Atul Srivastava
Professor Shanthenu Jha
Data Structures and Algorithms Assignment 3
April 2, 2019

*All of the problems are using https://github.com/peterhil/leftrb library which has implementations for BST and for 2-3 left leaning RB tree.

Problem 1

I have found an implementation of a left leaning red black tree that models the tree parameters specified in the problem and in the book. It is https://github.com/peterhil/leftrb. I put the relevant source code in the folder for this problem, and in addition, have written an inorder traversal.

Essentially, a 2-3 tree without balance restriction and a RBT without any left or right rotations, or any color flips. When a new node is added, we have to make sure that the parent node is hooked with a black node. The implementation in the library is as follows: a red node is the default insertion, and if 3-node present, it should be a black node.

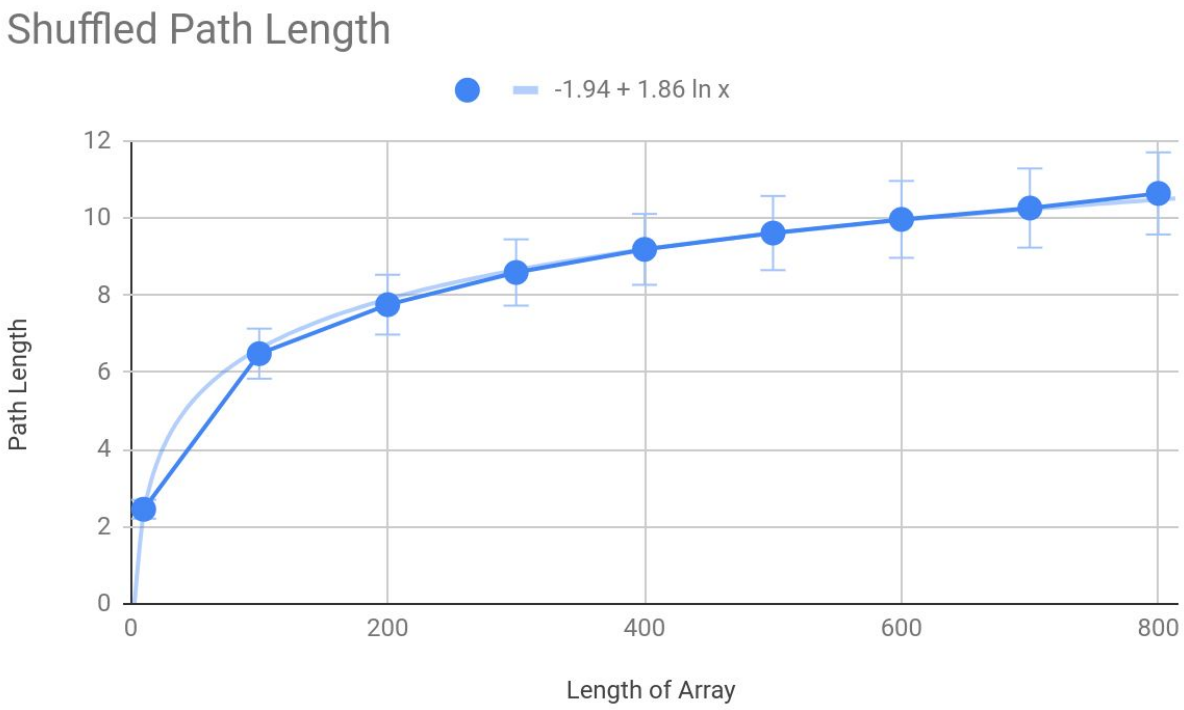
Problem 2

This problem is using a BST implementation that is provided by: https://github.com/peterhil/leftrb.

The tree is built with both a shuffled array and a sorted array of varied input lengths. I ran it for 10 trials of each path length. I have commented some of the different tests I used for different situations.

Length	Path length (ordered)	Path length (shuffled)
10	2.75	2.45099
100	25.5	6.4821
200	50.5	7.752
300	75.5	8.5856
400	100.5	9.1843
500	125.5	9.6069

600	150.5	9.9611
700	175.5	10.255
800	200.5	10.634
900	225.5	10.874
1000	250.5	10.858



For the sorted path list, there seems to be a linear relationship in relation to the path length.

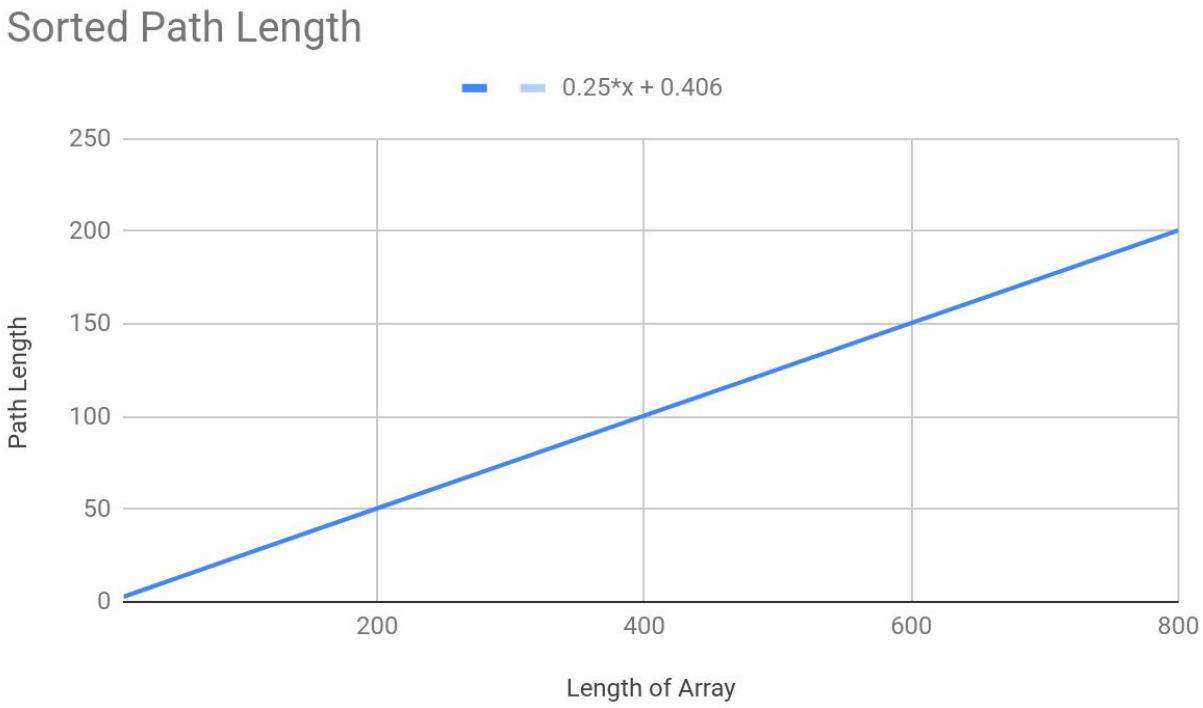
Using left leading 2-3 trees, we get similar results in relations between the ordered path length and shuffled path length.

The relation between the shuffled path lengths is described by the equation of
$f(x) = 1.86ln(x) - 1.94$

In order to find the path length of a tree, we should use the following relation. The path of a tree is found by the length of the current node + the length of left subtree + the length of right subtree. As a general form, we can formalize this in a hypothesis

We can come up with the following hypotheses:
Average path length (ordered) ~ N/4 +0.5
Average path length (shuffled) ~= length of current Node + length of left subtree + length of right subtree

Based on our data, this is given by $f(x) = 1.86ln(x) - 1.94$



Problem 3
After running 100 trials on N = 10^4, 10^5, and 10^6, we get a fairly consistent percentage of red nodes.

Result of 10000: 22.28%

Result of 100000: 23.14%

Result of 1000000: 23.19%

Note: when testing the code, you can reduce the number of trial size (it is currently set to 100), because it took quite long to run on my computer for the entire size.

After running these tests, we can make a hypothesis that ~23.2%

Theoretically, the floor of the amount of red nodes should be ~25%.

The classes for a left leaning red black tree is taken from someone's github library, and I have implemented all the traversal and counting functions for determining the amount of red nodes. The code can be found here:
https://github.com/peterhil/leftrb

Problem 4

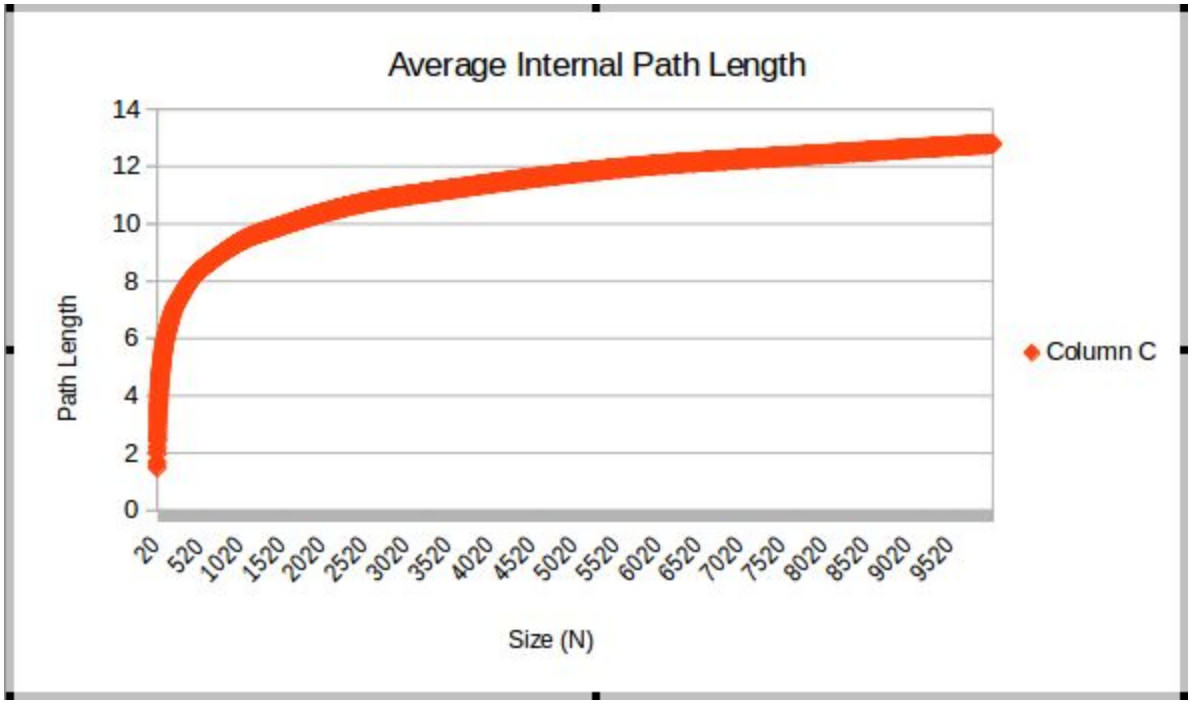
The average path length calculating algorithm in this problem is essentially the same as it was in problem 2. However, we are going to be using a different type of tree when doing this problem. Also, we will be keeping track of the standard deviation and the average of the average length of a path to a random node.

```
def path_calculation(root, curr_depth=0):

    if not root:
        return 0

    else:
        return curr_depth + path_calculation(root.left, curr_depth + 1) + path_calculation(root.right, curr_depth + 1)
```

Using pseudocode provided in the slides, this is my simple algorithm for calculating the path length. It is the same algorithm that is used in the second question.

All the relevant data is stored in the Q4 folder when running these tests. Since I ran these tests relatively late before the due date, I wasn't able to run it for the full amount of trials desired, but I have some data that is useful.

The equation based on the data is $f(x) = 2.462\ log(x) - 1.167$



Problem 5

I implemented the select and rank algorithms by performing inorder traversals on a binary search tree. The binary search tree was populated with data provided in the question, and an inorder traversal puts all the data in order so it is easy to perform rank and select operations.

The result are:

select(7) = 8

rank(7) = 6