-WALK algorithm for sorting the binary search tree:

	Order	Sorted(1)	Random(0)	Inverse sorted(-1)
n (Input Values)	Time (ms)			
5000		13	17	16
65000		17	23	22
80000		21	34	27
95000		26	48	31
11000		27	54	36
125000		32	62	40

	Order	Sorted(1)	Random(0)	Inverse sorted(-1)
n (Input Values)	Parameters			
5000		Case 1: 49966 Case 2: 0 Case 3: 49971 Left Rotate: 0 Right Rotate: 49971 Duplicates: 0	Case 1: 25672 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: 49971 Right Rotate: 0 Duplicates: 0

65000		Case 1: 64961 Case 2: 0 Case 3: 64971 Left Rotate: 0 Right Rotate: 64971 Duplicates: 0	Case 1: 33425 Case 2: 12507 Case 3: 25095 Left Rotate: 18729 Right Rotate: 18873 Duplicates: 0	Case 1: 64961 Case 2: 0 Case 3: 64971 Left Rotate: 64971 Right Rotate: 0 Duplicates: 0
80000		Case 1: 79965 Case 2: 0 Case 3: 79970 Left Rotate: 0 Right Rotate: 79970 Duplicates: 0	Case 1: 41196 Case 2: 15401 Case 3: 31031 Left Rotate: 23197 Right Rotate: 23235 Duplicates: 1	Case 1: 79965 Case 2: 0 Case 3: 79970 Left Rotate: 79970 Right Rotate: 0 Duplicates: 0
95000		Case 1: 94962 Case 2: 0 Case 3: 94970 Left Rotate: 0 Right Rotate: 94970 Duplicates: 0	Case 1: 48787 Case 2: 18567 Case 3: 37099 Left Rotate: 27800 Right Rotate: 27866 Duplicates: 2	Case 1: 94962 Case 2: 0 Case 3: 94970 Left Rotate: 94970 Right Rotate: 0 Duplicates: 0
11000		Case 1: 109961 Case 2: 0 Case 3: 109969 Left Rotate: 0 Right Rotate: 109969 Duplicates: 0	Case 1: 56478 Case 2: 21259 Case 3: 42952 Left Rotate: 32254 Right Rotate: 31957 Duplicates: 3	Case 1: 109961 Case 2: 0 Case 3: 109969 Left Rotate: 109969 Right Rotate: 0 Duplicates: 0
125000		Case 1: 124963 Case 2: 0 Case 3: 124969 Left Rotate: 0	Case 1: 64205 Case 2: 24235 Case 3: 48509 Left Rotate:	Case 1: 124963 Case 2: 0 Case 3:

		Right Rotate: 124969 Duplicates: 0	36374 Right Rotate: 36370 Duplicates: 1	124969 Left Rotate: 124969 Right Rotate: 0 Duplicates: 0



ANALYSIS continued below

ANALYSIS

From the tabular data as well as the graphs, we can clearly see that for Binary Search Tree, when the worst case scenarios are handled, the time taken is extremely large. But for random the time taken is not large and that is because the height of the tree in case of random is not large and is O(logN) which when compared to worst case scenarios, it is O(N) for Inorder tree traversal.

However in the case of RBT we write a few extra lines of code to make sure no matter what, the tree stays balanced. This is done through colour denotation of nodes in the form of red and black and the use of the properties of red black trees that ensure its sustainability. And, as we can see in the graphs as well as the tabular data, the time taken for worst case scenarios is very less as compared to the time taken when done through BST. This is because the traversal in case of RBT for the worst case scenario still stays O(logN) and as the data increases, the time complexity of O(logN) helps do the traversal very fast.

Also, when we notice the time taken for Random in case of RBT, we notice that it is almost comparable to that of BST. This is expected because the traversal for both the situations is O(logN).

We also notice that the duplicates in case of Sorted and Inverse sorted for BST as well as RBT are almost all 0 but for random there are a few. This is an expected behaviour.

And in RBT, for random, the left and right rotations are almost the same in numbers, whereas for sorted, there are no left rotations but only right rotations and for inverse sorted, there are no right rotations and only left rotations. Also for sorted and inverse sorted, case 1 and case 3 are present but case 2 is 0.

Hence the analysis of practical coding and application is in accordance with our theoretical knowledge that we gained in our class lectures.