Joshua Tanner, Colin Robinson

4.1 a. A
b. G, H, I, L, M, K

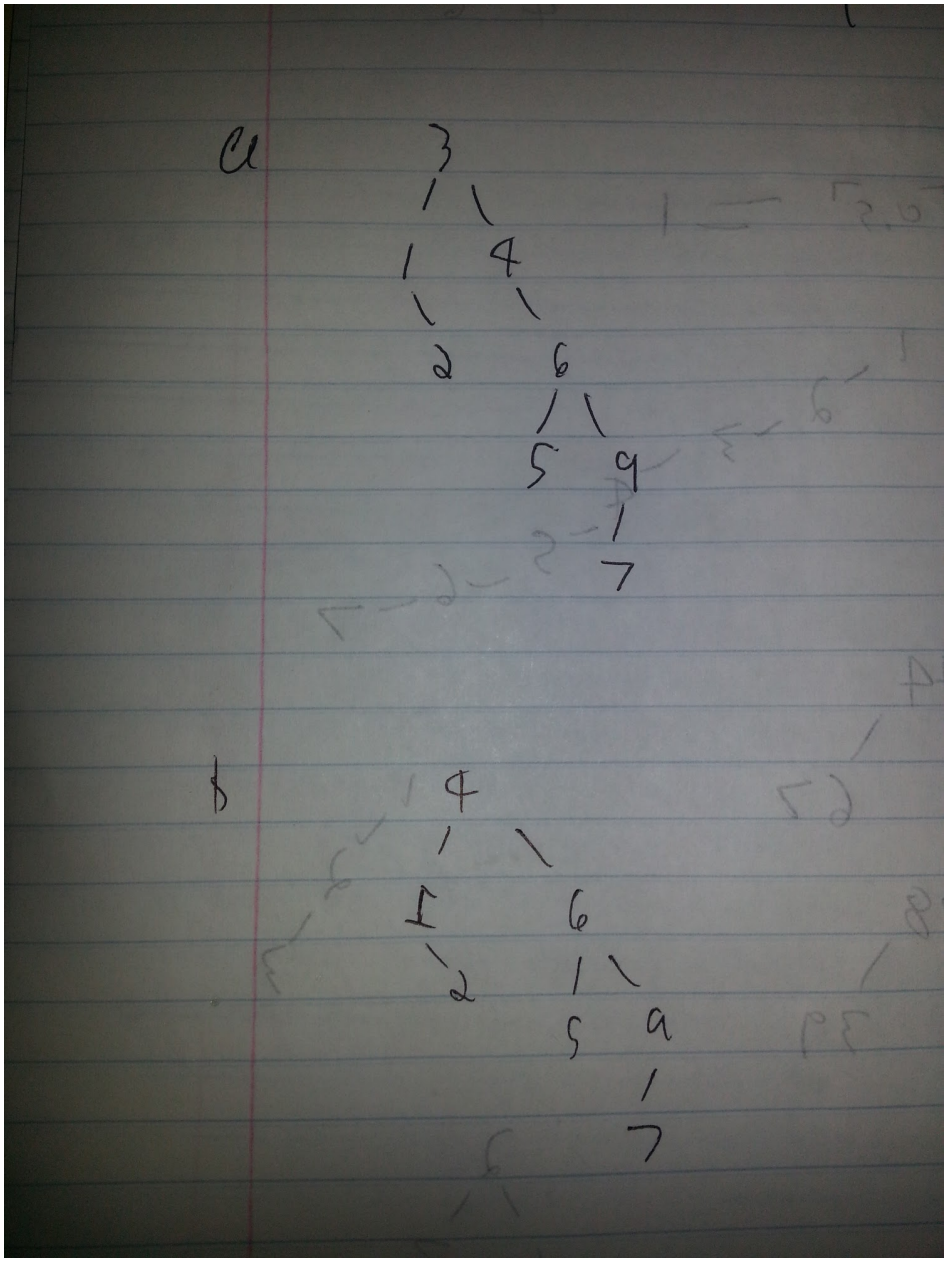
4.2 we're doing these by node:
G: parent D, no children, Siblings H, height 0, depth 3.
H: parent D no children, siblings G, height 0, depth 3.
D: parent B, children G,H, siblings E, height 1, depth 2.
B: parent A, children D,E, siblings C, height 2, depth 1.
E: parent B, children I,J, siblings D, height 2, depth 2.
I: parent E, children none, siblings J, height 0, depth 3.
J: parent E, children L, M, siblings I, height 1, depth 3.
L: parnet J, no children, Sibling M, height 0, depth 4.
M: parent J, no children, sibling L, height 0, depth 4.
A: parent none, children B,C, sibling none, height 4, depth 0.
C: parent A, children F, sibling B, height 2, depth 1.
F: parent C, children K, sibling none, height 1, depth 2.
K: parent F, children none, sibling none, height 0, depth 3.

4.7
sum(i,1,M,2^-(d[i]))
= 2^0 * count[0] + 2^-1 * count[1] + 2^-2 * count[2]
= 2^0 *

the summation sum(i,1,M, 2^-(d[i])) is the loop equivalent of the recursive function leaf_score(tree), where:
- leaf_score(tree) is leaf_score(right) * leaf_score(left) if the root has two children, neither of which is a leaf.
- leaf_score(tree) is 1 if the root has two children, which are both leaves.
- leaf_score(tree) is 1 * leaf_score(child) if the root node has two children, one of which is a leaf and the other of which we call simply "child".
- leaf_score(tree) is .5 if the root node has only one child, which is a leaf.
- leaf_score(tree) is .5 * leaf_score(child) if the root node has only one child, which is not a leaf.

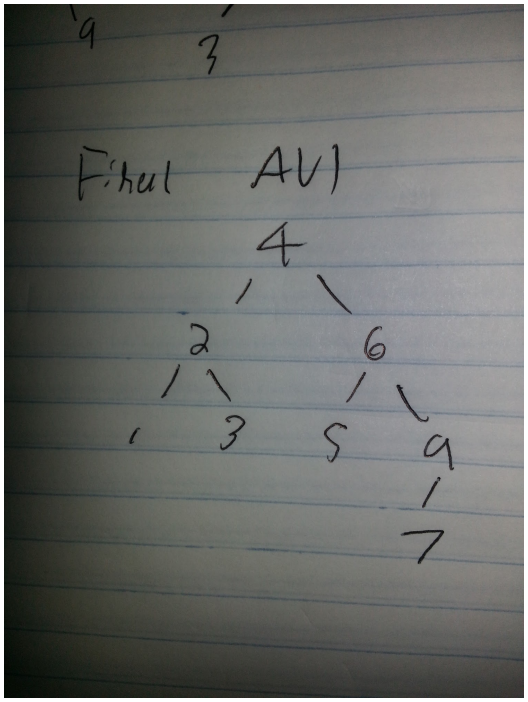
There is obviously no way that this function could return a value greater than one, so the only problem is showing that it is actually equivalent to the sum.

4.9

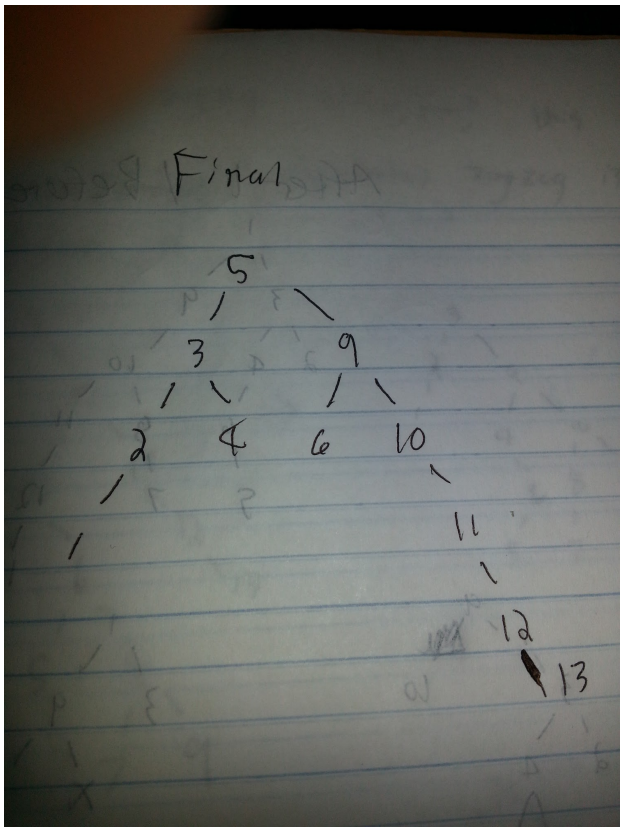


4.11, 4.26, 4.31, 4.45 Are done separately in .cpp files

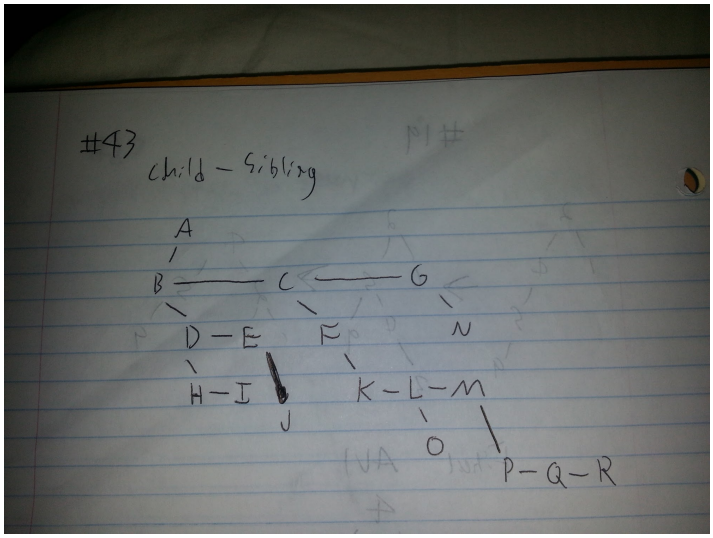
4.19 (avl)



4.27 (splay)



4.43 (child-sibling)


#43
child - sibling
A
B
C
G
D
E
F
N
H
I
J
K
L
M
O
P
Q
R