Question 1. (Exercise 13.1-3 Cormen et al.)

[Estimate: 5 minutes] Let a relaxed red-black tree (RBT) be a binary search tree that satisfies red-black properties 1, 3, 4, and 5. In other words, the root may be either red or black. Consider a relaxed RBT, T, whose root is red. If we color the root of T black but make no other changes to T , is the resulting tree a RBT? Explain your answer.

Yes, the resulting tree is RBT. Property 1 is satisfied as the nodes are red or black. Property 2 is satisfied as the root is black. Property 3 is satisfied as nodes with no parents or leafs will have NIL instead. Property 4 is satisfied as the remainder of the nodes will remain the same as from the relaxed RBT and so the red nodes still have black leaves. Property 5 is satisfied because bh(x) from the relaxed RBT is the same for every node and by changing the root node to black, we just add one to bh(x). The nodes to descendants still contain the same value of black nodes.

Question 2. (Exercise 13.1-5 Cormen et al.)

[Estimate: 5 minutes] Show that the longest simple path from a node x in a RBT to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

Property 4 and 5 ensures that the length of the longest path from a node x to a descendant leaf is at most twice that of the shortest path. The shortest path would be if there is a black node with a left black leaf. The depth is one and this is the shortest path. With the same example, if the right leaf is red, the leaf of that node has to be black (property 4) and from there, the tree cannot "grow" any more because the bh(x) is 1 for that node. This gives us a longest path of length two and shortest path of length one. This applies for more complex examples because of property 4 and 5. A red node can't grow forever because the child of a red node has to be black and the number of black nodes is limited thanks to property 5.

Question 3. (Exercise 13.1-6 Cormen et al.)

[Estimate: 5 minutes] What is the largest possible number of internal nodes in a RBT with black-height k? What is the smallest possible number?

To get the largest possible number of internal nodes, for every bh(x) of k, there will be red nodes between every black node. For example, if there is a k of 1, the root will be black, the children will be red, and the red nodes will have two black nodes.As there is a limit to how many black nodes we can have, this alternating pattern of black and red will give us the most nodes. Even with a k of 1, the number of nodes is root(1) + red(2) + black_for_each_red(4) = 7. This is 1 + 2 + 4 and we can see that as k increases, the summation series will continue. With k of 2, we will get two more layers of red and black: 1 + 2 + 4 + 8 + 16 = 31. The equation for the largest possible number of nodes with black-height k is therefore, $\sum_{0}^{2k} 2^{k}$.

To get the smallest possible number of internal nodes, we don't include any red nodes. If there is a k of 1, the tree will be a root with one black leaf. This makes 2 nodes and this pattern can be generalized

for all k. If k is 2, the tree is a root with one black child which has one black leaf; 3 nodes. The formula then is simply k+1.

Question 4. Left-Right Balance

[Estimate: 7 minutes] We can define the left-right balance of a binary search tree (BST) as the ratio of the number of nodes in the left sub-tree to the number of nodes in the right. Answer the following questions:

1. If a BST can be successfully colored red-black, then find bounds on the left-right balance of that tree, or prove why there is no bound on the left-right balance of that tree.

There is no bound on the left-right balance of the tree. If we imagine a scenario where the left subtree has the smallest possible nodes and the right subtree has the largest, we can see how the left subtree would grow linearly while the right subtree would grow exponentially. The ratio, left_nodes : right_nodes, then has the potential to reduce to 0 because right_nodes will exponentially grow. More specifically, the most unbalanced scenario would have a ratio of n : $\sum_{0}^{2n-1} 2^n$. This the "upper bound" of the ratio. This is technically a bound but as n gets larger, the unbalance can potentially increase so much that I said that there is no bound.

2. Does your answer change as the number of nodes in the tree increases? Explain your answer.

My first answer already considers the scenario where the number of nodes increase and so the answer doesn't change.

Question 5. From Binary Search Trees to Red-Black Trees

[Estimate: 5 minutes] Consider the Node class you have worked on for the previous lessons on BSTs. Describe how you could start from that class to adapt it to be applicable to RBTs. Enumerate what different attributes and methods you would need, and any other changes you would need to make to the code.

The main difference we need to make is assigning colors to each node. The five properties have to be met and so we have to implement a function that can make sure that the colors follow the five property rule. Red nodes have to have black children and the root node has to be black for instance. The bh(x) has to be equal for any node to leaf path and this can also be checked with a function or global variable that keeps track of these.

[Challenge] Implement these changes in Python, and clearly demonstrate that your code works by applying it to a few test cases.