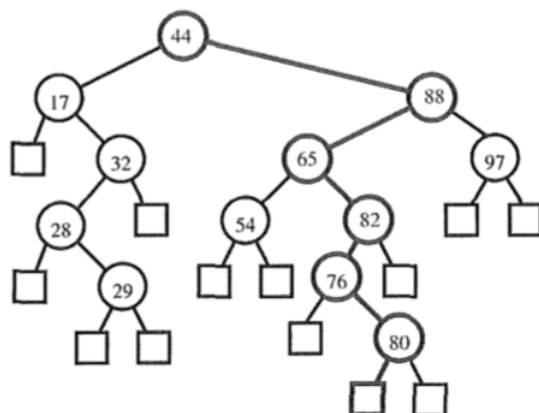
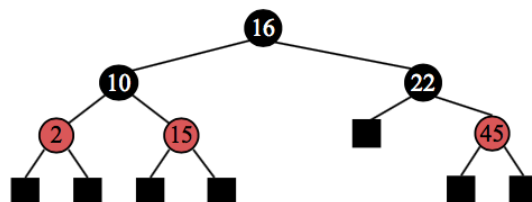


Homework 3 (70pts)

1. (5 points) Remove from the binary search tree in Figure 3.7(a) the following keys (in this order): 32, 65, 76, 88, 97. Draw the tree after **each** removal.



2. (10 points) Let T be a binary search tree, and let x be a key. Give an efficient algorithm for finding the smallest key y in T such that $y > x$. Note that x may or may not be in T . Explain why your algorithm has the running time it does.
3. (5 points) Consider the following sequence of keys: (18, 30, 50, 12, 1). Insert the items with this set of keys in the order given into the red-black tree in the figure below. Draw the tree after **each** insertion.



4. (10 points) Design and give the **pseudocode** for an $O(\log n)$ algorithm that determines whether a red-black tree with n keys stores any keys within a certain (closed) interval. That is, the input to the algorithm is a red-black tree T and two keys, l and r , where $l \leq r$. If T has at least one key k such that $l \leq k \leq r$, then the algorithm returns true, otherwise it returns false. *Hint:* You can use the TreeSearch algorithm (page 146) as a subroutine.
5. (a) (5 points) Draw the merge-sort tree for an execution of the merge-sort algorithm on the input sequence: (2, 5, 16, 4, 10, 23, 39, 18, 26, 15) (like in Figure. 4.2).
- (b) (5 points) Draw the quick-sort tree for an execution of the quick-sort algorithm on the input sequence from part (a) (like in Figure 4.12). Use the last element as the pivot.
- (c) (3 points) What is the running time of the version of quick-sort that uses the element at rank $\lfloor \frac{n}{2} \rfloor$ as the pivot, provided that the input sequence is already sorted? Explain.

6. (10 points) Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented by an array, give a linear-time **in-place** algorithm for ordering S so that all the blue elements are listed before all the red elements. What is the running time of your method?
7. (10 points) Let A and B be two sequences of n integers each. Give an integer m , describe an $O(n \log n)$ time algorithm for determining if there is an integer a in A and an integer b in B such that $m = a + b$.
8. (a) (5 points) Suppose we are given a sequence S of n elements, each of which is an integer in the range $[0, n^2 - 1]$. Describe a simple method for sorting S in $O(n)$ time. [*Hint*: think of alternative ways of viewing the elements].
(b) (2 points) Does the running time of radix-sort depend on the order of keys in the input? Explain.