LLRBs

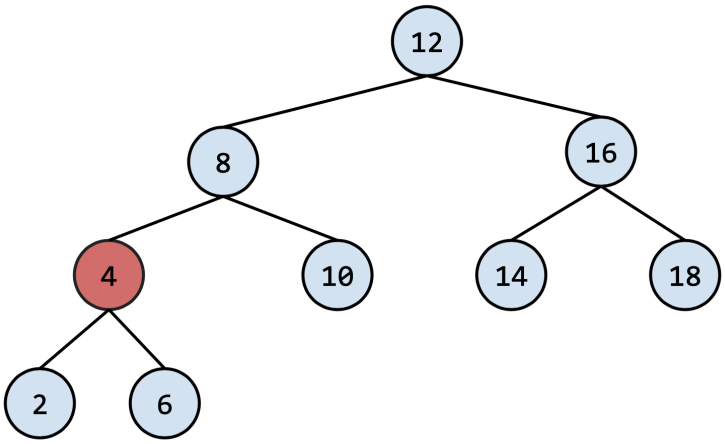
Here is a video walkthrough of all parts of this problem.

a) (2 Points). Perform the following insertions on the Left Leaning Red Black Tree (LLRB) given below. For each insertion, give the fix up operations needed. Recall a fix up operation is one of the following:

- `rotateLeft`
- `rotateRight`
- `colorFlip`
- change the root node to black.

Note that insertions are **dependent**. If only two operations are necessary, pick "None" for the third operation. If only one operation is necessary, pick "None" for the second and third operation. If no operations are necessary, pick "None" for all three operations.

If you put "None" for the "Operation applied", **leave the "Node to apply on" blank.** (Summer 2021 MT2)



i) (0.5 Points). Insert 17

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

Solution:

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black √ **None**	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black √ **None**	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black √ **None**	

Explanation: 17 is inserted as the left child of 18. No fixes are required at this point.

ii) (0.5 Points). Insert 15. Note that insertions are dependent, so insert 15 into the state of the LLRB after the insertion of 17.

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

Solution:

	Operation applied	Node to apply on
1st operation	√ **`rotateLeft()`** ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	14
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black √ **`None`**	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black √ **`None`**	

Explanation: 15 is inserted as the right child of 14. This requires a left rotation of 14 to maintain the left-leaning invariant.

iii) (0.75 Points). Insert 13. Note that insertions are dependent, so insert 13 into the state of the LLRB after the insertion of 15.

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

Solution:

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` √ **`rotateRight()`** ◯ `colorFlip()` ◯ change root to black ◯ `None`	15
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` √ **`colorFlip()`** ◯ change root to black ◯ `None`	14
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black √ **`None`**	

Explanation: 13 is inserted as the left child of 14. This requires a right rota-

tion on 15, since you cannot have 2 left red nodes in a row; then you must color flip 14 to break up the 4-node.

iv) (0.75 Points). Insert 19. Note that insertions are dependent, so insert 19 into the state of the LLRB after the insertion of 13.

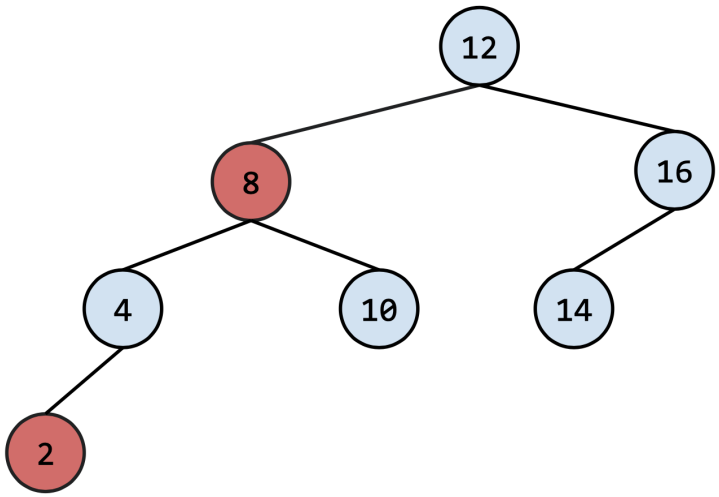
	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

Solution:

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` √ **`colorFlip()`** ◯ change root to black ◯ `None`	18
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` √ **`colorFlip()`** ◯ change root to black ◯ `None`	16
3rd operation	√ **`rotateLeft()`** ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	12

Explanation: 19 is inserted as the right child of 18. This requires a color flip on 18 to break up the 4-node, then a color flip on 16 which not has 2 red children. After this, a left rotation on 12 is required since it has a red right child.

b) (1.5 Points). The tree below is **not** a valid LLRB (hint: to see why this is the case, draw the corresponding 2-3 tree) but it's close! In this part, we will try to *transform* it into a valid LLRB in two different ways. Note that each way acts **independently** of the previous. If a way isn't possible, put `impossible`. Recall that LLRBs **cannot** have duplicates.



i) (0.75 Points). Way 1: Remove a **single leaf** node from the tree. Which leaf node?

◯ 2 ◯ 4 ◯ 8 ◯ 10 ◯ 12 ◯ 14 ◯ 16 ◯ impossible

Solution:

◯ 2 ◯ 4 ◯ 8 ◯ 10 ◯ 12 √ **14** ◯ 16 ◯ impossible

Explanation: A LLRB always has the same "black height" (number of black nodes from root to leaf). Note that the left child has a "black height" of 2 but the right has a black height of 3; thus deleting 14 makes this a valid LLRB.

ii) (0.75 Points). Way 2: Flip the color of a **single** node. Which node?

◯ 2 ◯ 4 ◯ 8 ◯ 10 ◯ 12 ◯ 14 ◯ 16 ◯ impossible

Solution:

◯ 2 ◯ 4 ◯ 8 ◯ 10 ◯ 12 √ **14** ◯ 16 ◯ impossible

Explanation: Like above, flipping 14 decreases the black height of the right child by 1, making it valid.