How efficient are binary search trees?

Binary search tree operations take time $O(h)$, where h is the height of the tree.

But what is the height of a binary search tree for n elements?

It depends on the insertion order!

In the best case $O(\log n)$. (Perfect binary tree)

In the worst case $O(n)$ (the tree is really a linked list).

If the insertions are in random order, then the expected height of the tree is $O(\log n)$.

Balanced search trees

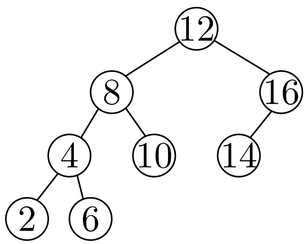
Balancing a tree means to keep the left and right subtree of every node of roughly "equal" size.

There are many kinds of balanced search trees:

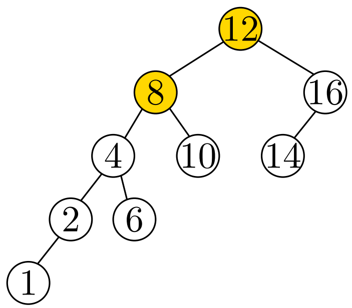
- Height-balanced trees (AVL-trees), (Adelson-Velsky and Landis, 1962);
- Weight-balanced trees (Nievergelt and Reingold, 1973);
- (a, b)-trees (Bayer and McCreight 1972);
- Red-black trees (Guibas and Sedgewick 1978);
- Splay-trees (Sleator and Tarjan 1985).

AVL-Trees

An AVL-tree is a binary search tree with an additional balance property: For every node of the tree, the height of the left subtree and the right subtree differ by at most one.



AVL-Tree



Not an AVL-Tree

AVL-Trees have logarithmic height

We ask the opposite question: For a given height h, what is the smallest number $N(h)$ of nodes an AVL-tree can have?

We have $N(0) = 1$, $N(1) = 2$, $N(2) = 4$, and
$N(h) \geq N(h-1) + N(h-2) + 1$.
So $N(h) \geq 2N(h-2)$, and induction gives us $N(h) \geq 2^{\lceil h/2 \rceil}$.

And therefore an AVL-tree with n nodes has height at most $2 \log n$.

A more careful analysis shows that $N(h) = F_{h+3} - 1$, and using the known formula for the Fibonacci numbers, we get the better bound $h \leq 1.44 \log(n+2)$.

Maintaining balance

We have to maintain the balancing condition when we insert or remove nodes in the tree.

Consider the insertion/deletion of a node w.

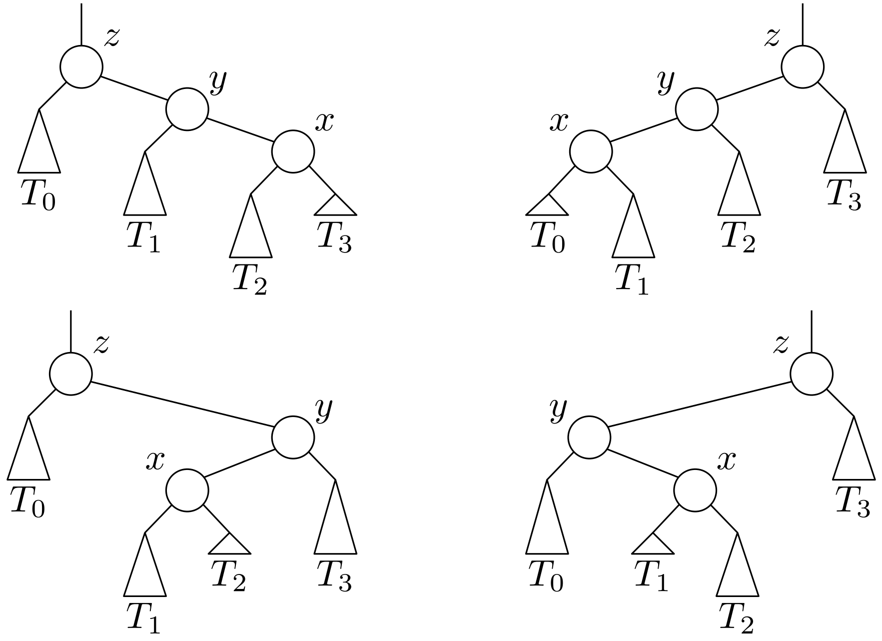
Heights change only on the path from the root to w.

Let z be the lowest ancestor of w that is now unbalanced. Let y be its child of larger height, and x the child of y of larger height (outer child in case of equal height).

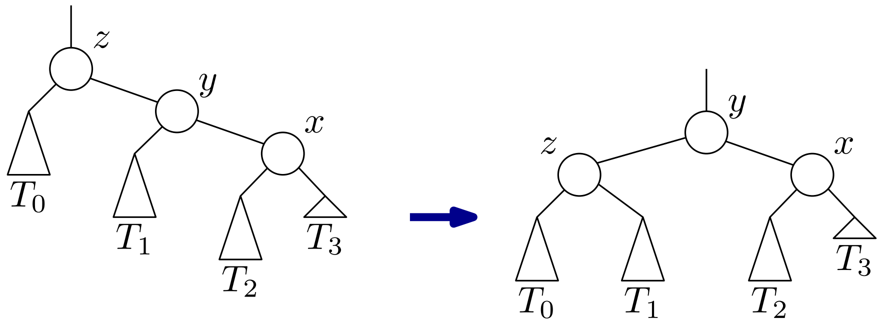
We restructure the subtree rooted at z, by moving x, y, and z and their subtrees.

There are four cases.

The four cases of restructuring



Single Rotation

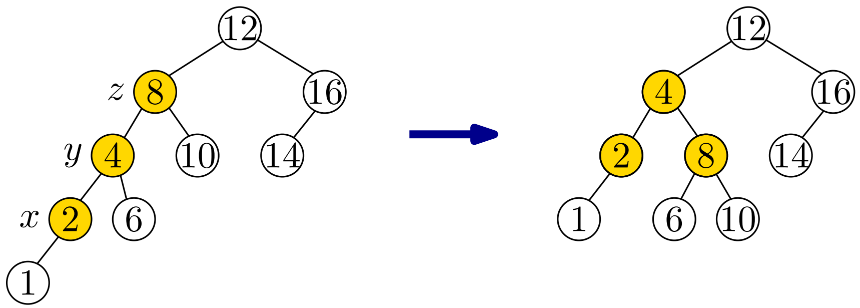


Left rotation

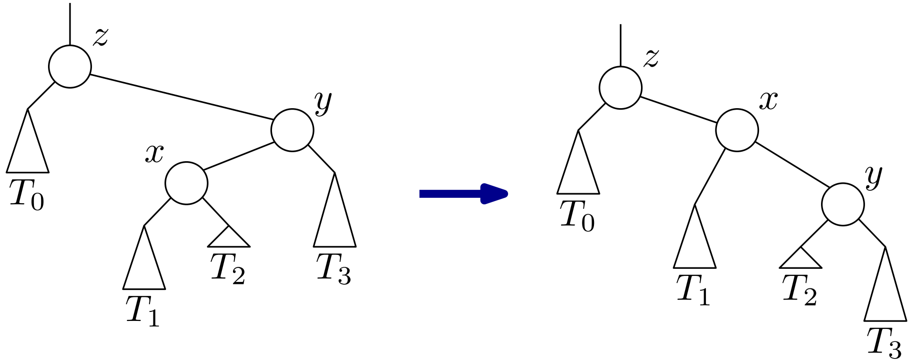
The new subtree at y is balanced since

$$h(T_0) - 1 \leq h(T_3) \leq h(T_0) = h(T_2) \leq h(T_1) \leq h(T_0) + 1$$

Single Rotation

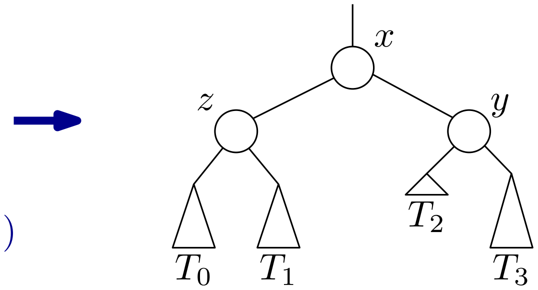


Double rotation



Right rotation around y

Left rotation around z



$h(T_0) = h(T_1) = h(T_3)$
$h(T_1) - 1 \leq h(T_2) \leq h(T_1)$

Double rotation