Shubham Sanjay Jain

CWID: 10456815

Stevens Institute of Technology

Master of Science in Computer Science

Submitted to: Prof. Iraklis Tsekourakis

CS 590 Algorithms

Report & Solution: Assignment 3

1. You are given an implementation of red-black trees. Implement a binary-search tree with the corresponding functionality. You can omit the delete functionality for binary-search and red-black trees, but you have to update the insertion routine of the binary-search and red-black tree to handle duplicate values. The insertion functions do not insert a value if the value is already in the tree.

   Solution: Implemented using if condition where whether the key of the inserting node is equal to the existing keys of the tree.

```
if (z->key == x->key) {

    ++duplicates;

    // cout<<"\nCan't insert, same key found\n";

    return;
```

2. 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. 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). **Notes:** The algorithm relies only on the binary-search tree properties which also red-black trees satisfy. Keep in mind that only the first n' elements of your array are afterwards sorted.

Solution: Developed an output function which has another output array as parameter and return count as asked. Here, the inorder_output function is called which assigns value to array declared in class. And finally, we get output in final array

```
output(int* output_array) {

        delete_array (A);

        A = new int[count];

        count = 0;

        inorder_output(T_root, 1);

        for (int i = 0; i < count - 1; i++) {

                output_array[i] = A[i];

        }

        // delete[] A;

        // count = 0;

return count;
}
```

3. Implement an insertion function that has an array of integers and the array length as arguments. 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).

  You should have 1 counter for binary-search trees and 6 counters for red-black trees altogether.

Solution: The counter functions of integer as return type are implemented for all the 7 counters and are displayed depending upon type of tree called. If BST is called then 1 counter output is displayed where as for RB Tree, 6 counters are displayed.

3. 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.
Notes: You could use your test function to verify whether or not your red-black tree implementation satisfies red-black property 5.

Solution: I have developed heightTree function which returns the black height of any given node and code can be uncommented to check the height of the node.

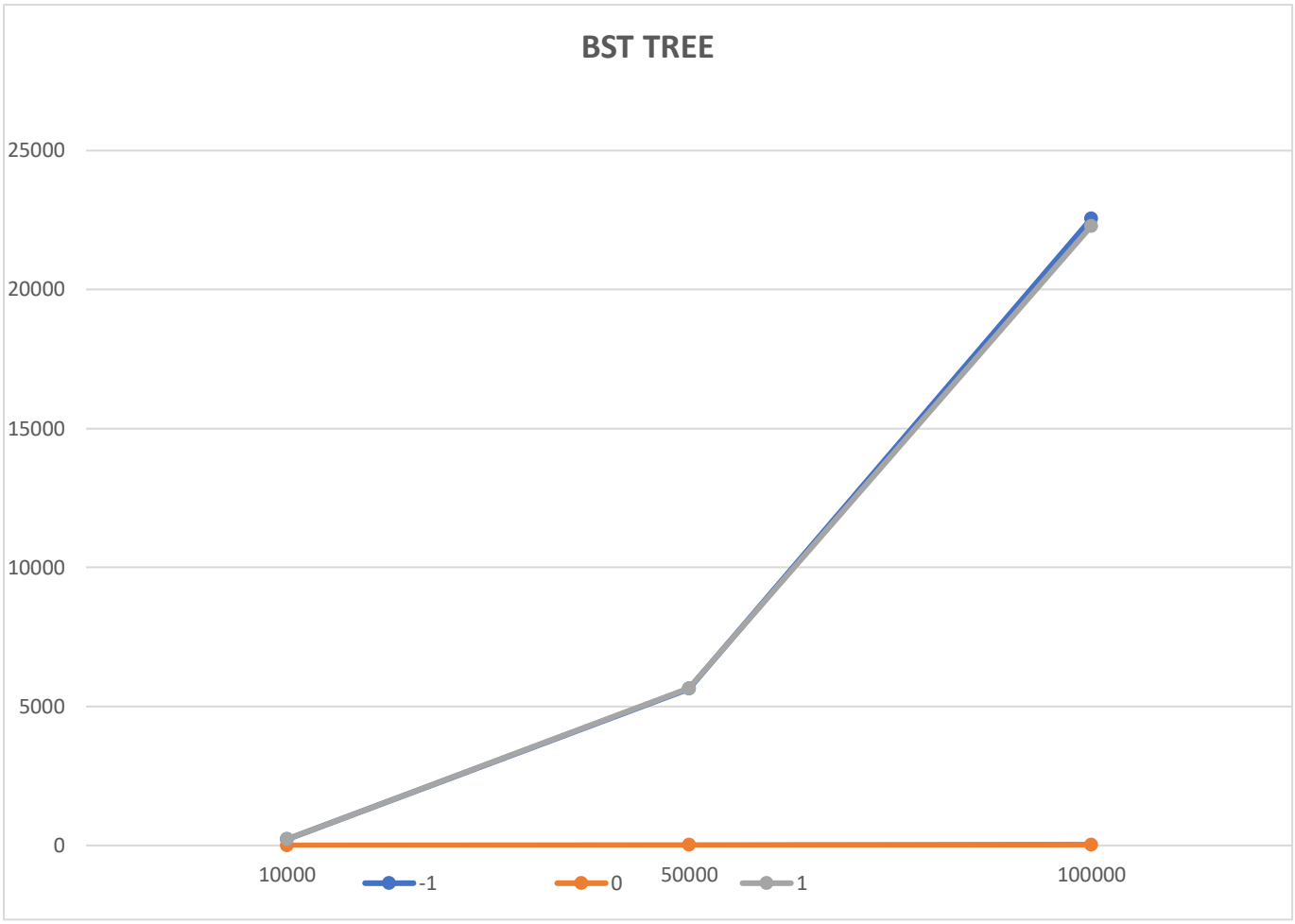
5. Measure the runtime performance of your "Binary-Search Tree Sort" and "Red-Black Tree Sort" for random, sorted, and inverse sorted inputs of size n = 50000; 100000; 250000; 500000; 1000000; 2500000; 5000000. You can use the provided functions *create random, create sorted, create reverse sorted*. Repeat each test a number of times (usually at least 10 times) and compute the average running time and the average counter values for each combination of input and size n. Report and comment on your results. How do the counters behave and how is the height of the respective trees in comparison to how the running time behaves.

(You might have to adjust the value for n dependent on your computers speed, but allow each test to take up to a couple of minutes. Start with smaller values of n and stop if one instance of the algorithm takes more than 5 min to complete).

Solution: The results are reported in below tables, my observation and conclusion are as follows,
The height increases, the input time also increase. The counter left for reverse sorted is difference of total elements and height of the tree and for sorted input, the right counter is difference of total elements and height of the tree. And height for random input is always smaller than the sorted and reverse sorted inputs. And the height of reverse sorted and sorted is same. Also, as the input size of random increases, the duplicated also increases. The time for insert increases exponentially as random inputs increases. Case 2 counter shows non zero for random inputs otherwise zero.

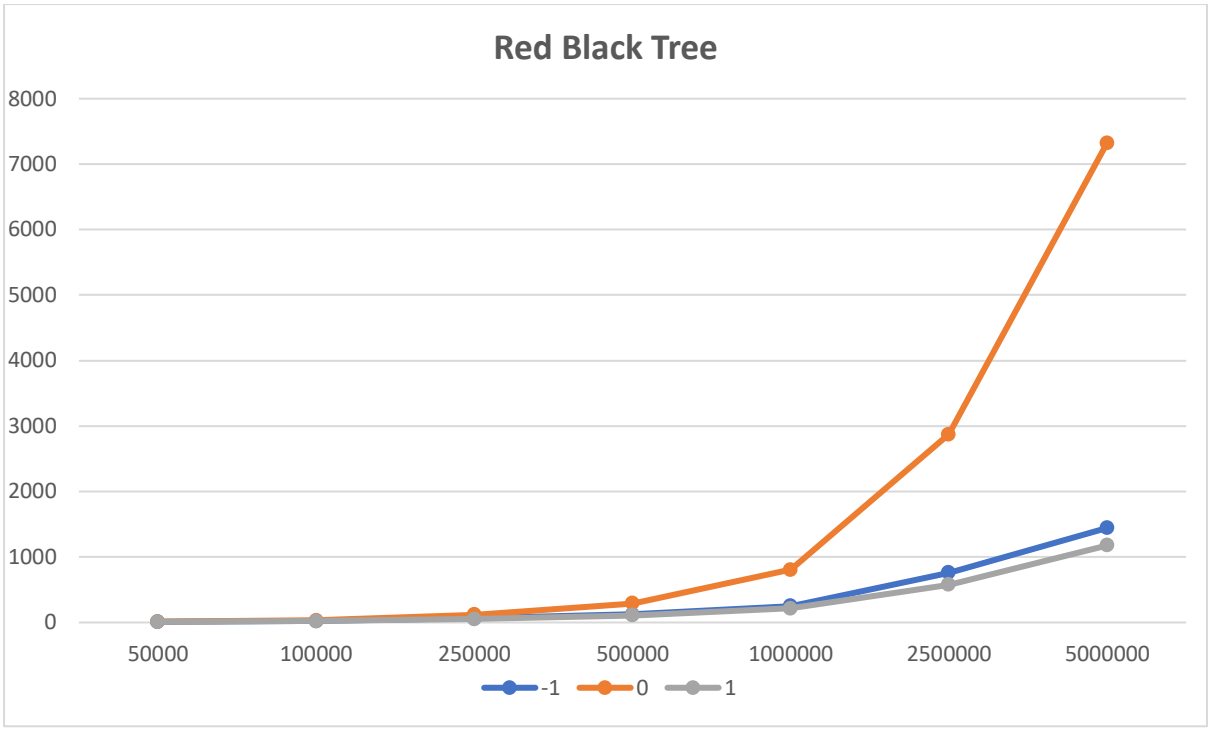
Binary Sort Tree

Size Input	Direction	Time (Millisec)	Height	Duplicates
10000	-1	223	10000	0
	0	1.7	31.4	0
	1	225	10000	0
50000	-1	5637.3	50000	0
	0	14.3	37.5	0.6
	1	5654.6	50000	0
100000	-1	22546.7	100000	0
	0	31.5	41	3.4
	1	22281.6	100000	0



RedBlack Tree

Size Input	Direction	T Real	Height	Duplicates	Count Left	Count right	Case 1	Case 2	Case 3
50000	**-1**	10.6	29	0	0	49971	49966	0	49971
	0	12.8	19	0.2	14534	14580.4	25691.6	9700.4	19414
	1	9.8	29	0	49971	0	49966	0	49971
100000	**-1**	21.5	31	0	0	99969	99964	0	99969
	0	33.7	20	2.4	29144.4	29140.3	51319.6	19428.7	38856
	1	20.5	31	0	99969	0	99964	0	99969
250000	**-1**	60.2	33	0	0	249967	249961	0	249967
	0	118.4	22	14.2	72902.1	72807.1	128384.7	48568.2	97141
	1	51.8	33	0	249967	0	249961	0	249967
500000	**-1**	123.6	35	0	0	499965	499959	0	499965
	0	292.9	23	60.6	145787	145603	256684.6	97088.3	194301.7
	1	106.4	35	0	499965	0	499959	0	499965
1000000	**-1**	249.9	37	0	0	999963	999957	0	999963
	0	805.6	24.1	236.8	291298.9	291077.4	513352.4	194133.2	388243.1
	1	212.4	37	0	999963	0	999957	0	999963
2500000	**-1**	758.4	40	0	0	2499960	2499952	0	2499960
	0	2873.7	26	1458	727631.7	727706.5	1282958.7	485122.1	970216.1
	1	577.6	40	0	2499960	0	2499952	0	2499960
5000000	**-1**	1442.4	42	0	0	4999958	4999950	0	4999958
	0	7322.2	27	5833.6	1454273.4	1454401.8	2564181.6	969281.5	1939393.7
	1	1175.4	42	0	4999958	0	4999950	0	4999958


Red Black Tree
8000
7000
6000
5000
4000
3000
2000
1000
0
50000
100000
250000
500000
1000000
2500000
5000000
-1
0
1