Names: MingYang Mao, Brock Chelle, Andrew Chen.

Q1. Red-black tree can balance itself, this is unlike a ordinary bst. In an ordinary bst, if we keep inserting nodes going from 0..n they will keep pointing towards the right. This means if we want to check that node n exists, we would need to traverse the bst's n nodes.This is unlike a red-black tree that can balance itself to be a height of log(n). Thus the operations do not have a worst case time complexity of O(n) like a bst would have.

Q3. Our system handles all erroneous conditions. We maintain nodes in the option class so that if nodes do not exist, they will return a None value. Thus, all errors are caught and not noticed by the user. Currently, for cli we only support i32 input type.

Q4. Red-black tree and AVL tree both have the concept of balancing the tree. However, they do their balance with different conditions and using different rotational conditions. Both the red-black tree and avl tree are bst and thus can be traversed in the same way, and their nodes can also be printed using the same dfs traversal pattern. They guarantee log(n) operations for search, insert and delete.

Q5. While there are some components that are tree-like in nature. 2-3-4 trees and B+ trees are too different for our implementation of both the AVL and RB trees. While we might be able to recycle general ideas, such as storing nodes in a vector instead right/left childs, the code would require extensive changes to allow for this. To design for future implementations, we could use the idea of vectors for children and build our implementation around vectors instead of static variables.

**User Guide**  
Insert: Inserts a node into the tree  
 cmd: insert #  
ie insert 3

Height: Returns the height of the tree  
 cmd: height

Number of leaves: Returns the number of leaves  
 cmd: num\_leaves

Empty: Returns whether the tree is empty or not  
 cmd: is\_empty

In-Order traversal: Returns the inorder traversal of the tree  
 cmd: inorder

Delete: Deletes a node from the tree  
 cmd: delete #  
 ie delete 4

Print: Pretty prints the tree  
 cmd: print

Close: exits the program  
 cmd: close

**Performance discussion.**

In our particular case the red-black tree and avl tree have similar performance. They both have log(n) insertion, deletion, search times. However AVL tree will performance slightly better in the benchmarking tests. We attribute this to AVL trees more rigid balancing, thus although it loses performance during the initial insertions and deletions, the AVL tree will have faster--although minimal--lookup times due to a more balanced structure. To thoroughly compare the performances, we should only consider tests where each tree does a single kind of operation such as insert/delete and not mix the search functionality. That way we can narrow down the tree we wish to use for a particular use case, it may be better to use RBT when we have more insertions/deletions, but better to use AVL when lookup times a important. We should also use a baseline such as a BST to form a control for our performance analysis, and if we truly wish to optimize performance, it is important to use as many different data structure as possible.