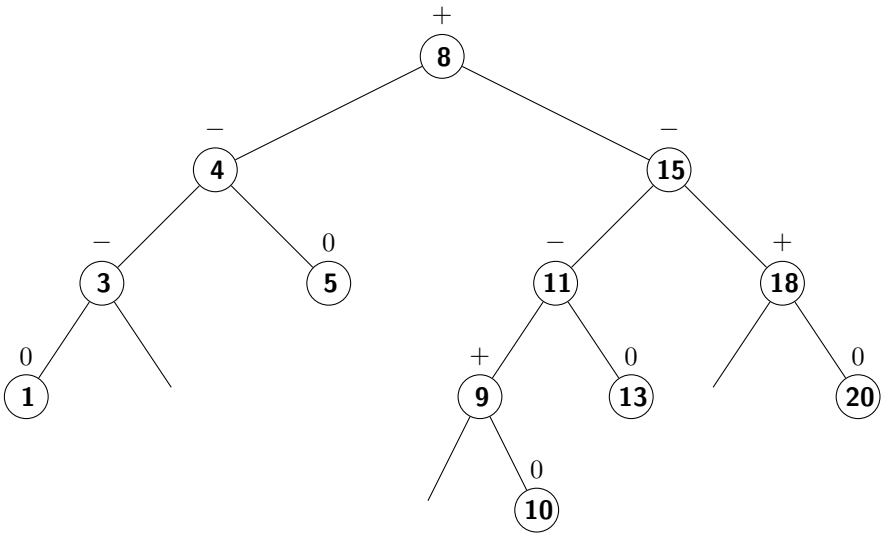
CSC263 A2

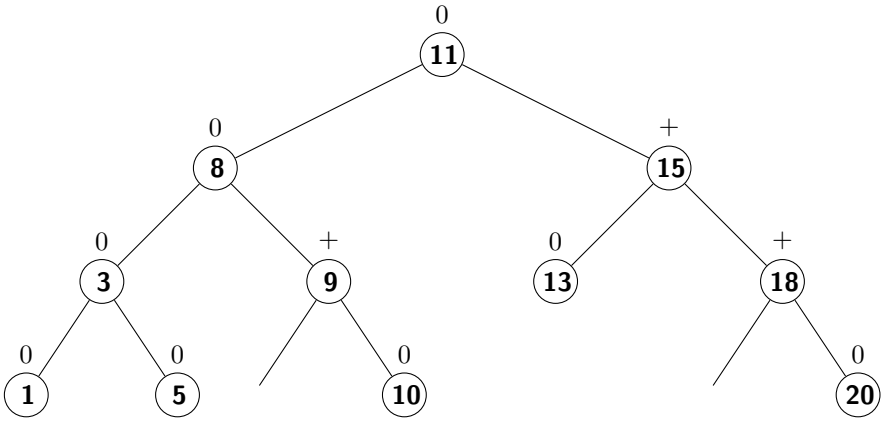
Wendy Moniuk - 996659343

February 12, 2014

1. (a)



(b)



2. (a) $m_h = m_{h-1} + m_{h-2} - m_{h-5}, h > 1$ with
$m_{-3} = m_{-2} = m_{-1} = 0, m_0 = 1, m_1 = 2$
I found the sequence to be $1, 2, 3, 5, 8, 12, 18, 27, 40$ starting from the tree of height 0 and increasing by 1. The pattern starts out like the Fibonacci sequence, but there is a correction factor needed for $h \geq 5$ that follows the Fibonacci sequence as well.

   (b) $m_0 = 1 > 1.4^0 - 1 = 0$

$m_1 = 2 > 1.4^1 - 1 = 0.4$

$m_2 = 3 > 1.96 - 1$

$m_3 = 5 > 2.744 - 1$

$m_4 = 8 > 3.8416 - 1$

Assume that it is true for some m_h where $h > 4$

Then $m_{h+1} = m_h + m_{h-1} - m_{h-5} \geq 1.4^h - 1 + 1.4^{h-1} - 1 - 1.4^{h-5} + 1 = 1.4^h(1 + 1.4^{-1} - 1.4^{-5}) - 1 > 1.4^h(1 + 0.7 - 0.18) - 1 > 1.4^h(1.4) - 1 = 1.4^{h+1} - 1$

3. (a) An AVL tree supports all of these operations, all in $O(\log n)$ time..

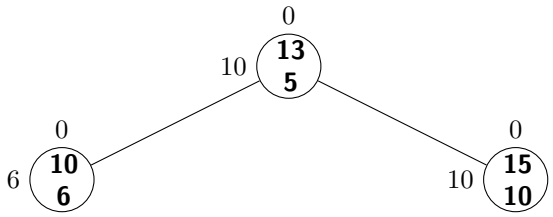
IsEmptyRange(S, a, b)

(b)1 **if** S is empty
then
2 **return** TRUE
3 **if** $S.root.key < a$
then
4 **return** IsEmptyRange$(S.right, a, b)$
5 **if** $S.root.key > b$
then
6 **return** IsEmptyRange$(S.left, a, b)$
7 **return** FALSE

This runs in $O(\log n)$ time because it traverses at most the height of the tree and takes a constant time at each node

4. (a) The data structure I will use is an AVL tree with the key being the price and each node containing the maximum floor area of its sub-tree. It can contain non-distinct nodes. For each node the right sub-tree contains prices strictly greater than itself and the left sub-tree has prices equal to or less than itself.

The diagram below shows the tree after (p, s) pairs $(13, 5), (10, 6), (15, 10)$ have been added.



(b) Insert is done at first the usual way for an AVL tree. After a new node is added, or after rebalancing, all the max areas of the ancestors, as well as nodes involved in rotations are updated. The updates would take the same amount of time as the balance factor updates. For delete it is similar, done in the usual way and then max areas are updated.

For MaxArea, we start at the root and check if p is greater than the root. An empty tree will return -1. If it is not we call again on the left sub-tree. If it is we keep a variable of the maximum of the

max-area of the left sub-tree and the root's area, and call again on to the right sub-tree passing along the current maximum. This is all done in a helper function so we can pass along the current maximum. Once we reach the bottom of the tree we return the maximum over the traversal.

All of these algorithms meet the time requirements. Insert for an AVL tree takes $O(\log n)$ and there will be on the order of log(n) ancestors to update the max-area, where each update takes constant time. Delete for AVL trees also takes $O(\log n)$ time, and again there are log(n) ancestors to update.

MaxArea takes a constant time at each node, 2 comparisons, and it traverses $O(\log n)$ nodes (the height of the tree).