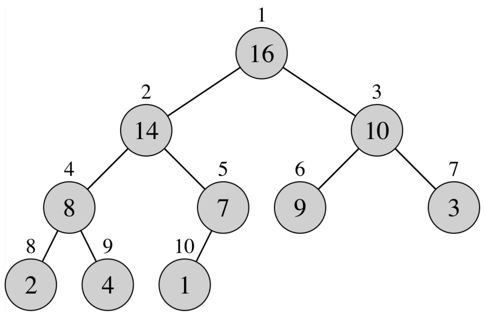
CSCI 200

HW #2: Heaps, heapsort, mergesort, quicksort, Binary Search, BSTs, and RBTs

Directions: *Complete your work on a separate sheet of paper. Submit the physical copy of your work at the beginning of class on the specified due date. Show your work. You may work in groups of up to 3 students provided that all students participate in each question. Provide a short preliminary explanation of how an algorithm works before running an algorithm or presenting a formal algorithm description, and use examples or diagrams if they are needed to make your presentation clear.*

1. Let T be a (max) heap storing n keys. Give the **pseudocode** for an efficient algorithm for printing all the keys in T that are greater than or equal to a given query key x (which is not necessarily in T). You can assume the existence of a $O(1)$-time *print(key)* function. For example, given the heap below and query key $x = 6$, the algorithm should report $16, 14, 10, 8, 7, 9$. Note that the keys do not need to be reported in sorted order. **Analyze the run time of your algorithm**. It should run in $O(k)$ time, where k is the number of keys reported.



2. Consider the following arrays. Indicate (yes/no) whether they are or are not a max-heap.
   (a) $[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$
   (b) $[50, 25, 28, 8, 3, 17, 14, 6, 5, 1, 2]$
   (c) $[4, 8, 10, 9, 12, 16, 18, 22, 40]$
   (d) $[15, 14, 13, 10, 12, 9, 8, 5, 7, 3, 2]$
   (e) $[12, 5, 13, 4, 2, 6, 10, 1, 3]$

3. (a) Illustrate the execution of the Build-Max-Heap algorithm on the following array: $[1, 5, 8, 10, 12, 3, 6, 13, 4, 2, 7, 9, 15, 10, 24, 28, 11]$. Show the resulting heap (as a tree) as each level of the tree is processed.
   (b) Illustrate (using a tree) the execution of heapsort on the same array as above. Use your max-heap data structure constructed above as your starting point. Use Figure 6.4 as a model, showing separately the resulting max-heap after each individual item is removed.

4. Illustrate the execution tree of mergesort on the following array: $[1, 5, 8, 10, 12, 3, 6, 13, 4, 2, 7, 9, 15, 10, 24, 28, 11]$.

5. Using the last element as pivot, illustrate the execution tree of quicksort on the following array: $[5, 8, 2, 4, 3, 7, 6, 9, 10]$.

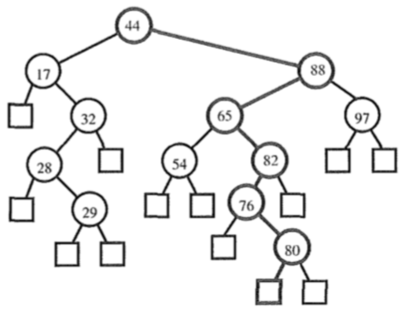
6. Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented by an array, give a linear-time **in-place** algorithm for ordering S so that all the blue elements are listed before all the red elements. What is the running time of your method?

7. 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. Suppose you are given the following sorted array: $A = [1, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584]$. Illustrate the execution of binary search for the number 148.

9. Insert into an initially empty binary search tree items with the following keys (in this order): 30, 40, 23, 58, 48, 26, 11, 13. Draw the single resulting tree after all insertions have been performed.

10. Remove from the binary search tree given below the following keys (in this order): 32, 65, 76, 88, 97. Draw the tree after **each** successive removal (5 trees total).



11. Give an algorithm that runs in $O(\log n)$ time which takes a sorted array A and two keys, x and z, which may or may not be elements of A. The algorithm should return the number of elements y in A which satisfy $x \leq y \leq z$.

12. Suppose that we have the numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. For each of the following sequences, indicate yes/no whether it could be the sequence of nodes examined in searching for the number 363. If not, explain.

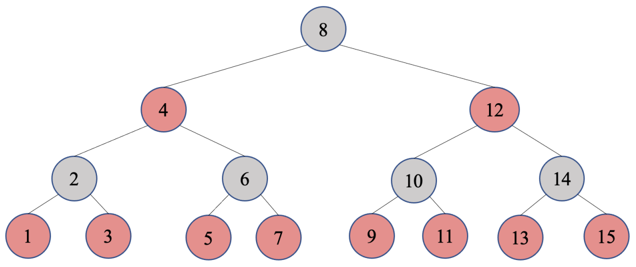
    (a) 2, 252, 401, 398, 330, 344, 397, 363
    (b) 924, 220, 911, 244, 898, 258, 362, 363
    (c) 925, 202, 911, 240, 912, 245, 363
    (d) 2, 399, 387, 219, 266, 382, 381, 278, 363
    (e) 935, 278, 347, 621, 299, 392, 358, 363

13. 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.

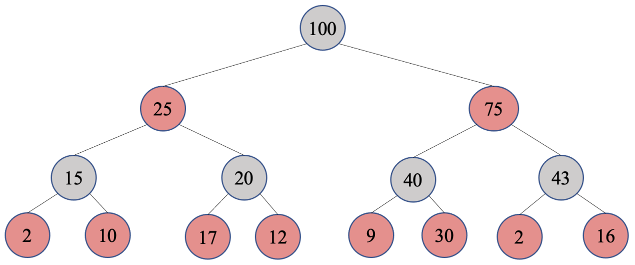
14. Design and give the **pseudocode** for an $O(\log n)$ algorithm that determines whether a red-black tree with n keys stores any keys within a certain (closed) interval. That is, the input to the algorithm is a red-black tree T and two keys, l and r, where $l \leq r$. If T has at least one key k such that $l \leq k \leq r$, then the algorithm returns true, otherwise it returns false. *Hint:* You can use the recursive or iterative TREE-SEARCH algorithm (CLRS 12.2) as a subroutine.

15. The NIL black leaf is omitted from the visualization in each of the trees shown below. For each tree, specify whether it is a red black tree (yes) or it is not a red black tree (no). If not, explain.

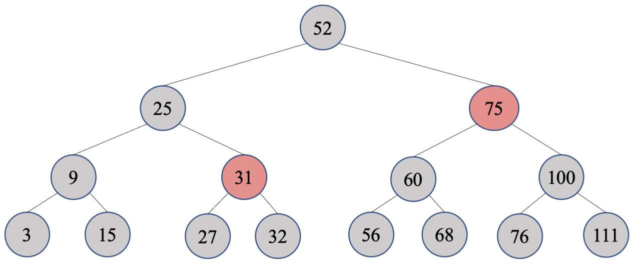
(a)



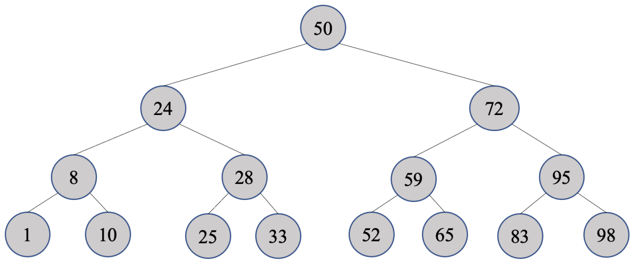
(b)


100
25
75
15
20
40
43
2
10
17
12
9
30
2
16

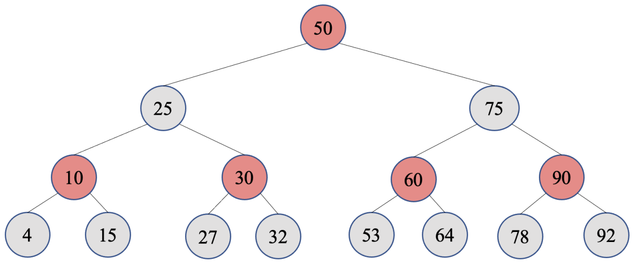
(c)


52
25
75
9
31
60
100
3
15
27
32
56
68
76
111

(d)


50
24
72
8
28
59
95
1
10
25
33
52
65
83
98

(e)


50
25
75
10
30
60
90
4
15
27
32
53
64
78
92