1 Problem 12.3-2 on page 299

According to the algorithm TREE-INSERT(T, z) (page 294) and ITERATIVE-TREE-SEARCH(x, k) (page 291), in inserting case, we examine all the nodes along the path till the new parent node, while we examine all the nodes till the last correct one or find no match in searching case. The path is exactly same, and there is one more examination for judging the found node is matching or not.

2 Problem 13.1-5 on page 312

2.1 Largest ratio

Because a red key must be a child or a black key, the upper bound of the ratio is 2. The ratio becomes 2 if and only if every black key in the tree has two red children. This is satisfied in any complete red-black tree with black keys at the odd layer and red keys at the even layer (Figure 1 for an example).

![Figure 1: Tree with largest ratio, height = 3](image)

In fact, considering the red-black tree’s property, any such complete red-black tree’s height is odd (because all leaf nodes must be black). Then, for any such tree, we have

\[
\frac{\text{Red internal node #}}{\text{Black internal node #}} = \frac{\sum_{k \text{ is odd, } k < \text{height}} 2^k}{\sum_{k \text{ is even, } k < \text{height}} 2^k} = \frac{2^1 + 2^3 + \ldots + 2^{\text{height}-1}}{2^0 + 2^2 + \ldots + 2^{\text{height}-2}} = \frac{2 \cdot (2^0 + 2^2 + \ldots)}{2^0 + 2^2 + \ldots} = 2
\]

2.2 Smallest ratio

We want least number of red keys and most number of black keys. Simply, we can have a tree with only one node, which is black. Then, the ratio is 0.
3 Problem 13.3-4 on page 322

According to the RB-INSERT(T,z) (page 315), RB-INSERT-FIXUP only modifies the child which is already colored red, and it never change the status of a child which is T.nil. That is, we only need to modify when there are two RED nodes in a parent-child relation. In this case, we know the parent of current node must not be the root, and its grandparent have to be black and cannot be T.nil.

Thus, it is impossible to modify T.nil to be red.

4 Problem 13.4-6 on page 330

At begining (line 3) in algorithm, w has be set to the sibling of x. Thus, if w.color == red, the parent of x (also the parent of w) cannot be red, otherwise property 4 is not satisfied. Thus, their concern is unnecessary.

5 Problem 14.2-2 on page 347

Yes. Because the black-height of a node can be computed by the black-height of a red child, or the black height of a black child plus one. According to Theorem 14.1 (page 346), insertion and deletion can be still performed in O(lg n) time.

As to the depths of nodes, the answer is "No". Because the depth of a node depends on the depth of its parent. So if a depth of a node has changed, all nodes below this node have to update their depth. Suppose the root has been deleted, then all the nodes have to update their depth, which will cost O(n) time.