Design Rational:

The functions of both trees are similar to that of a regular binary search tree. The changes come when rebalancing and recolouring for the Red-Black Tree. The Nodes are given to us in reference counters, and to get values, a borrow is required to be invoked on them. This can cause issues as if a borrowed node is passed through any function, the value cannot be borrowed again. To overcome this issue, whenever a node is borrowed, it is immediately returned. This is the reasoning for the Enumeration of type scenarios of the state of the tree after insertion and recoloring. It allows for a node to be returned so rotation operations can be done on it.

The command-line interface was implemented using the terminal and inputs are taken through the keyboard for user ease.

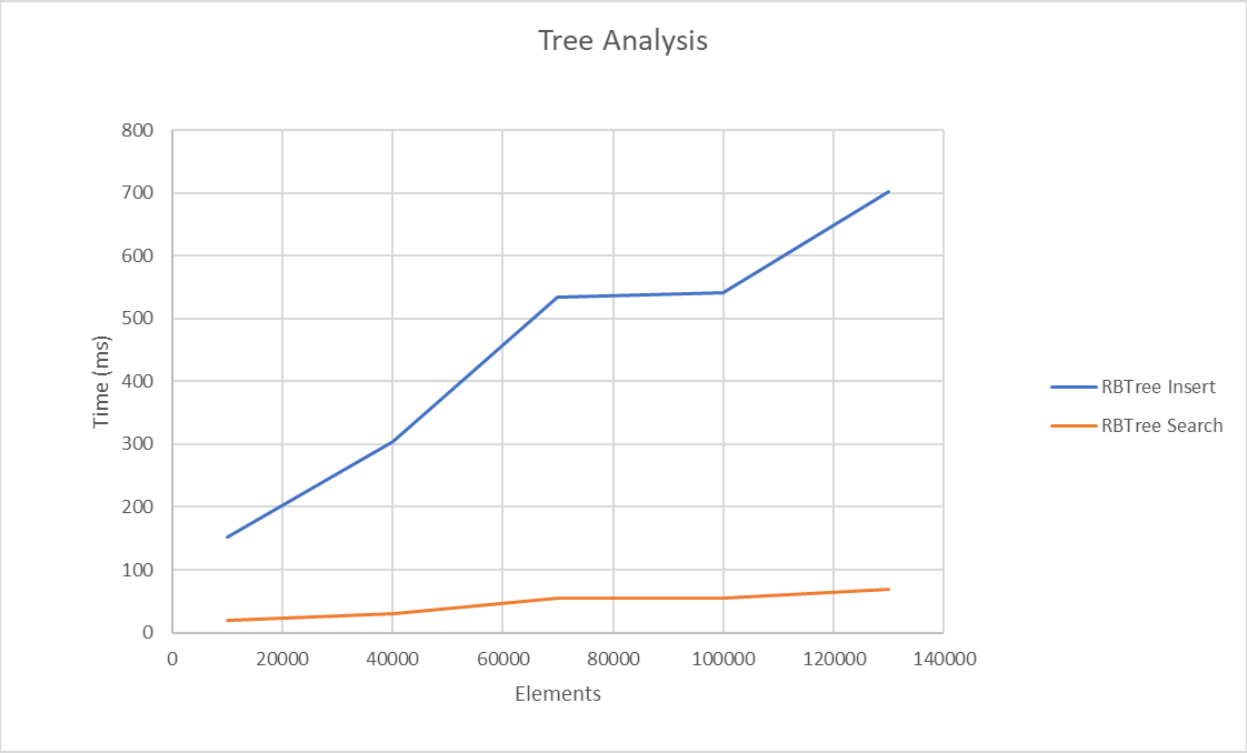
Incomplete Functionality:

Delete for the Red-Black Tree is simply a binary delete and does not rebalance, nor recolour the tree. Attempted implementation is commented out, it ran into issues at runtime. The AVL Tree is not implemented.

Design Questions:

1. A red black tree balances the nodes. This avoids the worst case scenario for inserting and searching a binary tree, which would be adding in values sequentially, making the tree tilted on one side. Balancing the nodes avoids this and makes searching in this scenario much faster.
2. Command line interface is implemented in the main function.
3. The need to check if a node is None, is done repeatedly to ensure that it can be unwrapped and that there is a value to use.
4. They both include rotations to balance the trees and are both based on binary search trees, so are searched through the same way, and inserting and deleting both include procedures from the binary search tree.
5. Using good generalizations that are based on the binary search tree could be used as a basis to build on to create other trees based on binary search trees.

Data:



As can be seen, there is a trend of a O(log(n)) time complexity when it comes to the search function, which is correct for the worst case scenario. The trend is seen in the inserting, however it is not as distinct, possibly hinting at some implementation snags. In theory, when it comes to searching for data, the AVL tree should be better and quicker. However, it requires more balancing so if the requirement is to continuously insert more and more into the tree, then a red-black tree might be better. Deleting might also need to be tested for complete testing. A baseline test for a binary search tree would not be a bad idea, just to see where the other trees improve upon it.

User Manual

Users can run the program using cargo main. From here, simply select the tree you want to work with. Once that has been selected, the interface in the terminal will display to the user commands to insert, search, delete and print out the tree.

Commands:

First to Select Tree:

RB: Use RBTree
AVL: Use AVL Tree

Once within tree commands:

I <integer>: Inserts integer into tree
D <integer>: Deletes node with integer from tree
C: Clears tree
P: Prints Tree
T: Executes a traversal:
    1: In-order Traversal
    2: Post-order Traversal
HT: Returns height of the tree
CN: Returns count of all nodes
CL: Counts leafs
E: Exit program
H: Displays list of commands