Balanced Search Trees

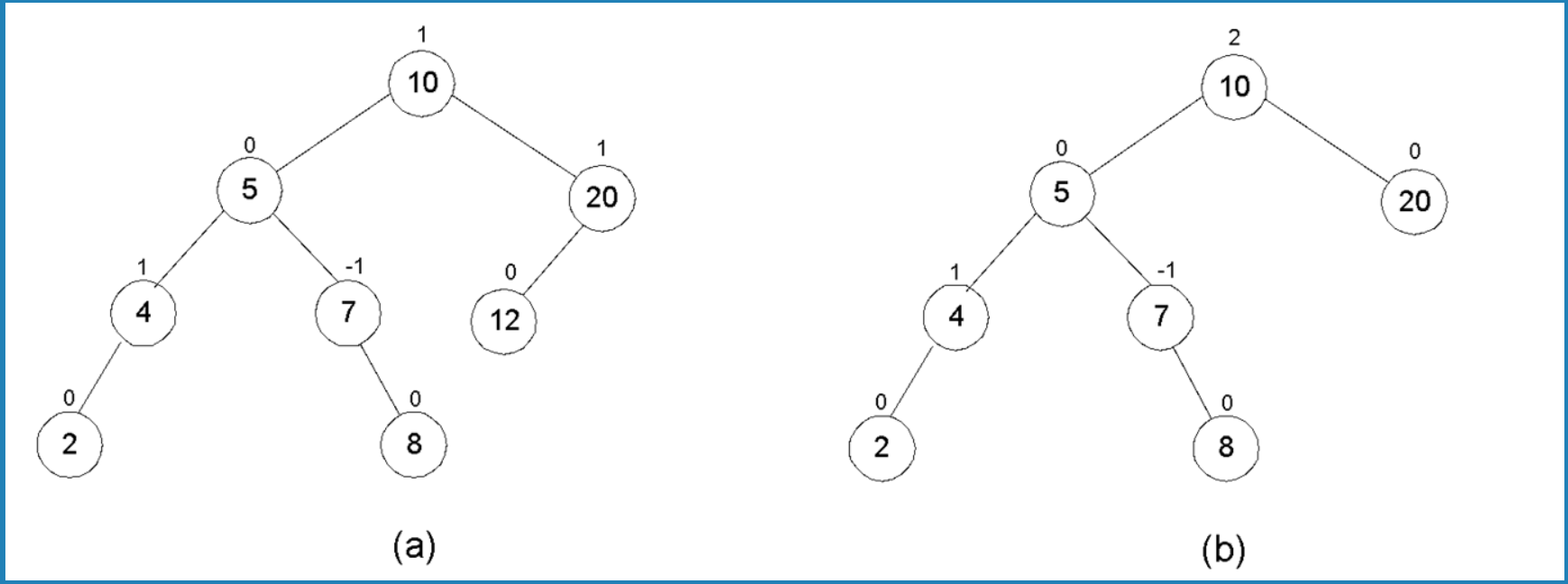
Attractiveness of *binary search tree* is marred by the bad (linear) worst-case efficiency. Two ideas to overcome it are:

- **to rebalance binary search tree when a new insertion makes the tree "too unbalanced"**
  - ***AVL trees***
  - ***red-black trees***

- **to allow more than one key and two children**
  - ***2-3 trees***
  - ***2-3-4 trees***
  - ***B-trees***

Balanced trees: AVL trees

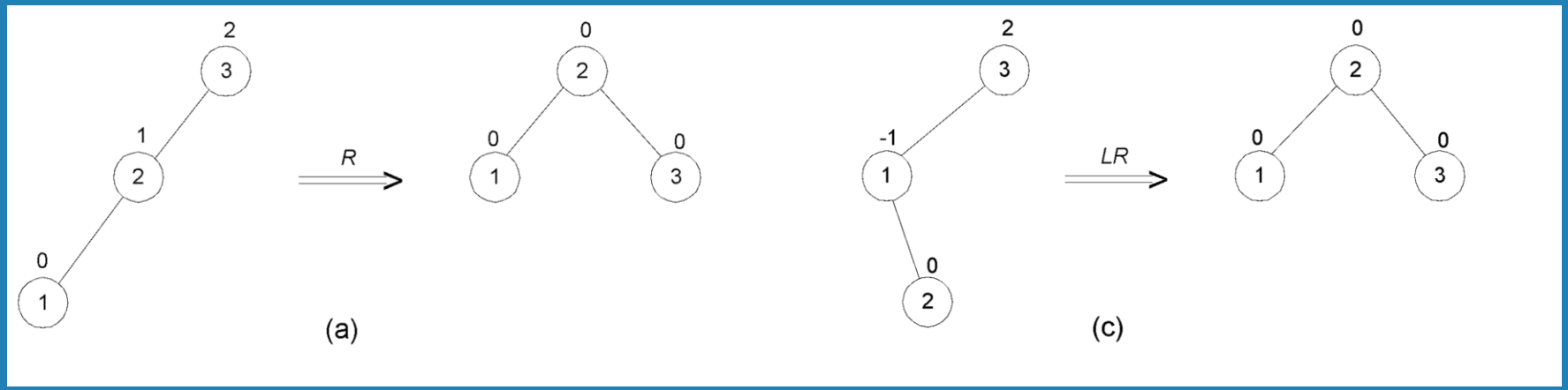
Definition An *AVL tree* is a binary search tree in which, for every node, the difference between the heights of its left and right subtrees, called the *balance factor*, is at most 1 (with the height of an empty tree defined as -1)



Tree (a) is an AVL tree; tree (b) is not an AVL tree

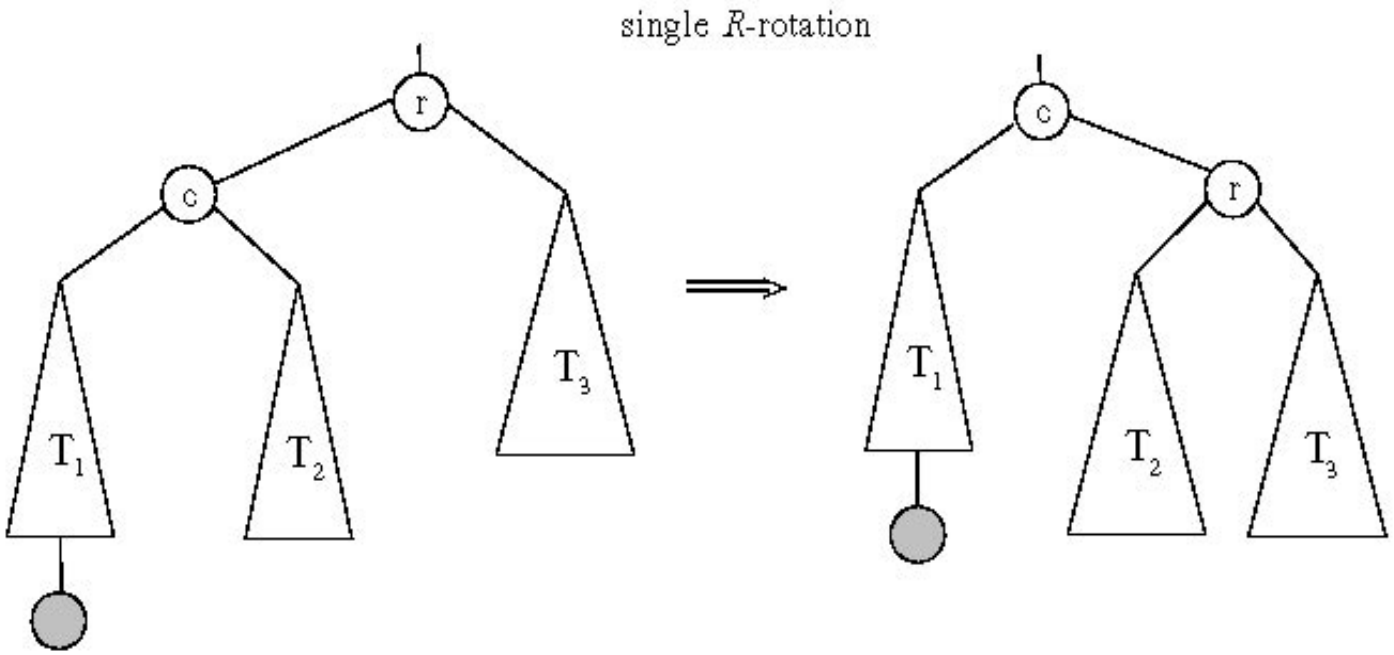
Rotations

If a key insertion violates the balance requirement at some node, the subtree rooted at that node is transformed via one of the four *rotations*. (The rotation is always performed for a subtree rooted at an "unbalanced" node closest to the new leaf.)

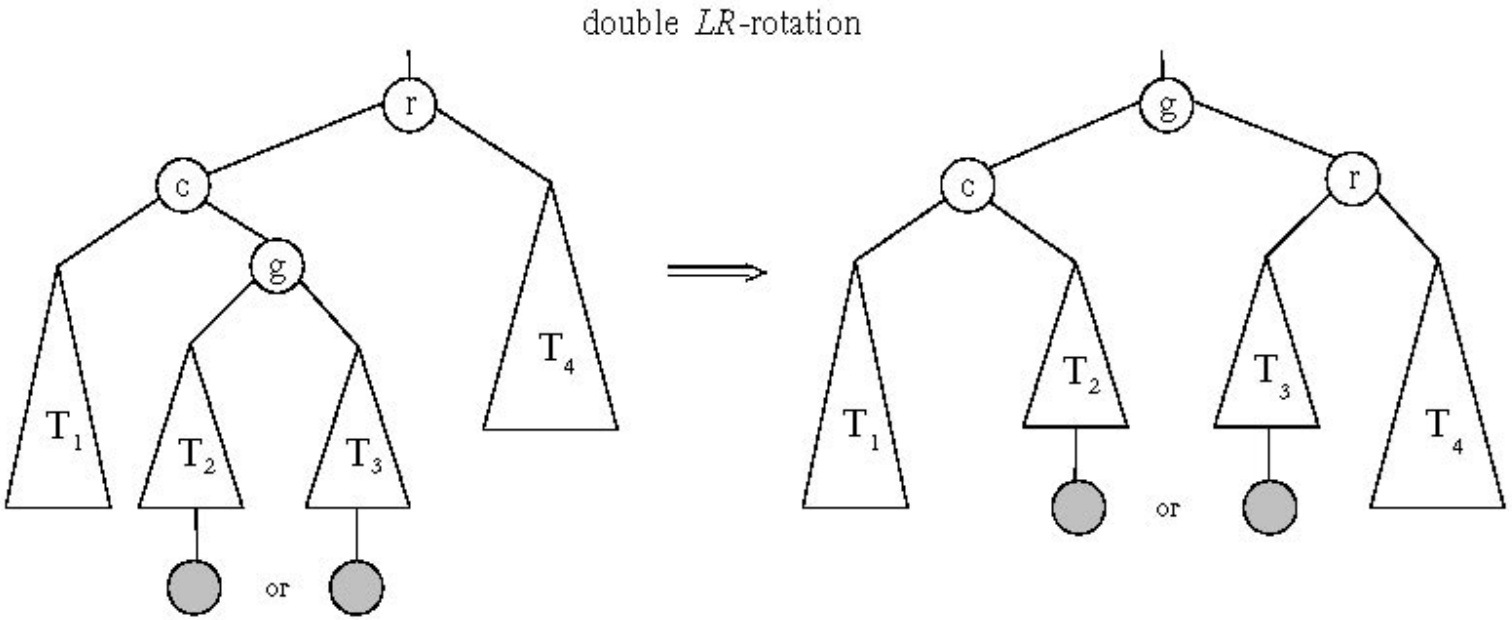


Single *R*-rotation **Double *LR*-rotation**

General case: Single R-rotation



General case: Double LR-rotation

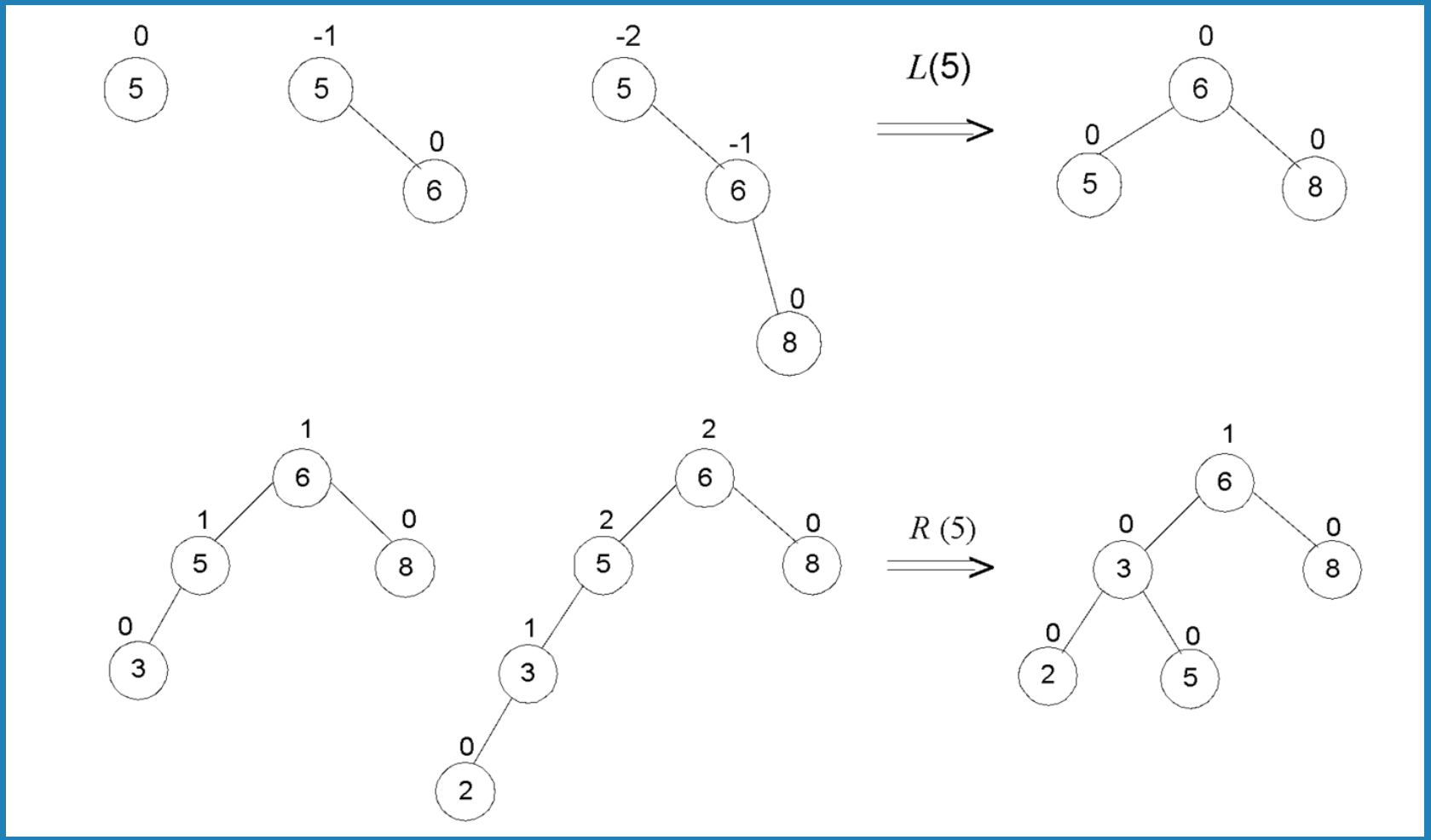


AVL tree construction - an example

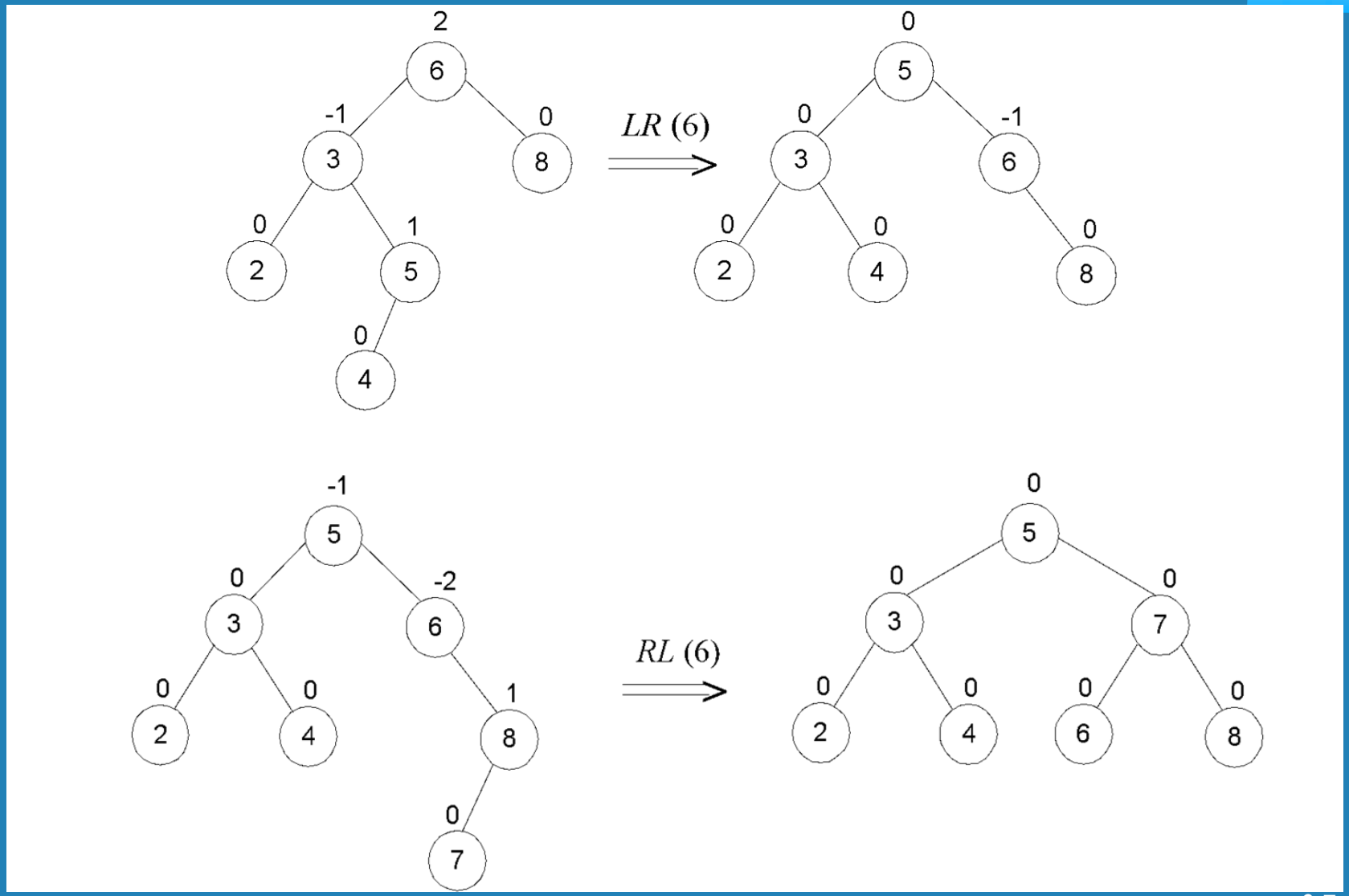
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7

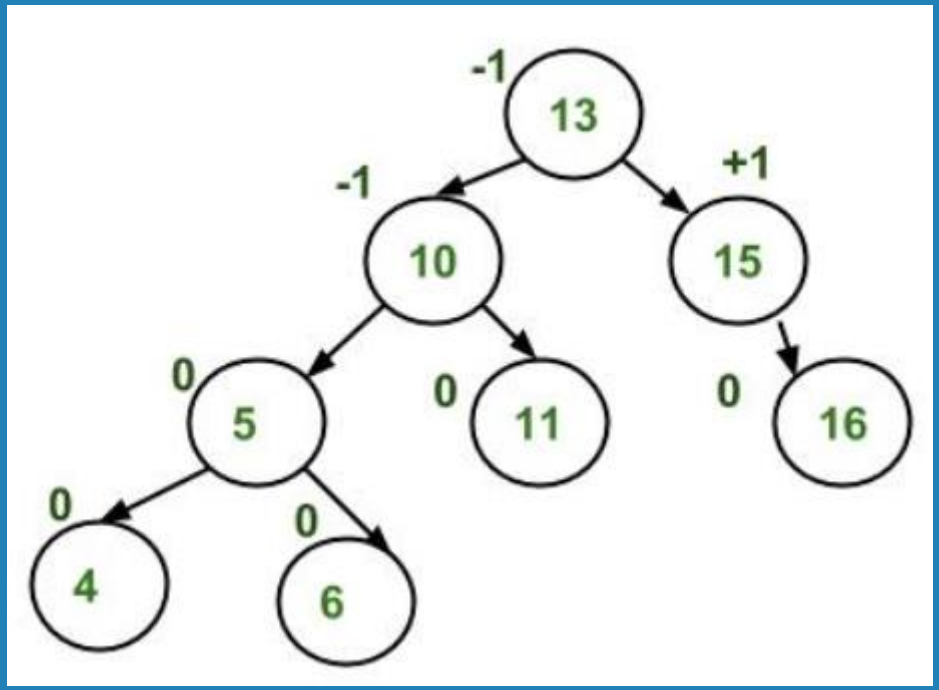
AVL tree construction - an example

Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7



AVL tree construction - an example (cont.)





Analysis of AVL trees

- $h \leq 1.4404 \log_2 (n + 2) - 1.3277$ N(h-1) + N(h-2) ≤ N(h)
  average height: $1.01 \log_2 n + 0.1$ for large n (found empirically)
- Search and insertion are O(log n)
- Deletion is more complicated but is also O(log n)
- Disadvantages:
  - frequent rotations
  - complexity
- A similar idea: *red-black trees* (height of subtrees is allowed to differ by up to a factor of 2)