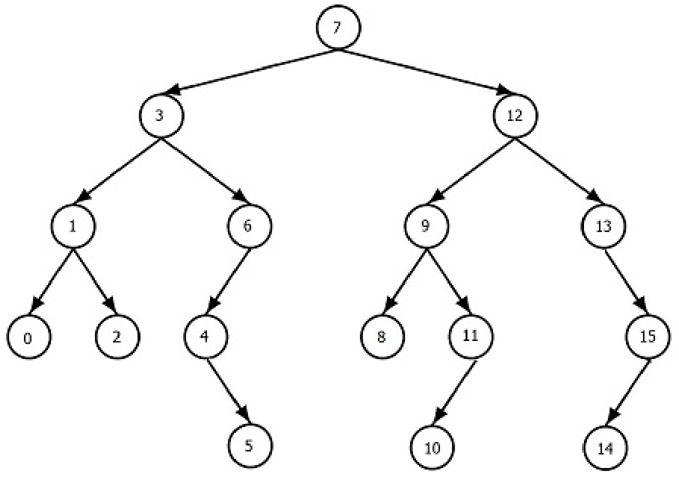
# Solutions to Assignment 2

## CS 213: Data Structures and Algorithms

1. Given a BST $T$ and an element $a$, the task is to delete all elements $b < a$ from $T$. Write a function del_less(T,a) in pseudo-code to perform this. What is the time complexity of your algorithm? Execute your algorithm for $a = 11$ on the following tree and show function calls with their input arguments.



*Solution.*

```
del_all(T)
  if T = null then
     return
  del_all(T.left)
  del_all(T.right)
  delete(T)

del_less(T,a)
  if T = null then
     return null
  if T.value >= a then
     T.left ← del_less(T.left, a)
     return T
```

```
   if T.value < a then
      new_T ← del_less(T.right, a)
      del_all(T.left, a)
      delete(T)
      return new_T
```

The time complexity of this algorithm is $\mathcal{O}(n)$ in the worst case, where $n$ is the total number of nodes in the tree. This is only because we free the memory of all the deleted nodes. If the deleted nodes are not freed by traversing all of them, then the time complexity would be $\mathcal{O}(h)$, where $h$ is the height of the BST (still can be $O(n)$ in the worst case).

When we execute `del_less(T,11)`:

(a) `del_less(T(7),11)`
$7 < 11$, so we use `del_all(T(3))` to delete the entire left subtree, and we delete the node 7. We then return `del_less(T(12),11)`.

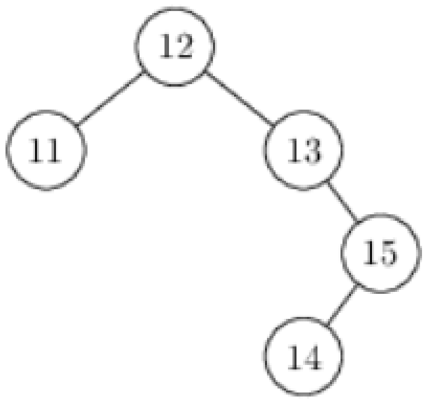
(b) `del_less(T(12),11)`
$12 >= 11$, so we replace the left subtree `T(9)` with `del_less(T(9),11)`.

(c) `del_less(T(9),11)`
$9 < 11$, so we use `del_all(T(8))` to delete the entire left subtree, and we delete the node 9. We then return `del_less(T(11),11)`.

(d) `del_less(T(11),11)`
$11 >= 11$, so we replace the left subtree `T(10)` with `del_less(T(10),11)`.

(e) `del_less(T(10),11)`
$10 < 11$, so we delete the entire left subtree (it is null so doesn't make a difference) and we delete the node 10. We return `del_less(T(10).right,11)`, but that is null, so the function calls all return one by one.

The final modified BST is:



2. We say that an AVL tree is deeply imbalanced if, at every internal node $x$, the left subtree $T_L$ and the right subtree $T_R$ are of different heights. Combining this with the AVL condition, the following holds at every internal node having subtrees $T_L$ and $T_R$:

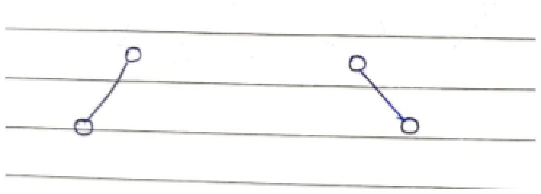
   $$|ht(T_L) - ht(T_R)| = 1$$

   (a) List all deeply imbalanced AVL tree structures of heights 1, 2, and 3.

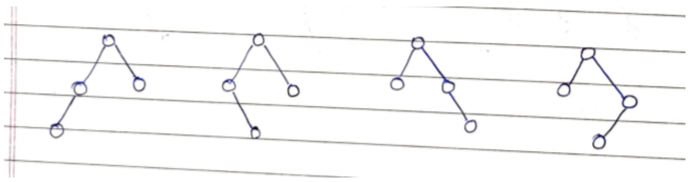
   (b) Is there a recurrence relation on the number of deeply imbalanced AVL trees having height $h$? If so, state and prove this relation. See if $h = 1, 2, 3$ satisfy it, and compute this number for $h = 4$.

   *Solution.*

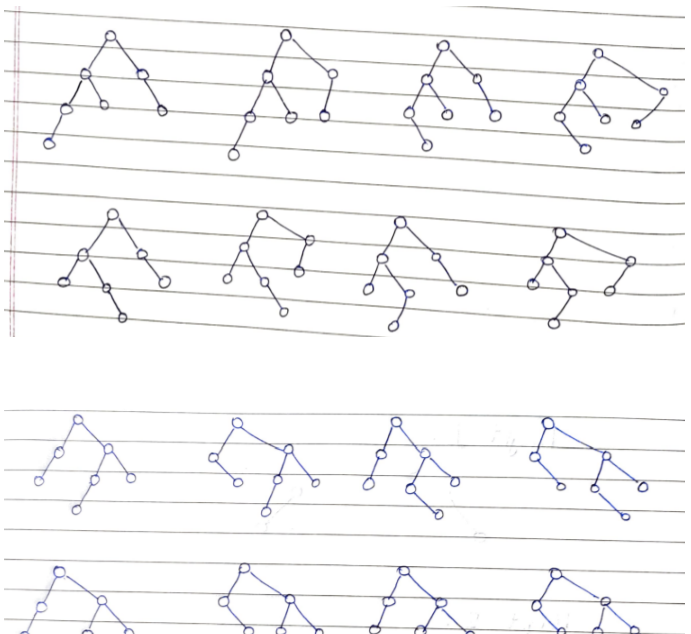
   (a) Height 1:

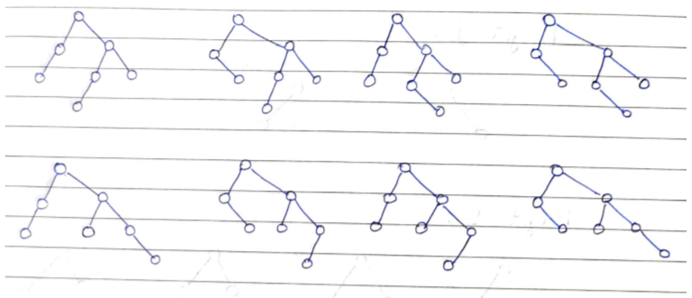


   Height 2:



   Height 3:





(b) The recurrence relation is:

$$f(h) = 2f(h-1)f(h-2)$$

The base cases are $f(0) = 1$ and $f(1) = 2$.

*Proof.* A deeply imbalanced AVL tree $T$ would have a left subtree height and a right subtree height that differ by 1. Furthermore, this difference of 1 is a consistent property throughout the tree, and holds at every internal node. Thus, the left subtree and the right subtree of $T$ also have to be deeply imbalanced AVL trees, since the same property holds in these subtrees too.

So, for $f(h)$ with $h \geq 2$, we wish to find the number of ways in which the left subtree and the right subtree can be designed to be deeply imbalanced AVL trees of height $h-1$ and $h-2$. Note that either left or right can be the deeper subtree. Thus, the number of such ways is $\binom{2}{1}$ (to select which subtree is the deeper one) multiplied by $f(h-1)f(h-2)$ (to select which deeply imbalanced AVL trees should be used).

Hence proved. $\square$

Clearly, $h = 1, 2, 3$ satisfy this relation. Also, $f(4) = 2f(3)f(2) = 128$.

*Extra.* Try solving this recursive relation! You will have noticed that $f$ always takes on values that are powers of 2. Observe these exponents. See the Fibonacci sequence?