Project 2

Addison Armstrong

CS:3330:001

10/22/2019

2 Project Report 2

This is a project report over the analysis of runtimes between Recursive insert/remove and Iterative insert/remove on Binary Search Trees, analysis of runtimes between AVL, Black-Red, and Splay Trees, and lastly, the analysis of runtimes between the sorting algorithm of AVL, Black-Red, and Splay Trees. We will start the analysis of Question0.

**Question0:**

Question0 prints the runtime of the creating and sorting algorithm for all trees, Binary Search Tree, AVL, Black-Red, and Splay. As the size of the tree is increased, the runtime grows linearly. See graph below.

(y-axis measured in milliseconds)

**Question1:**

Students were asked to make Iterative insert and remove methods called insert1 and remove1 in the treeNode class. These methods could not call upon itself but could call about helper classes. I have ran the Question1 method 500 times with an input ranging from 10000-500000 to get 5,000,000 to 250,000,000 different operations. Shown below, the Iterative Insert and Remove methods ran significantly faster than the recursive methods. When input nodes was increased, the runtime grew linearly.

(y-axis measured in milliseconds)

**Question2:**

Students were asked to show the creating and sorting algorithms for AVL, Red-Black, and Splay trees and find the average runtime of all of them. Like in Question1: I have ran the Question2 method 500 times with an input ranging from 10000-500000 to get 5,000,000 to 250,000,000 different operations. You can see that Red-Black trees are significantly faster when the node size is increased. But when the node size is minimal, there is almost no difference in times. You can see this in the Graph below.

(y-axis measured in milliseconds)

**Question3:**

Students were asked to create a new sort method to sort the AVL, Red-Black, and Splay trees. To find the inorder sort of the tree, I found the smallest node key, deleted it, and put the key into an array. Deleting the node that contains the key will remove that key from the tree and to find the next smallest key in the array. Like in Question1 and Question2: I have ran the Question3 method 500 times with an input ranging from 10000-500000 to get 5,000,000 to 250,000,000 different operations. This showed that Red-Black trees had the fastest run times. Almost by half. I believe that this happened because Red-Black trees already have a unique sorting algorithm to put them in red or black nodes. You can see the significance of how fast the Red-Black trees sorted in the graph below.

(y-axis measured in milliseconds)

**Conclusion:**

My hypothesis of the project is that AVL trees were going to be the fastest trees besides Binary Search trees. But to my disbelief, Red-Black trees were one of the fastest trees to sort and manipulate. This was a great challenge to code these unique algorithms.

I believe in this project, I would expect Recursion methods to run faster in Binary Search Trees, but I was wrong, the Iterative methods ran faster than the Recursion Methods. Maybe it is because that Recursion calls itself multiple times, causing a lot of different objects and methods to be created and called.

Errors in project maybe that my computer is too slow to calculate 500,000 correctly. My computer ran for 30 mins on one question. Along with this, many of my graphs do not contain units. All units are provided below the graphs. I believe that my recursion was supposed to be faster, so there may be an error somewhere that caused a miscalculation.