Question 1. (Exercise 13.2-1, Cormen et al.)

[Estimate: 3 minutes] Write pseudocode for RIGHT-ROTATE.

```
RIGHT-ROTATE(T, x):
    x = y.left               // set x
    y.left = x.right         // y's left node is now the right node of x
    if x.right != T.nil      // if the right node of x is not NIL, its parent is y
        x.right.p = y
    x.p = y.p                // parent of x changes to parent of y
    if y.p == T.nil          // if the parent of y is NIL, the root is now x
        t.root = x
    elseif y == y.p.right    // if y is the right node of its parent, the right node is now x
        y.p.right = x
    Else                     // else (y is left node), the left node is now x
        y.p.left = x
    x.right = y              // right node of x is set to y
    y.p = x                  // parent of y is now x
```

Question 2. (Exercise 13.2-2, Cormen et al.)

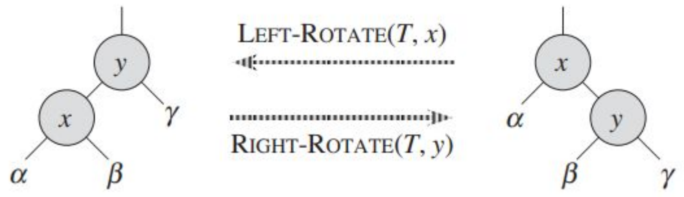
[Estimate: 3 minutes] Argue that in every n-node binary search tree, there are exactly (n−1) possible rotations.

Not sure, but I'm guessing that any node can be rotated either to the left or right as long as it has children or parents. This is n-1 because the root node doesn't count.

Question 3. (Exercise 13.2-3, Cormen et al.)

[Estimate: 4 minutes] How do the depths of nodes in a BST change when a rotation is performed? Explain your answer.

The depth seems to remain the same looking at this visualization.



However, we can see how rotating would make room for more nodes if one side is unbalanced. For example, if we need to add more to the left node of x (a), rotating it and then adding nodes would keep the tree balanced and end up with a smaller depth.

Question 4. (Exercise 13.3-2, Cormen et al.)

[Estimate: 5 minutes] Write down or illustrate the red-black trees that result after successively inserting the keys 41; 38; 31; 12; 19; 8 into an initially empty red-black tree.

41 - This key is inserted as the root node and is black

38 - This key is inserted to the left of 41; node is red

31 - This key goes to the left of 38 and is black. However, left subtree has more black nodes so rotate right and make 38 the root and 41 its right node. Color 38 black, 41 red, and 31 red. Now the tree is balanced.

12 - This key goes to the left of 31. Structure of the tree doesn't change, only the colors. The top three nodes are now black and this key is red.

19 - Goes to the right of 12. Tree is unbalanced again. Left rotate to make 19 the parent of 12 and then right rotate to make 31 right node of 19. This way, the tree is balanced because 19 has two nodes instead of a linked-node.

8 - Inserted to the left of 19. Structure doesn't need to change. Colors change to keep bh(x) balance.