Jeremy Venne
11/21/19
Algorithms

5)

```
        NODE:
 Key: 3
 Colour: Black
 Parent: 4
 Right Child: 9
 Left Child: 2


Left:

         NODE:
 Key: 2
 Colour: Red
 Parent: 3
 There is no right child of the node.
 There is no left child of the node.


Right:

         NODE:
 Key: 9
 Colour: Black
 Parent: 3
 Right Child: 16
 Left Child: 4


Left:

         NODE:
 Key: 4
 Colour: Red
 Parent: 9
 There is no right child of the node.
 There is no left child of the node.


Right:

         NODE:
 Key: 16
 Colour: Red
 Parent: 9
 There is no right child of the node.
 There is no left child of the node.
```

Looking at my example below, you can see that the tree is self balancing. Each left child is less than its parent, while a right child is greater than its parent. At each node, the two sub tree heights are kept as equal as possible. This eliminates any chance of the tree being one long branch.

Black red trees' heights are maintained at O(logn). This means that all of the tree's main operations have a time complexity of O(logn). This is an improvement to the basic balanced binary trees. For example, inserting an element in a binary search tree can have a worst case of O(n). Due to the additional organization of red and black trees, each operation is guaranteed to operate at O(logn).

This is further verified by the following proof:

$n \geq 2^{BH(root)} - 1$

$n \geq 2^{h/2} - 1$

$\log_2(n+1) \geq \log_2 2^{h/2}$

$\log_2(n+1) \geq h/2$

$h \leq 2\log_2(n + 1)$

Looking at my example tree above, you can see that the height of the tree falls within this range.