1. Problem 12.3-2 on page 299.

While inserting an element into a binary search tree we compare an element with n other elements to insert it in the right position. So when searching for the same element we have to compare the search value with those n elements that we compared with previously and the element itself. Thus, we will make n+1 comparisons while searching.

This property makes a connection between the cost of successful and unsuccessful searches because a successful search will take (n+1) comparisons since the value had been inserted whereas an unsuccessful search will take just (n) comparisons since the value had not been inserted so there will be no final value to compare with. This connection is the depth of the tree that the search function will have to traverse to find the value according to the insertion path.

Citations: 1) http://www.transtutors.com/questions/suppose-that-we-construct-a-binary-search-tree-by-repeatedly-inserting-distinct-valu-677206.htm, Author: Shubham G.,

Title: Suppose We Construct a BST, Date: June 11, 2015

2. Problem 13.1-5 on page 312

One of the red-black tree properties is that, every simple path from a node(x) to a descendent leaf has the same number of back nodes and red nodes do not occur immediately next to each other on such paths. Therefore the shortest possible simple path will have all black nodes and the longest possible simple path will have alternating black and red nodes. Also, since the leaf must be black there are at most the same number of red nodes as black nodes on the longest simple path making it twice as long.

Citations: 1) https://www.cs.duke.edu/courses/cps130/summer02/Homeworks/Solutions/H12-solution.pdf, Author: None, Title: CPS 130 HW12-Solutions, Date: July 2002

3. Problem 13.3-4 on page 322

First of all, z is RED. At the beginning of RB-Insert-Fixup , the only violation of the Red-Black conditions could be that we have two RED nodes in a parent-child relation. If z.p.color = BLACK, we are done (the loop is skipped) and nobody is set to RED. If z.p.color = RED, then we know z.p.p = BLACK and z.p.p cannot be T.nil, since the root (root.color = = BLACK) must be an ancestor of z.p, and the nearest ancestor of z.p is z.p.p. The first IF simply checks whether z's parent is a left or right child pf z's grandparent (z.p.p), and sets to color. The first branch of the inner IF sets the color of the uncle and parent of z to BLACK, moving the RED up to the grandparent z.p.p. Since none of these nodes was T.nil, we are safe. The second branch of the inner IF involves some rotations, and you must make sure that, at the end of the rotations, z.p.p ≠ T.nil so it can be safely set to RED.

Citations: 1) http://www.cs.uml.edu/~giam/91.404/OldExams/Exam2S10HintsAndSolns.pdf
Author: Giampiero Pecelli, Title: Algorithms - 91.404 - Spring '10 - Exam 2,

Date: April, 13, 2010.

4. Problem 13.4-6 on page 330

Case 1 occurs only if (x)'s sibling (w) is red. If p[x] were red, then there would be two reds in a row, namely p[x] (which is also p[w]) and (w), and we would have had these two reds in a row even before calling RB-DELETE.

Citations: 1) https://ipfs.io/ipfs/QmTmMhRv2nh889JfYBWXdxSvNS6zWnh4QFo4Q2knV7Ei2B/Algorithms/Introduction%20to%20Algorithms-Cormen%20Solution.pdf Author: Cormen, Lee, Lin, Title: Instructor's Manual, Date: 2002

5. Problem 14.2-2 on page 347

Yes, we can maintain black-heights as attributes in the nodes of a red-black tree without affecting the asymptotic performance of the red-black tree operations. This is because the black-height of a node can be found from the information at the node and its two children or even just from one child since the black-height of a node is the black-height of a red child, or the black-height of a black child plus one. Then the second child does not need to be checked because of property 5 (paths from the root to any leaf all pass through the same number of black nodes). Thus, according to Theorem 14.1 insertion and deletion operations can be still performed in O(lg n) time.

Now, maintaining the depths of nodes cannot be done without affecting the asymptotic performance of the red-black tree operations. This is because the depth of a node depends on the depth of its parent. So when the depth of a node changes, the depths of all nodes below it in the tree must be updated. Therefore, updating the root node causes n-1 other nodes to be updated, which would mean that operations on the tree that change node depths might not run in O(n lg n) time.

Citations: 1) http://ripcrixalis.blog.com/2011/02/08/clrs-14-2-how-to-augment-a-data-structure/, Author: Rip, Title: CLRS 14.2 How to augment a data structure, Date: February 8, 2011.

2) http://s3.alirezaweb.com/91-5/introduction-to-algorithms/solution-manual/3th-edition-exercises-and-problems/chap14-solutions-www.alirezaweb.com.pdf , Author: None, Title: Selected Solutions for CH 14, Date: None.

3) Lecture 8 Notes, Author: Edward M. Reingold, Date: February 8, 2016.