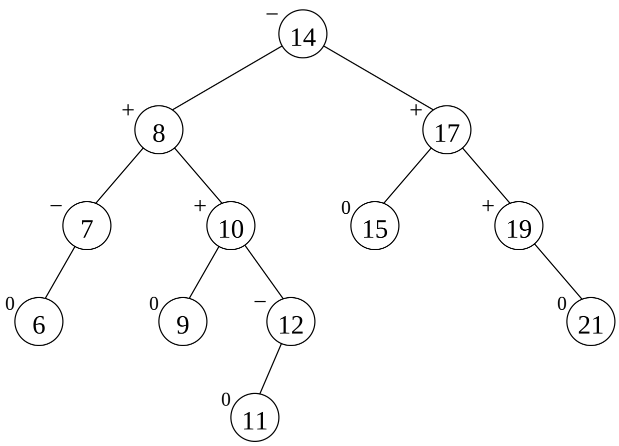
Computer Science CSC263H — February 9, 2017
St. George Campus — University of Toronto

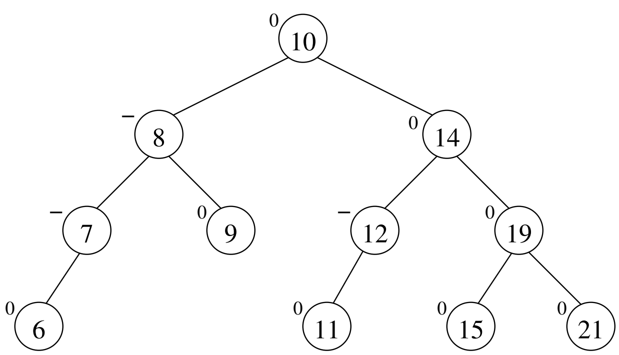
Solutions for Homework Assignment #2

**Answer to Question 1.**

- The tree resulting after the sequence of insertions is:

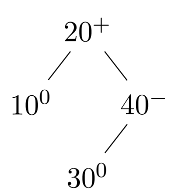



- The tree resulting after the deletion of 17 is:

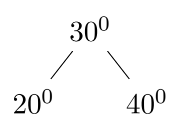



**Answer to Question 2.**

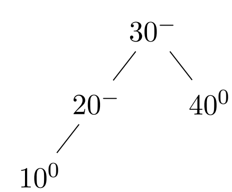
(a) We **disprove** the statement: $x = 10, T =$



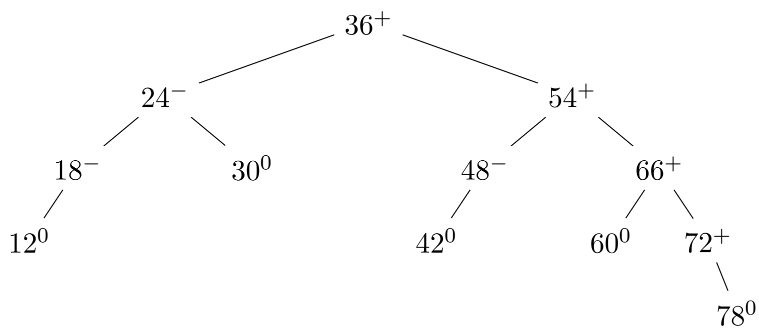

$T' =$



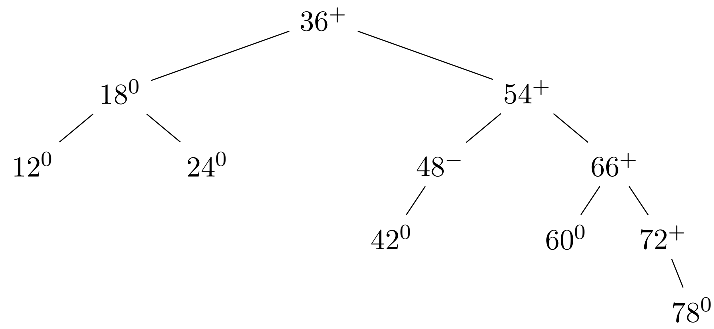

$T'' =$



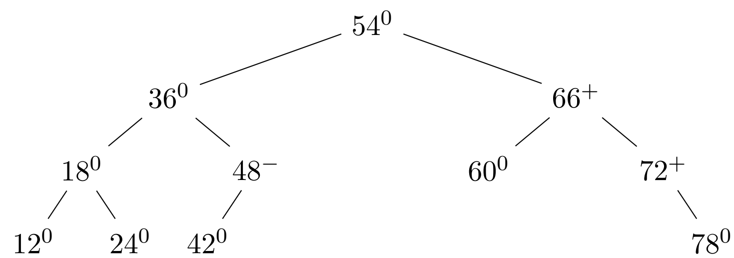

(b) $x = 30, T =$



After first rebalance: $T =$




After second rebalance: $T =$




**Answer to Question 3.** Given a (pointer to) any node $u$ of $T$, procedure $\text{Height}(u)$ does the following:

- if the subtree rooted at $u$ is *balanced* then it returns the height of that subtree;
- if the subtree rooted at $u$ is *unbalanced* then it returns UNBALANCED;

> HEIGHT($u$)
> **if** $u = \text{NIL}$ **then return** $-1$
> **else**
>     $H_L :=$ HEIGHT($lchild(u)$)
>     $H_R :=$ HEIGHT($rchild(u)$)
>     **if** $(H_L \neq \text{UNBALANCED}) \;\wedge\; (H_R \neq \text{UNBALANCED}) \;\wedge\; (|H_R - H_L| \leq 1)$
>         **then return** $\max(H_R, H_L) + 1$
>         **else return** UNBALANCED

Execute HEIGHT($r$) where $r$ is the root of $T$; return FALSE iff HEIGHT($r$) returns UNBALANCED.

The algorithm recursively visits every node of the tree $T$ exactly once, and for each node that it visits the algorithm takes constant time (to execute the if..then..else code). Since $T$ has $n$ nodes, the worst-case running time is $\Theta(n)$.

**Answer to Question 4.**

*Algorithm description.* Maintain an *AVL tree* $T$ that contains the $m$ smallest keys input so far.

- When a key input occurs, insert it into $T$, find the ***maximum*** key in $T$ and ***remove*** it.
- When a *query* occurs, print the $m$ keys in $T$ in sorted order by doing an ***inorder traversal of the tree*** $T$.

*Algorithm's worst case running time.* Since the AVL tree $T$ has $m$ keys:

- Each key insertion and each key removal takes $O(\log m)$ time in the worst-case. Finding the maximum key in $T$ also takes $O(\log m)$ time in the worst-case. Thus, the worst-case running time to process each input key is $O(\log m)$.
- Each inorder traversal of $T$ takes $\Theta(m)$ time. Thus, the worst-case running time to perform each input key is $O(\log m)$.