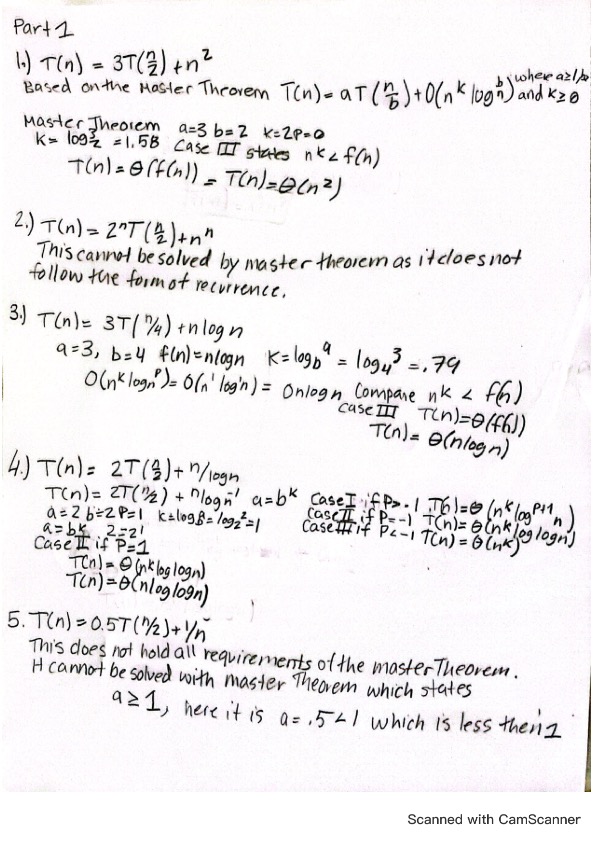
Part 1: Master Theorem



**Abstract:**

The purpose of this assignment was to implement both a Binary Search Tree and a Red Black Tree. For both the Binary Search Tree and the Red Black code was written to not allow duplicate values. The Inorder tree walk was also updated for both the binary search tree and the Red Black Tree in order for it to traverses the tree to copy its elements back to an array in ascending order. This assignment also modified the insertion routines for both algorithm Binary Search Tree and Red Black tree to count the number of duplicates, Count the insertion case cases , Case 1, Case 2 and Case 3, which is something that only is done for the Red Black Tree. The insertion was also modified to count the left rotate and right rotate which also is only done for Red Black Trees. Lastly a Test function was written to test the height of the red black tree and that counts the black nodes of said tree.

Once the code was written, the test for both algorithms were performed for the following n inputs: n = 50000; 100000; 250000; 500000; 1000000; 2500000;   
5000000.

Analysis

Binary Search Tree

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Random Average | Random Duplicates | Sorted Average | Sorted Duplicates | Inverse Avg | Inverse Duplicates |
| |  | | --- | | 50000 | | |  | | --- | | 7.8 | | .7 | |  | | --- | | 2381 | |  | |  | |  | |  | |  | |  | | 0 | |  | | --- | | 23 | |  | | 0 |
| 100000 | 17.6 | 2 | 11418.7 | 0 | 2873.6 | 0 |
| 250000 | 54.6 | 14 | - | - | - |  |
| 500000 | 156.4 | 55.8 | - | - | - |  |
| 1000000 | 407.8 | 231.2 | - | - | - |  |
| 2500000 | 1649.9 | 1453.8 | - | - | - |  |
| 5000000 | 4286 | 5823.5 | - | - | - |  |

Graph 1: Binary Search Tree

Table 2 Red Black Tree

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | Random Average | Tree Height | Case 1/Case 2/Case 3 | Random Duplicates | Sorted Average | Red Black Tree Height | Sorted Duplicates | Case 1/Case 2/Case 3 | Inverse Avg | Red Black | Inverse Duplicates | Case 1/Case 2/Case 3 |
| |  | | --- | | 50000 | | |  |  | | --- | --- | | |  | | --- | | 0.6 | | | 10 | 2565541/970094/1938507 | .4 | |  | | --- | | 0.1 | |  | |  | |  | |  | | |  | | --- | |  | |  | | |  | |  | |  | |  | |  | |  | | 15 | 0 | 479966/0/49971 | |  | | --- | | 0 | |  | | 21 | 0 |  |
| 100000 | 1.2 | 10 | 51406/19460/38801 | 1.7 | 0.3 | 15 | 0 | 9964/0/99969 | .3 | 16 | 0 | 9964/0 9969 |
| 250000 | 3.8 | 11 | 128405/48347/969990 | 13.5 | 1.5 | 17 | 0 | 249967/0/249967 | 1.4 | 17 | 0 | 249961/0/149967 |
| 500000 | 155 | 12 | 256802/96447/193531 | 55.9 | 2.6 | 18 | 0 | 499959/0/499965 | 2.8 | 18 | 0 | 499959/0/4999965 |
| 1000000 | 380.1 | 12 | 513443/194344/291757 | 234.4 | 5 | 19 | 0 | 999963/0/999963 | 5.1 | 19 | 0 | 999957/0/999963 |
| 2500000 | 1758.2 | 13 | 1283263/485611/970207 | 1455 | 12.8 | 20 | 0 | 2499952/0/2499960 | 13.1 | 20 | 0 | 24999952/0/24999960 |
| 5000000 | 4335.7 | 14 | 2564615/970179/1940602 | 5846.5 | 25.6 | 21 | 0 | 4999958/0/4999958 | 26.4 | 21 | 0 | 4999950/0/499999958 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Left Rotation - Random | Right Rotation -Random | Left Rotation- Sorted | Right Rotation- Sorted | Left Rotation- Inverse | Right Rotation-Inverse |
| |  | | --- | | 50000 | | 14565.1818 | 14561 | 49971 | 0 | 0 | 49971 |
| 100000 | 29094.8182 | 29115.4545 | 99969 | 0 | 0 | 99969 |
| 250000 | 132278.818 | 72887.7273 | 249967 | 0 | 0 | 249967 |
| 500000 | 145709 | 145705.909 | 499965 | 0 | 0 | 499965 |
| 1000000 | 291340.182 | 291226.364 | 999963 | 0 | 0 | 999963 |
| 2500000 | 727468.182 | 727732.727 | 2499960 | 0 | 0 | 2499960 |
| 5000000 | 2644127.55 | 1454982.73 | 4999958 | 0 | 0 | 4999958 |

Graph 2 Red and Black Trees

**Analysis:**

Table 1 above shows the values gathered by running 10 test each for all m numbers for the Binary Search Tree, for Random Sorted, Sorted and Inverse Sorted. The Binary Search tree behaved exactly how I thought it would for the Sorted and Inverse, where I expected it to follow the Worst Case Scenario, which states that the Binary Search Tree would perform with a Time complexity of   
Θ 𝑛^2, which happens when a linear chain of nodes results from the repeated Tree-Insert operations. The worst case in this case mimics the idea of a linked list. Which was the behavior I noticed when performing the Inverse and Sorted, whereas the n value grew the time exceeded 5-10 minutes, and where not able to finish.

For the random sorted Binary Search tree, it behaved much better, and mimics much closer the best-case scenario of the Binary Search Tree which states that it follows the time complexity of: Θ 𝑛 lg𝑛. Here, we can see that the random Binary Search Tree was able to complete the full run from all n input sizes. By definition the Best Case Scenario states that it occurs when a binary tree of height O(lg N) results from repeated Tree Insert Operations. For all of these I also drafted some graphs, but the graphs are not the best graphical representations, but the Graph 1 shows the representation of Binary Search Tree, and here we can again see clearly that Sorted and the Inverse behaves it a much worse case scenario then Random Binary Search Tree. I also tried to compare the Random Binary Search Tree with C1nlogn and C2nlogn, but after a few failed tries with picking the constant, I was not able to get any meaningful data from it.

The duplication counter was also implemented for the Binary Search Tree which showed a 0 duplicated for Inverse and sorted Binary Search Tree which made sense. I did see the duplication counter in action for the random Sorted Binary Search tree, which showed me that as the value of n increased the running time for the algorithm increased at the same time as the the duplication counter. It was also apparent that even though the Height of the tree has not increased much, the duplication counter increased significant as the n number increased.

The second part of this assignment was working with Red Black Trees. Red Black Trees generally behave in the Insertion with a running time complexity of O(lg n). For this algorithm I soon realized as the input size n increased, the average running time increased rapidly compared to the height of the tree, which can be seen by the above Table. This is also shown by the duplicate counter, Left and Right Rotations, and the Cases. Here we can also see that for Sorted and Inverse, we did not get 0 for those criteria. However, for the Random Binary Search Tree I received duplicates and they rapidly increased as n increased. I also noticed a pattern between the Left and Right Rotations. When a Sorted Binary Search Tree is done, only left rotations are showing a value and a right rotation shows as 0, and this is flipped for Inverse, where I got no left rotations but right instead. I also noticed that for those scenarios, the numbers of rotations are the same between left and right when switching for Inverse and Sorted at a specific n input. For the Red Black Tree the case still continues to go up the same consistency as it did earlier for the Binary Search Tree. When looking at the cases it also was apparent that Cas 2 only had values for the Random Sorted Red Black Tree but none for Sorted and Inverse. I also added a graph to show the time complexity but for this as well it was tough to get a fair represation with the C1nlogn. However based on the graph one can see that C1nlogn does behave on a similar trend as the Red Black Trees for Random and Inverse,but Sorted behaved much different and not on a path that can be compared to C1nlogn.

**Conclusion**

Based on the analysis and the Running Time table, I came to the conclusion that if I was to choose between the Binary Search Tree and a Red Black Tree time complexity. I would choose the Red Black Tree, because it shows that it is more efficient than the Binary Search Tree. This is shown by the high time complexity which can be seen by table 1. The behavior of this is completely expected especially since the RBT is defined as a self-balanced tree which does not form as a single list that behaves more like a linked list such as the BST.