insert 40, 21, 5
Algorithm \textbf{putAVL} \((r, k, \text{data})\)

\textbf{In:} Root \(r\) of an AVL tree, record \((k,\text{data})\)

\textbf{Out:} \{Insert \((k,\text{data})\) and re-balance if needed\}

\(O(\text{height})\)

\{\text{put}(r,k,\text{data})\} \quad // \text{Algorithm for binary search trees}

\{\text{Let } p \text{ be the node where } (k,\text{data}) \text{ was inserted} \}

\textbf{while} \((p \neq \text{null})\) \textbf{and} \((\text{subtrees of } p \text{ differ in height } \leq 1)\) \textbf{do}

\{\text{if } p \neq \text{null then} \text{ rebalance subtree rooted at } p \text{ by}

\text{performing appropriate rotation} \}

\(O(1)\)

\[ f(n) \text{ is } O(\text{height}) = O(\log n) \]
Algorithm: remove AVL (r, K)
In: Root r of an AVL tree, key K
Out: {Remove K from the tree and rebalance, if needed}

O(height) { remove (r, K) // Same algorithm for BST
    C1 { Let p be the parent of the removed node
        while p != null do
            if subtree rooted at p is not AVL then
                Rebalance it
                P ← parent of P
    C2 x height
}
f(h) is O(height) = O(log n)