Transform and Conquer

Balanced Search Trees

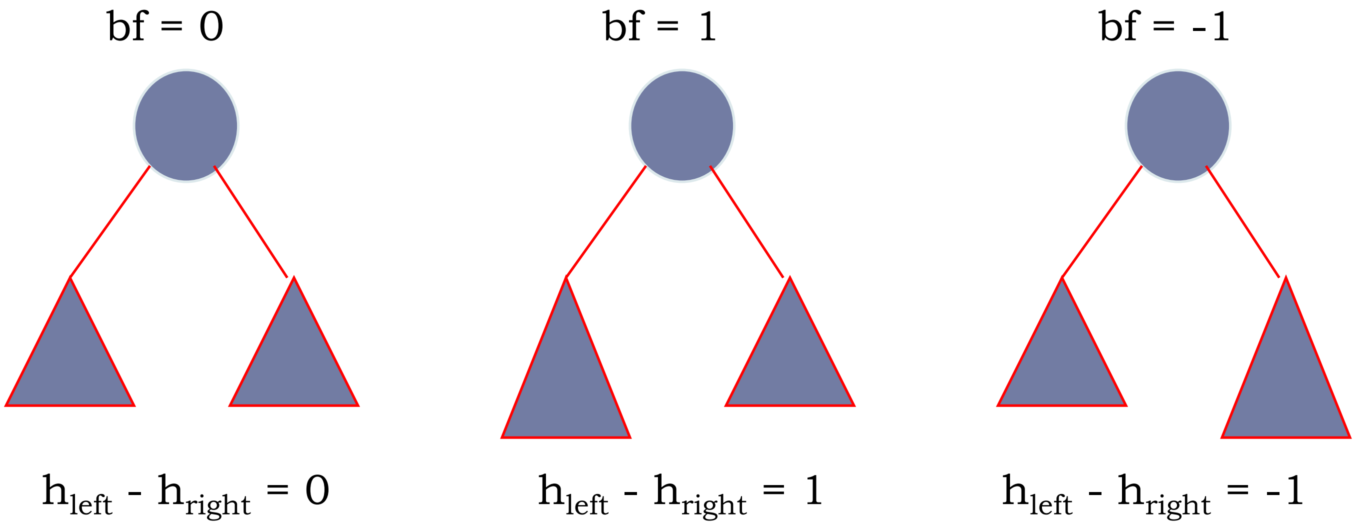
Attractiveness of binary search tree is marred by the bad (linear) worst-case efficiency. Two ideas to overcome it are:

- to rebalance binary search tree when a new insertion makes the tree "too unbalanced" (instance simplification)
  - AVL trees
  - red-black trees
- to allow more than one key per node in a search tree (representation change)
  - 2-3 trees
  - 2-3-4 trees
  - B-trees

Balanced trees: AVL trees

- An AVL tree is a binary search tree in which, for every node, the balance factor is -1, 0 or 1;
- the balance factor is the difference between the heights of its left and right sub-trees
- the height of an empty tree is defined as 0

AVL Tree Invariants



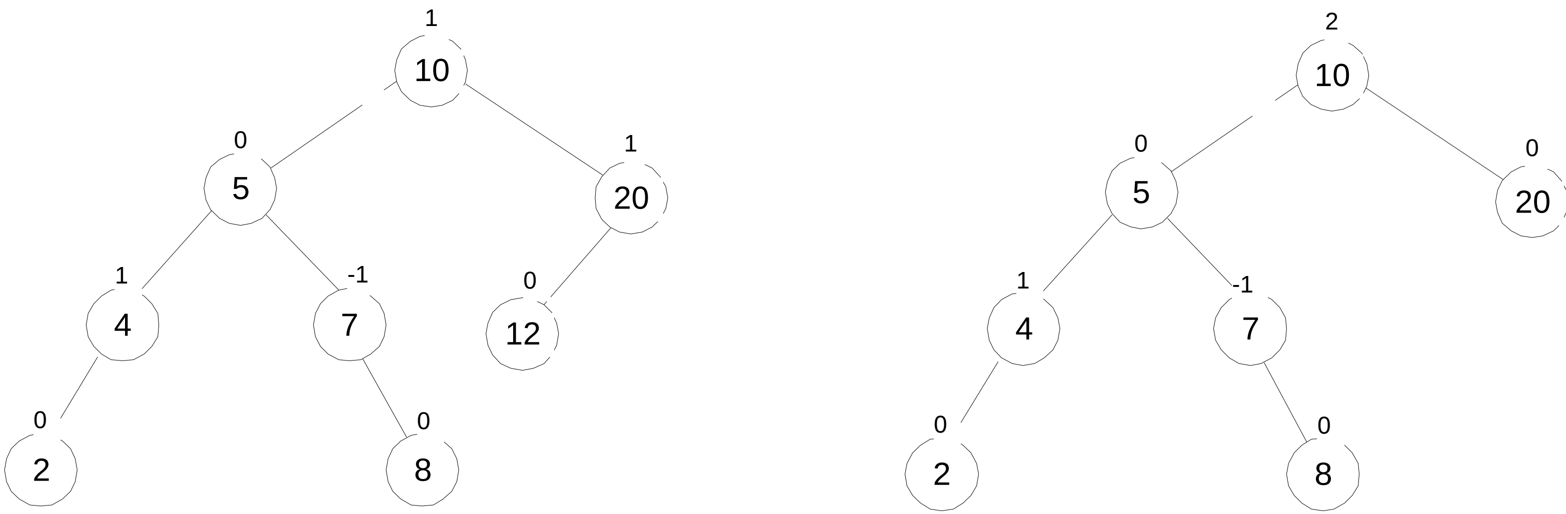
Node Invariant 1: All keys in left sub-tree are less than key in current node. All keys in right sub-tree are greater than key in current node.

Node Invariant 2: The balance factor is 1, 0 or -1.

If we assume invariants are true as we enter any operation, the design problem reduces to making sure they are true when we leave any operation.

Balanced trees: AVL trees

An AVL tree is a binary search tree in which, for every node, the difference between the heights of its left and right sub-trees, called the balance factor, is at most 1 (with the height of an empty tree defined as 0)

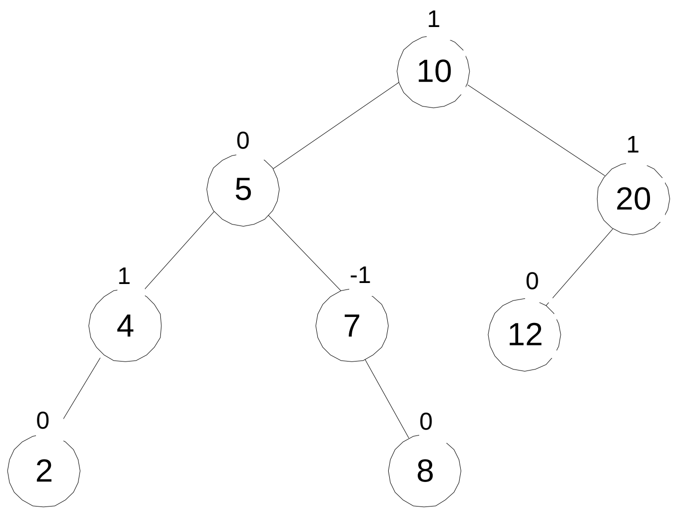


(a)

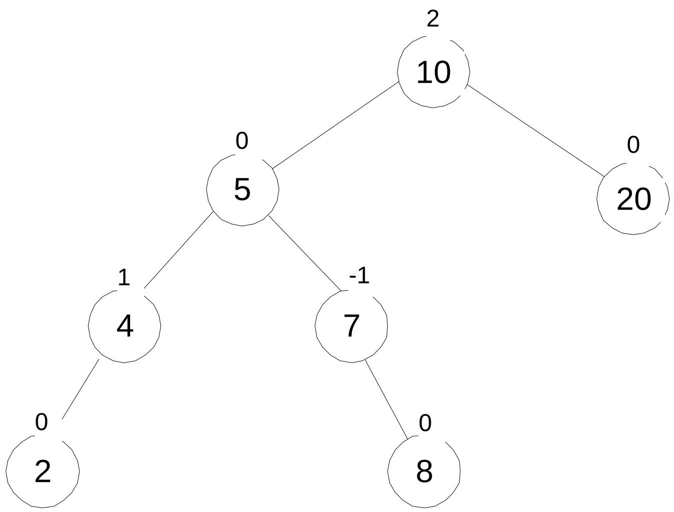
(b)

Balanced trees: AVL trees

An *AVL tree* is a binary search tree in which, for every node, the difference between the heights of its left and right sub-trees, called the *balance factor*, is at most 1 (with the height of an empty tree defined as 0)



(a)



(b)

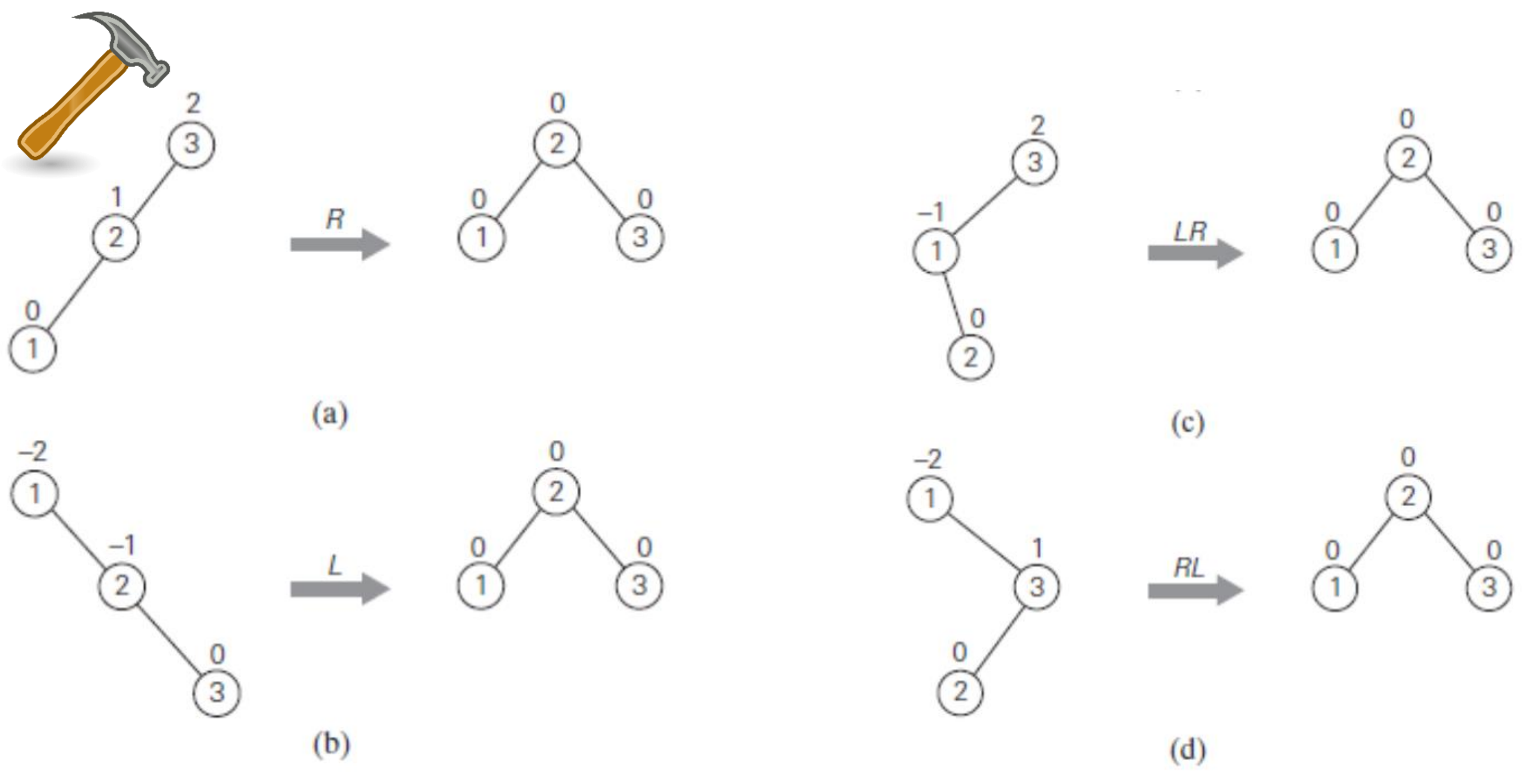
(a) is an AVL tree

(b) is not an AVL tree

Rotations

If a key insertion violates the balance requirement at some node, the sub-tree rooted at that node is transformed via one of the four rotations.

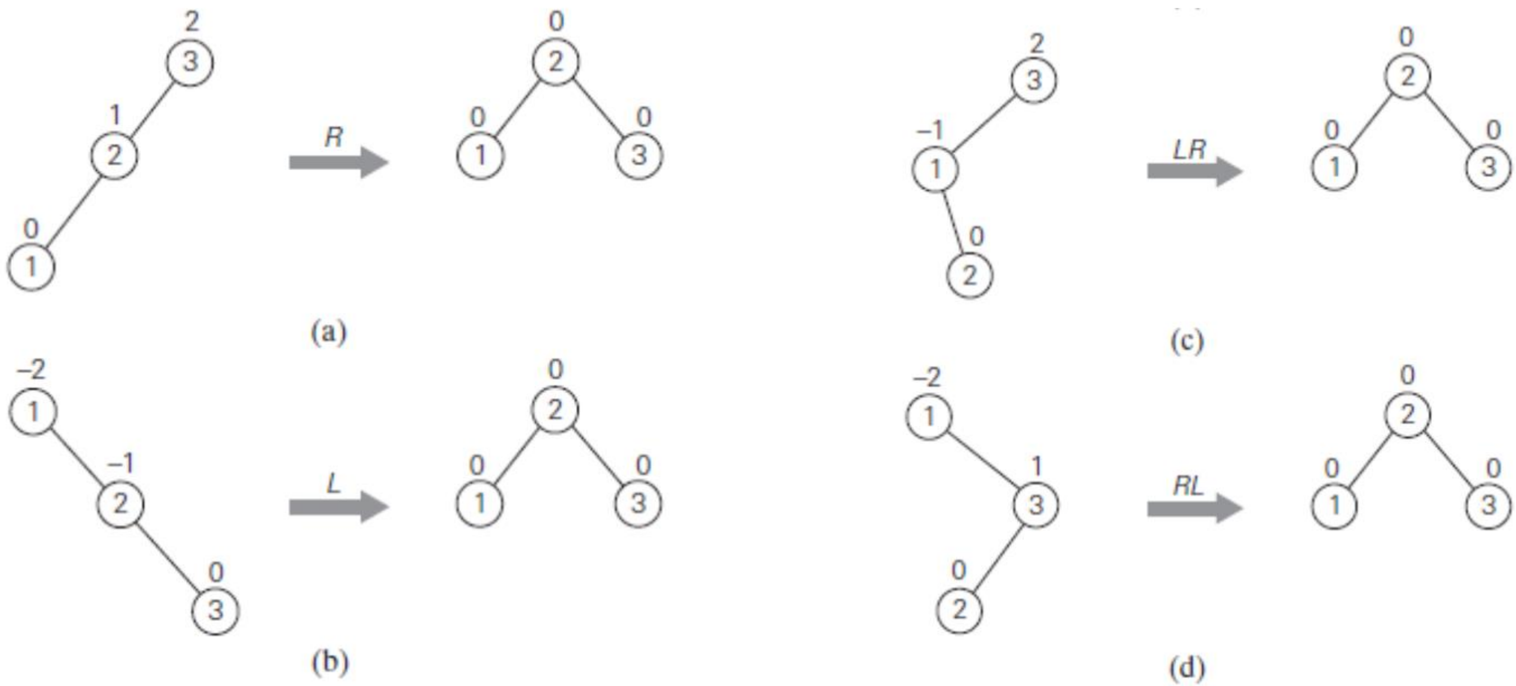
The rotation is always performed for a sub-tree rooted at an "unbalanced" node closest to the new leaf.



Rotations

If a key insertion violates the balance requirement at some node, the sub-tree rooted at that node is transformed via one of the four rotations.

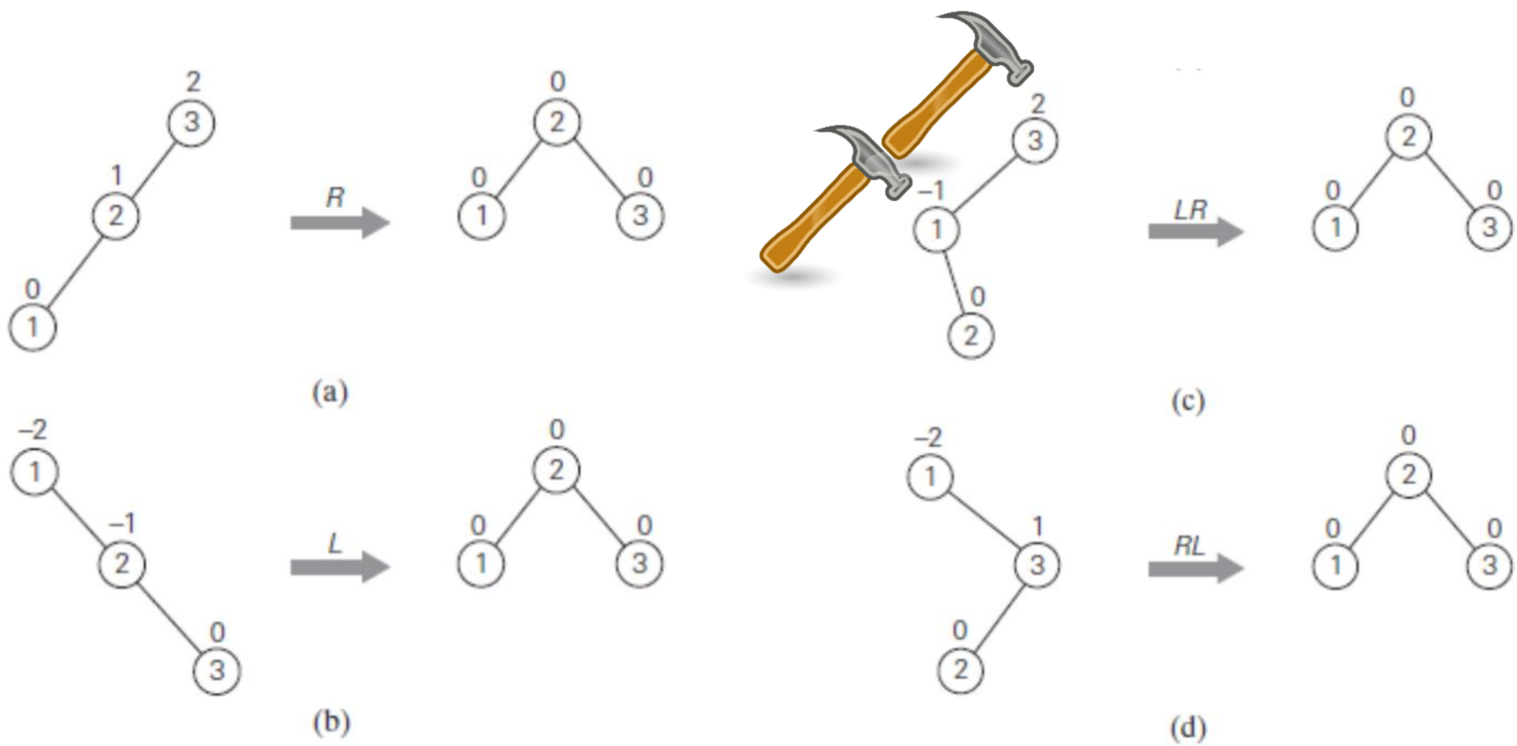
The rotation is always performed for a sub-tree rooted at an “unbalanced” node closest to the new leaf.



Rotations

If a key insertion violates the balance requirement at some node, the sub-tree rooted at that node is transformed via one of the four rotations.

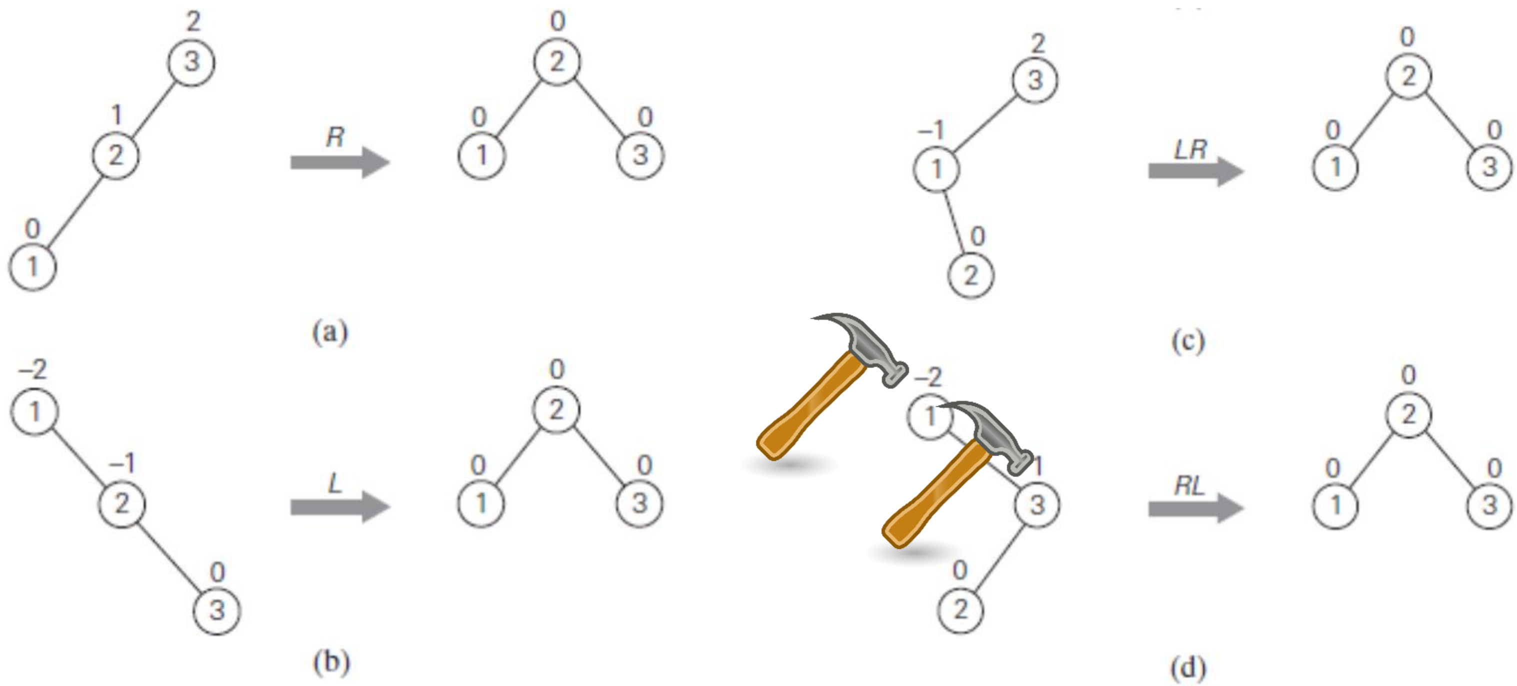
The rotation is always performed for a sub-tree rooted at an “unbalanced” node closest to the new leaf.



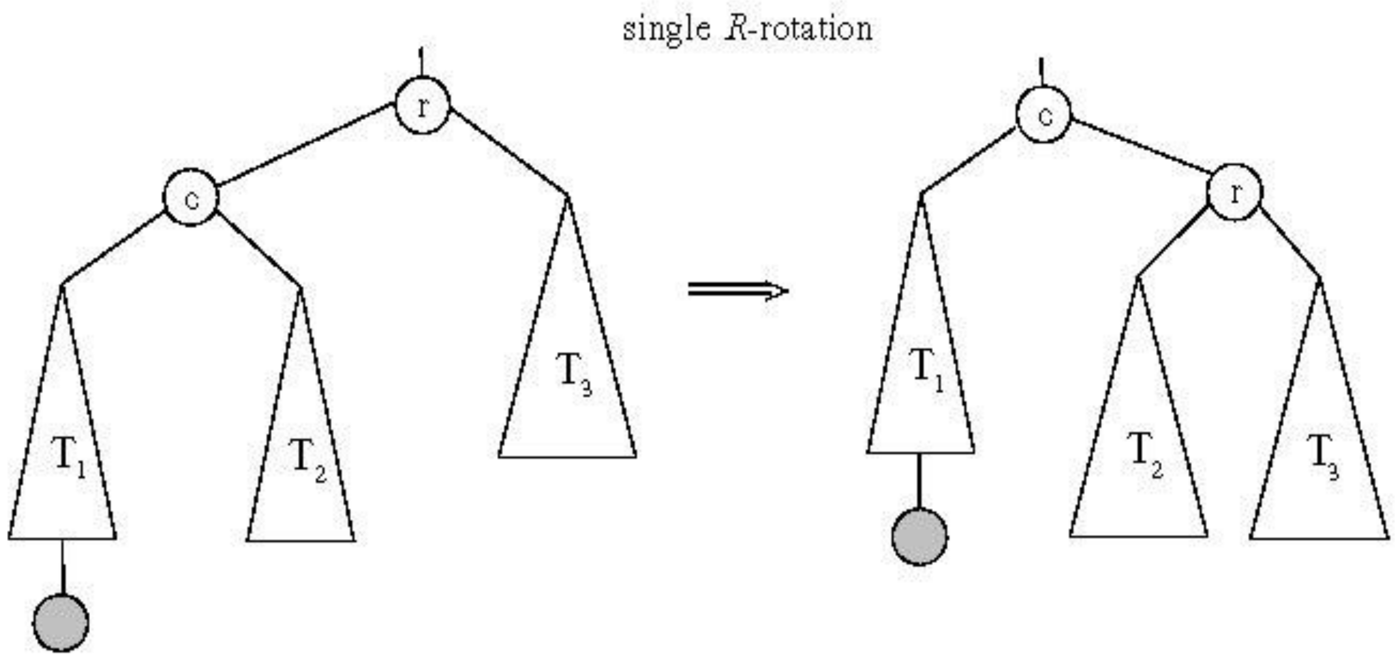
Rotations

If a key insertion violates the balance requirement at some node, the sub-tree rooted at that node is transformed via one of the four rotations.

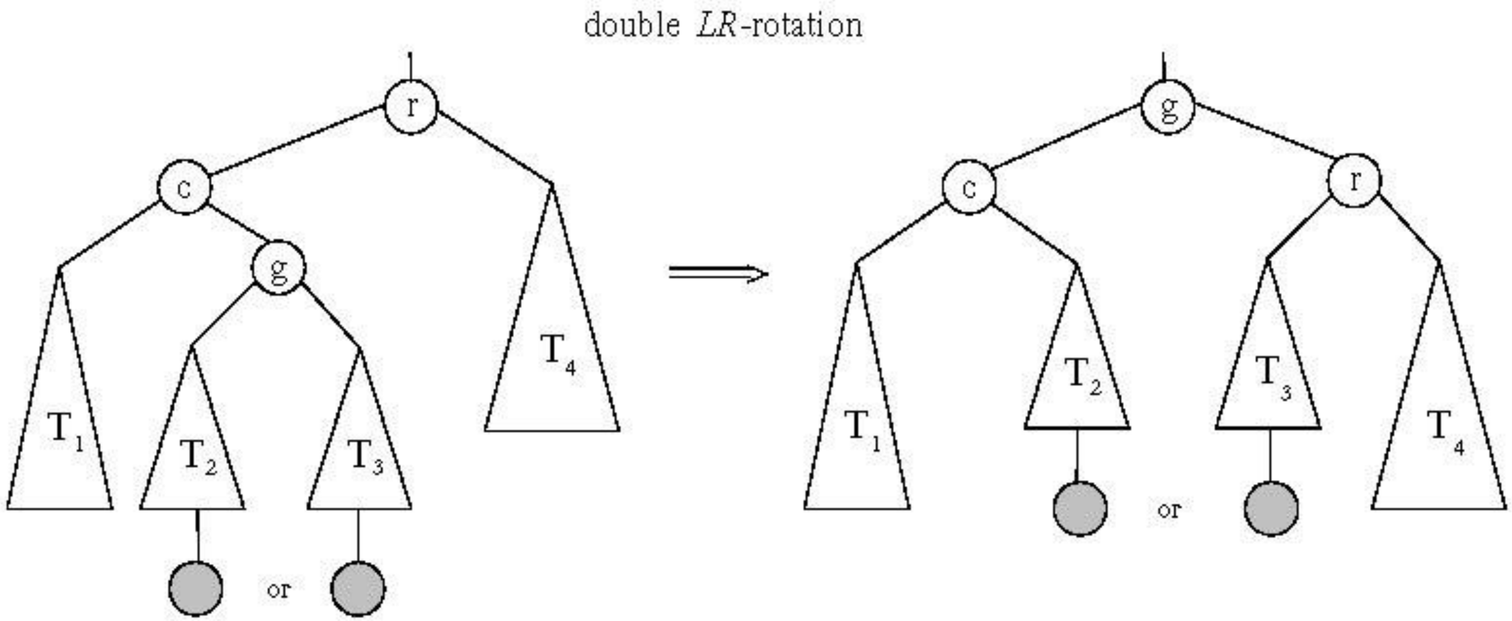
The rotation is always performed for a sub-tree rooted at an "unbalanced" node closest to the new leaf.



General case: Single R-rotation

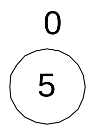


General case: Double LR-rotation



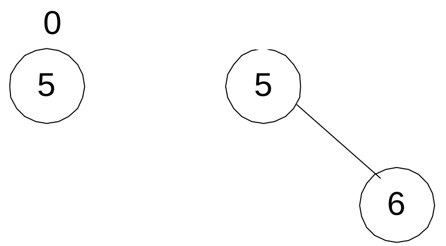
AVL tree construction - an example

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



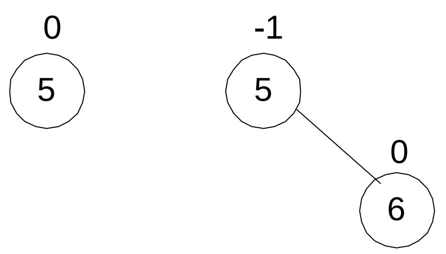
AVL tree construction - an example

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



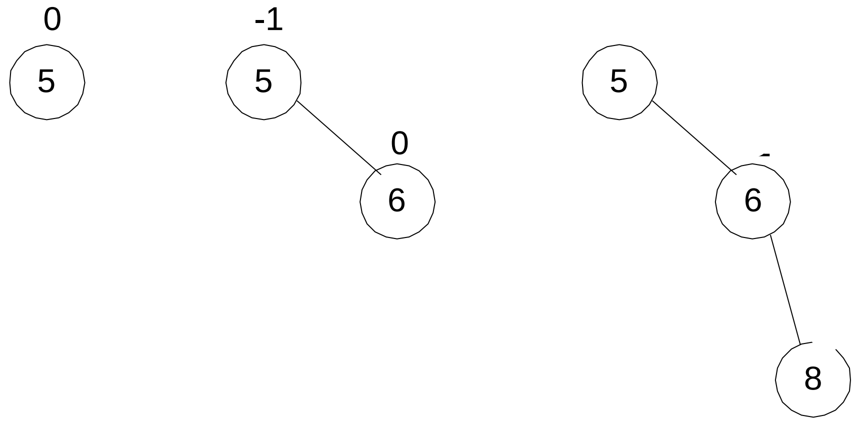
AVL tree construction - an example

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



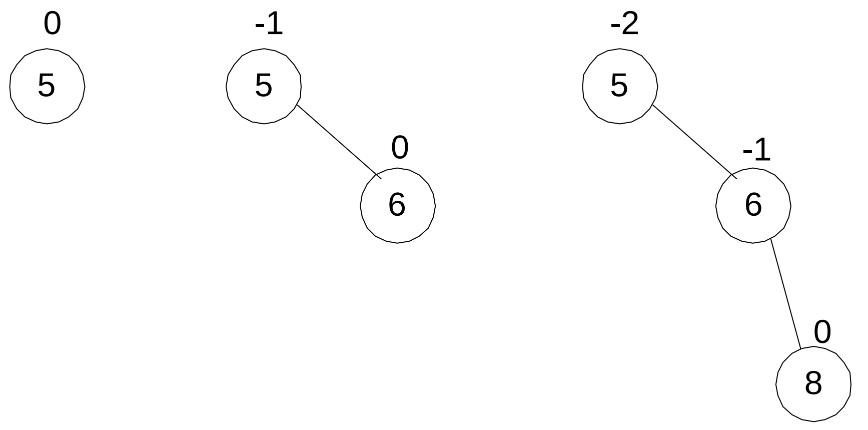
AVL tree construction - an example

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



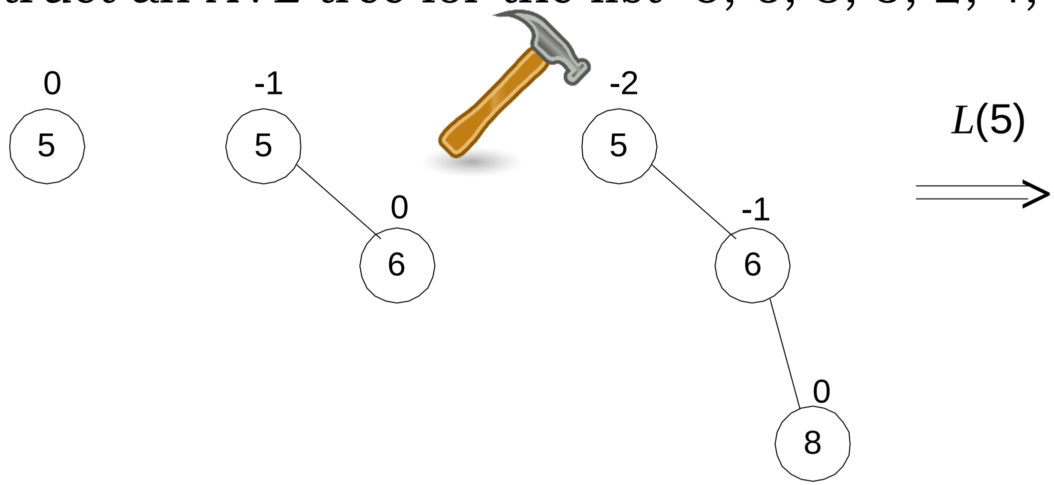
AVL tree construction - an example

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



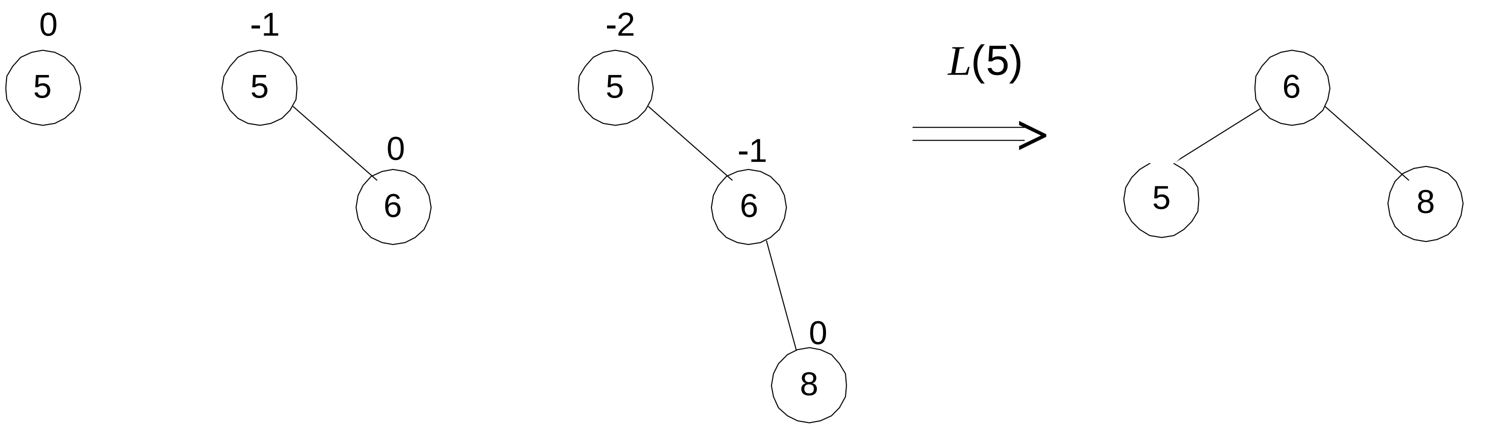
AVL tree construction - an example

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



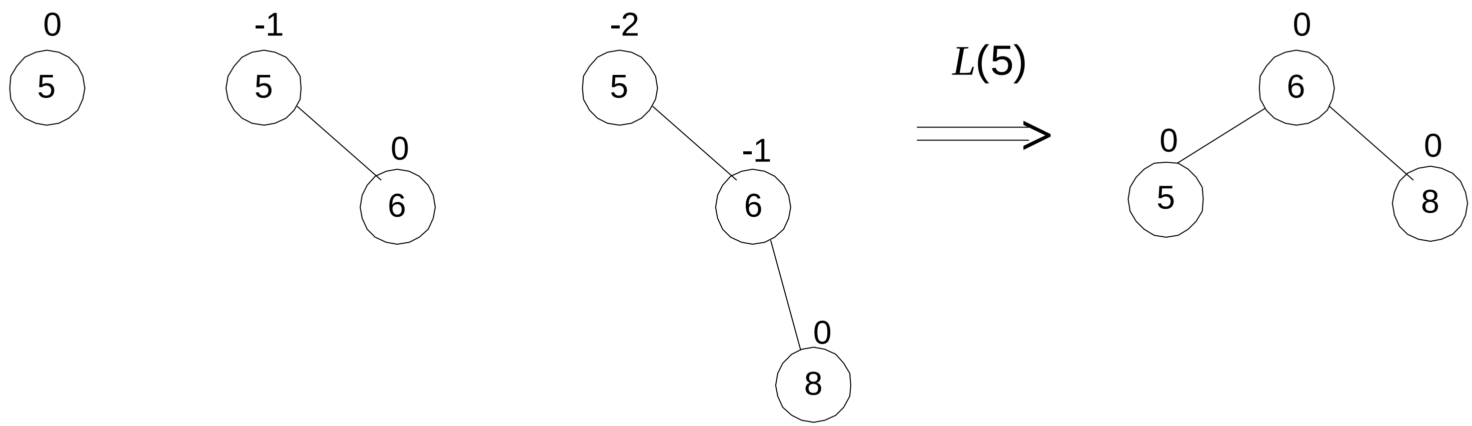
AVL tree construction - an example

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



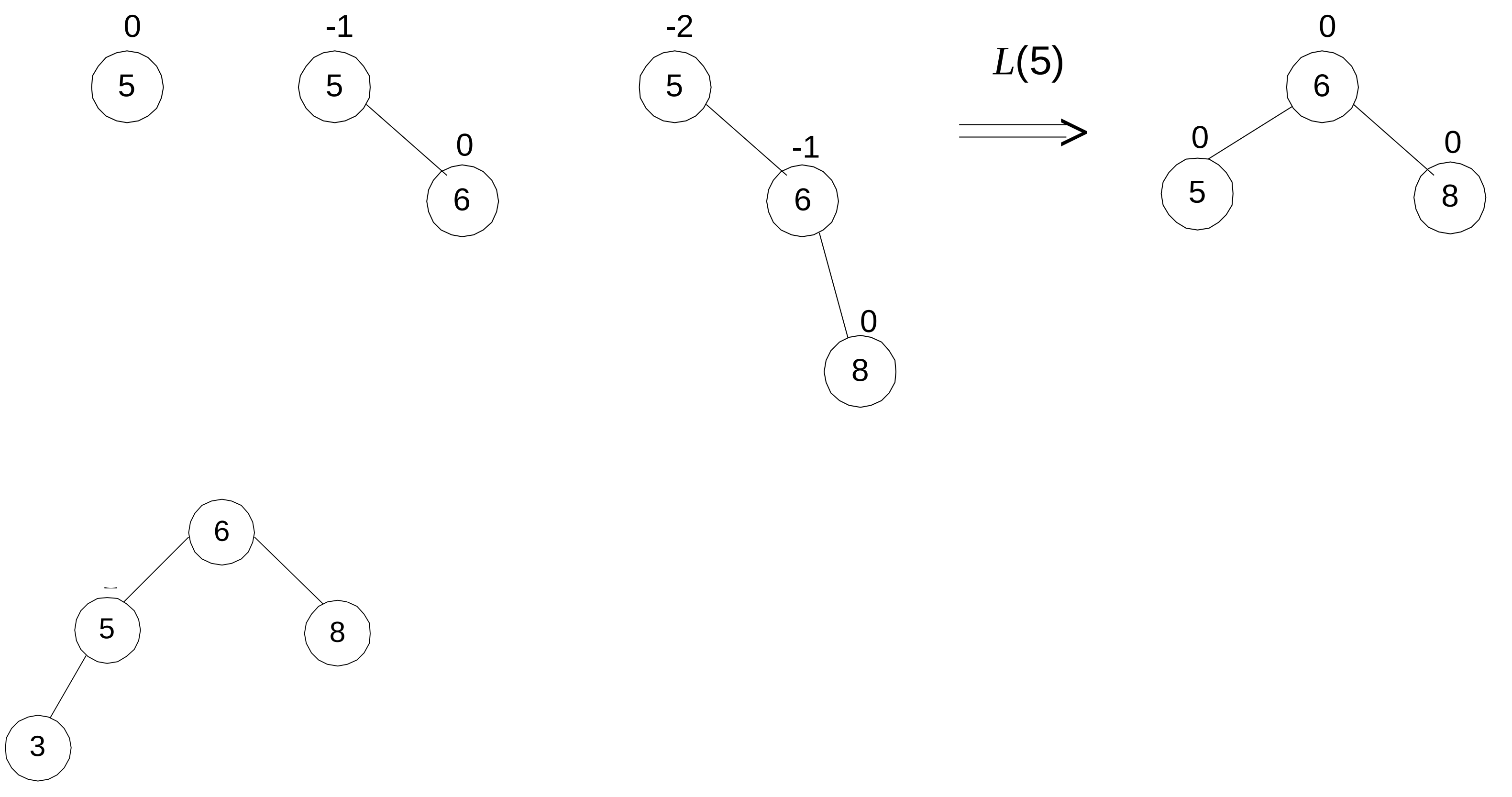
AVL tree construction - an example

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



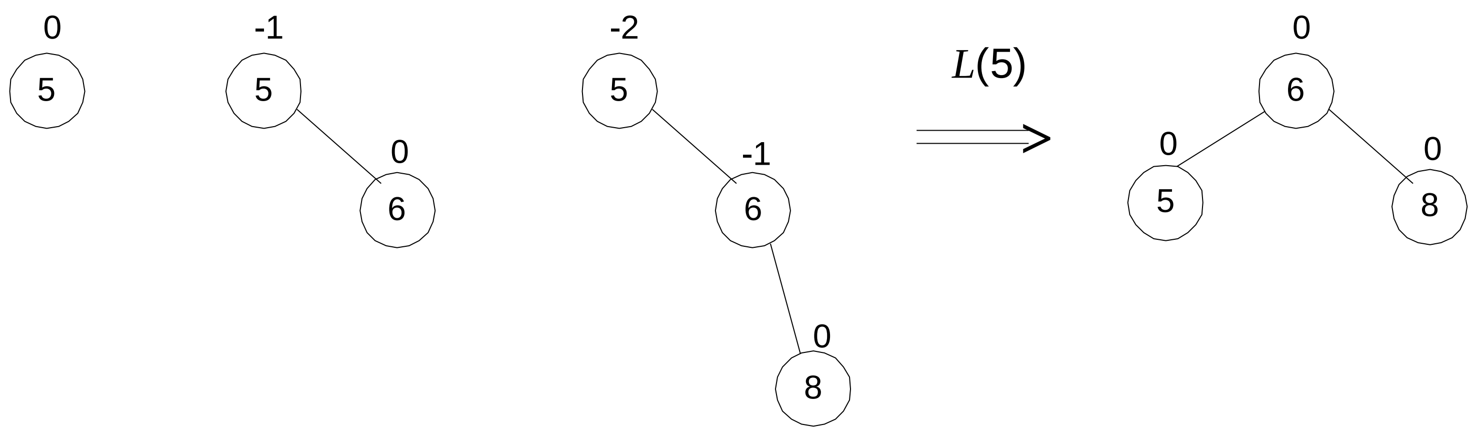
AVL tree construction - an example

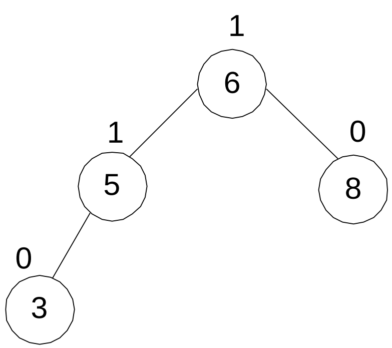
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7



AVL tree construction - an example

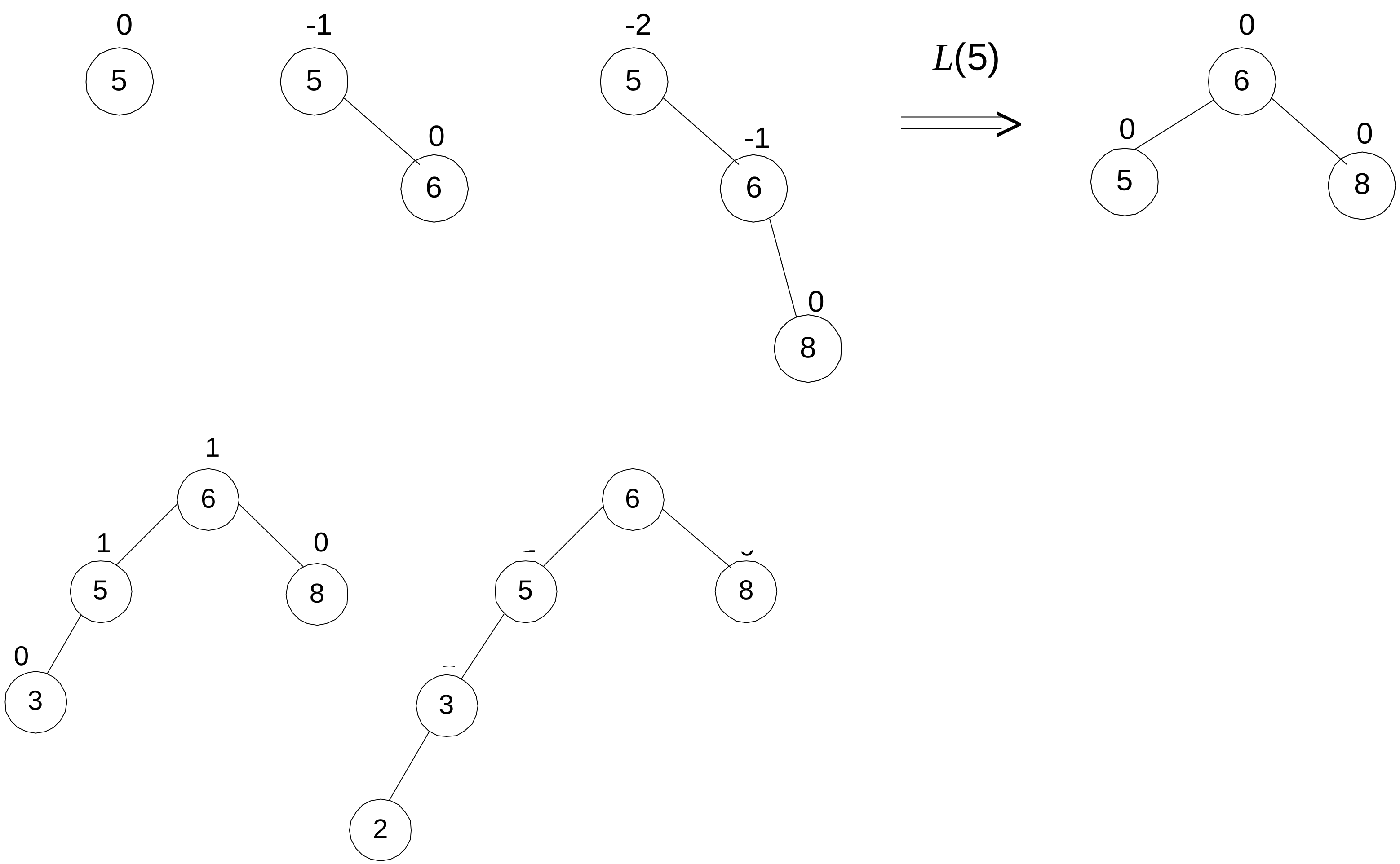
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7





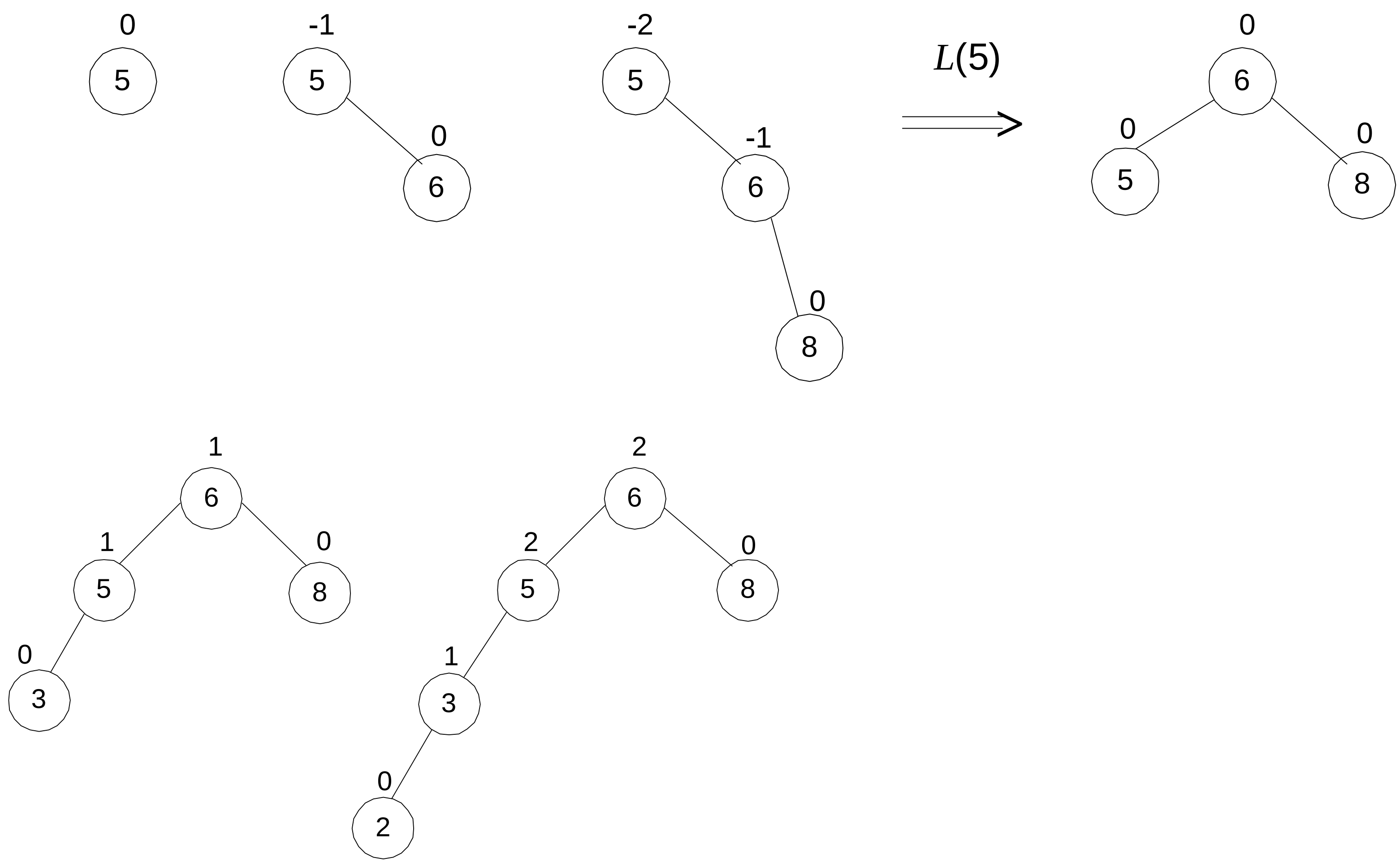
AVL tree construction - an example

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



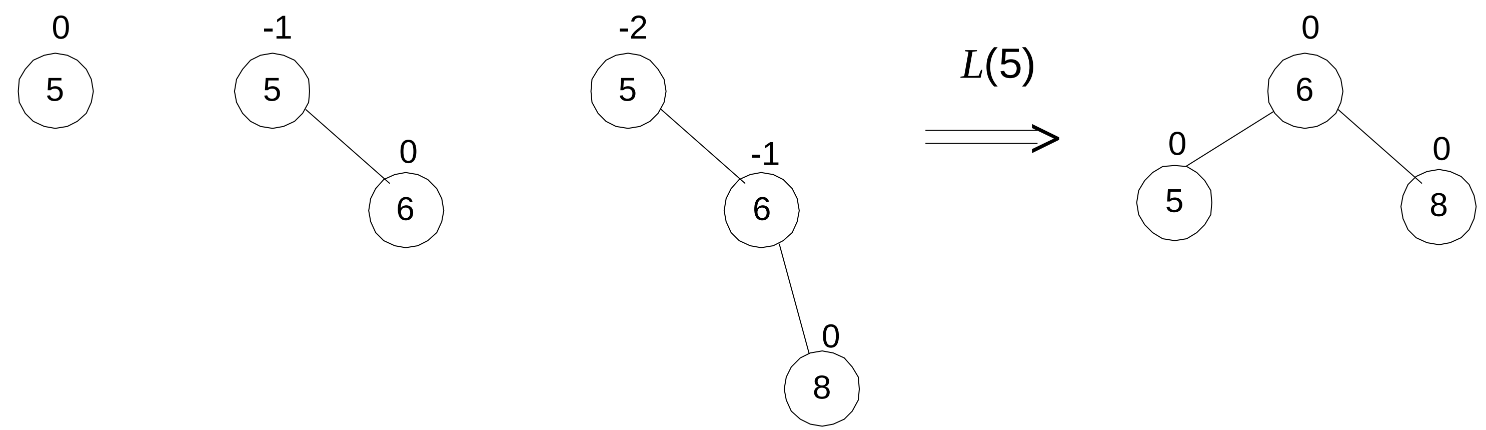
AVL tree construction - an example

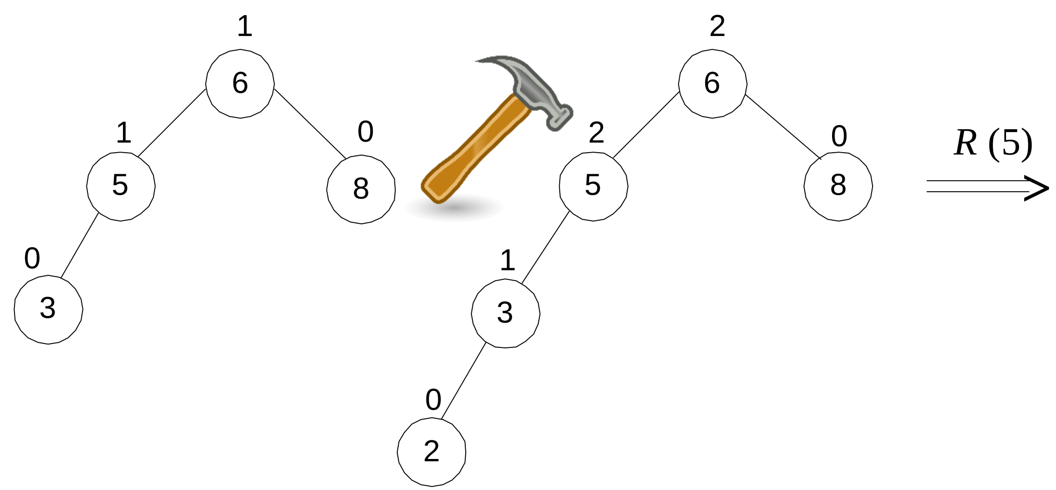
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7



AVL tree construction - an example

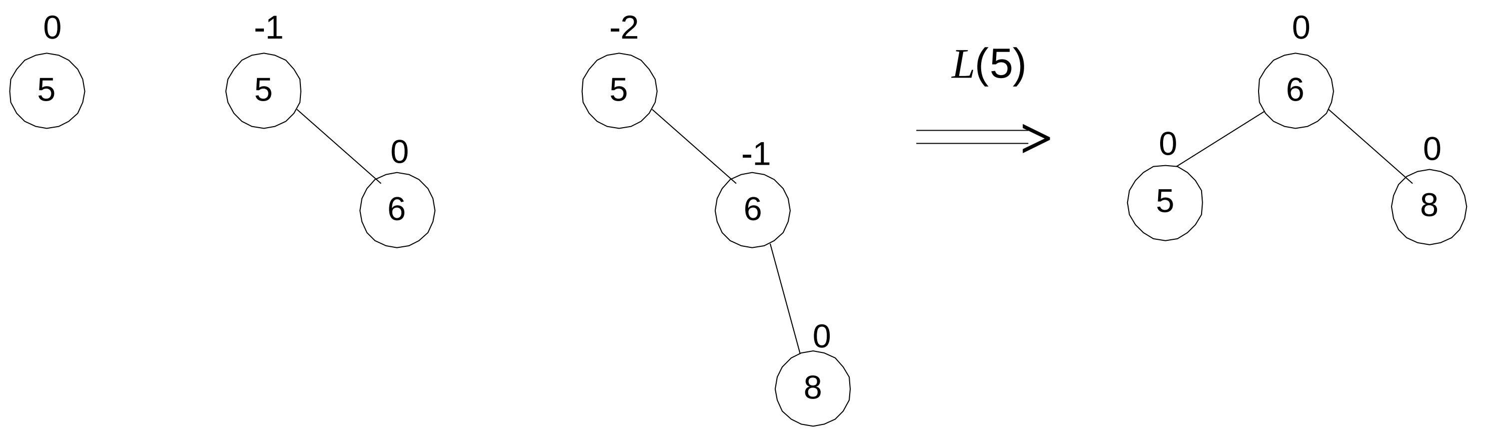
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7

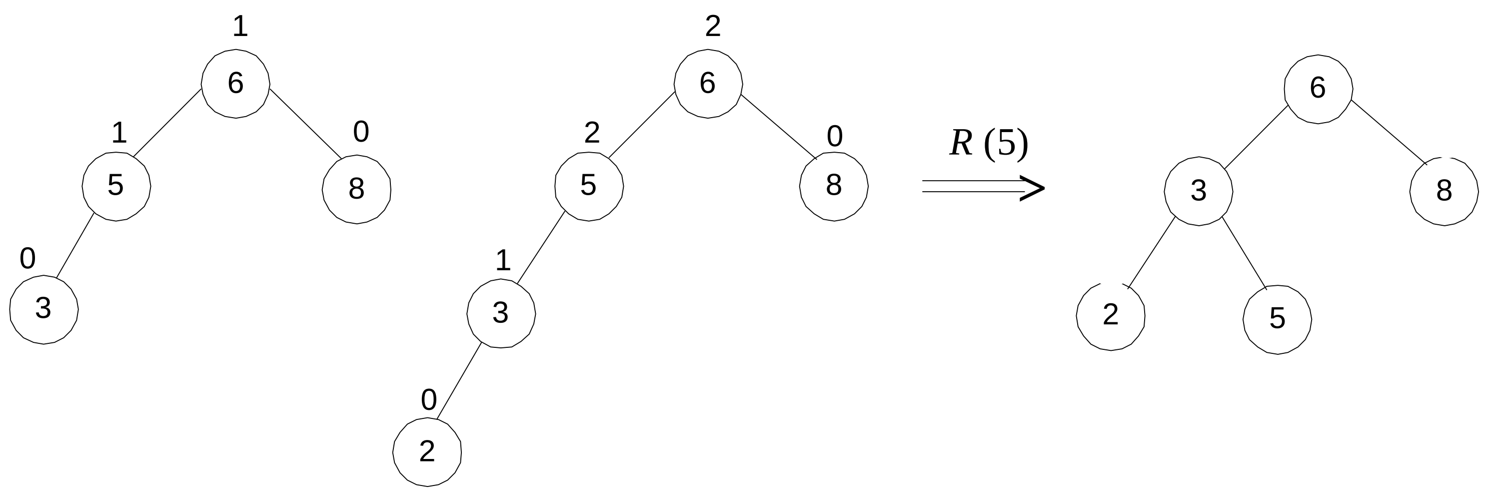




AVL tree construction - an example

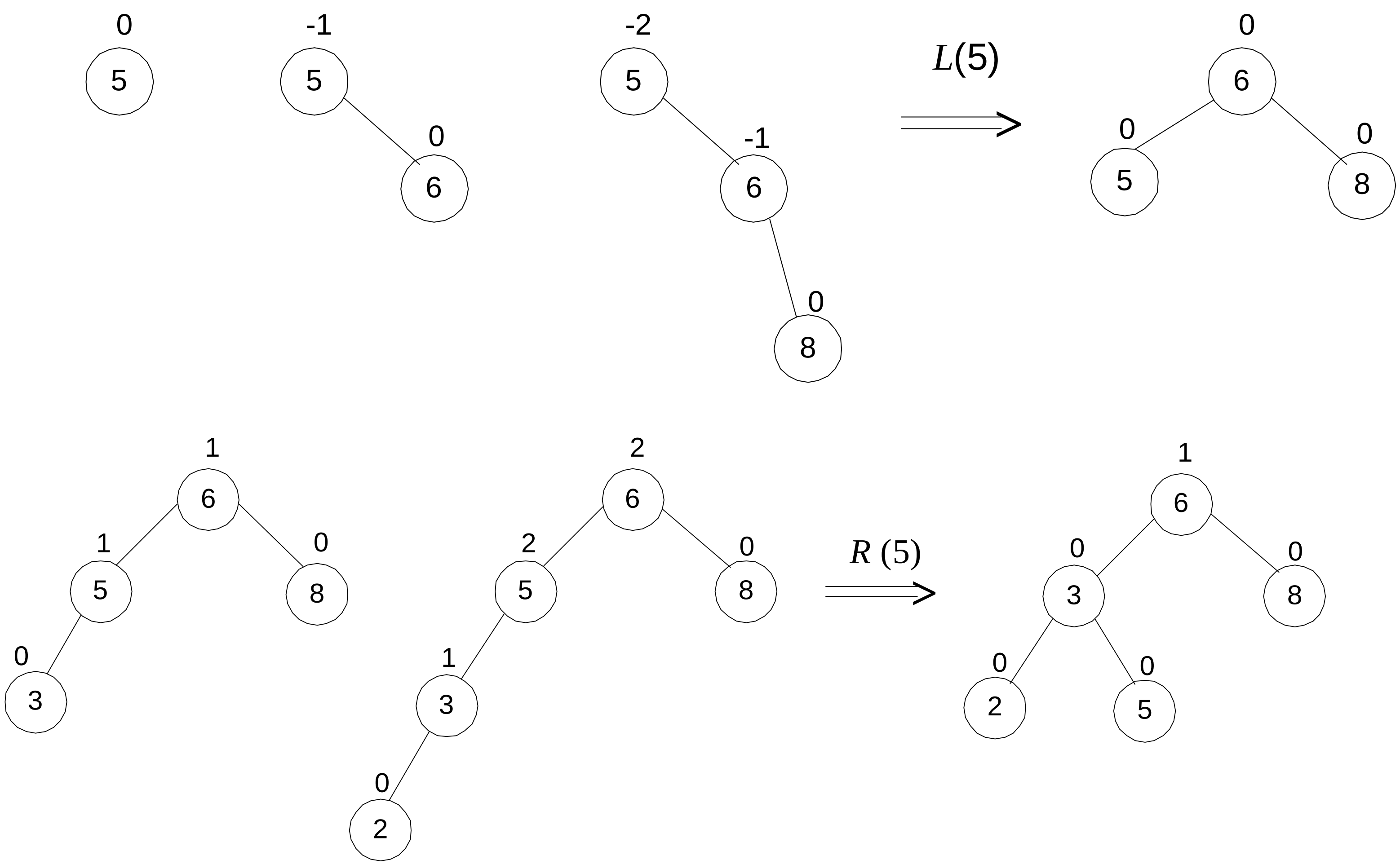
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7



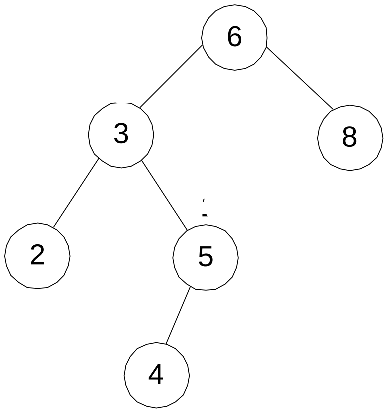


AVL tree construction - an example

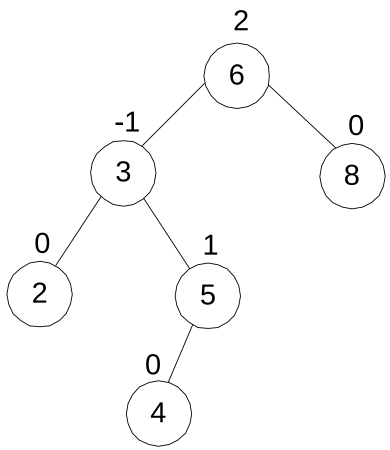
Construct an AVL tree for the list 5, 6, 8, 3, 2, 4, 7



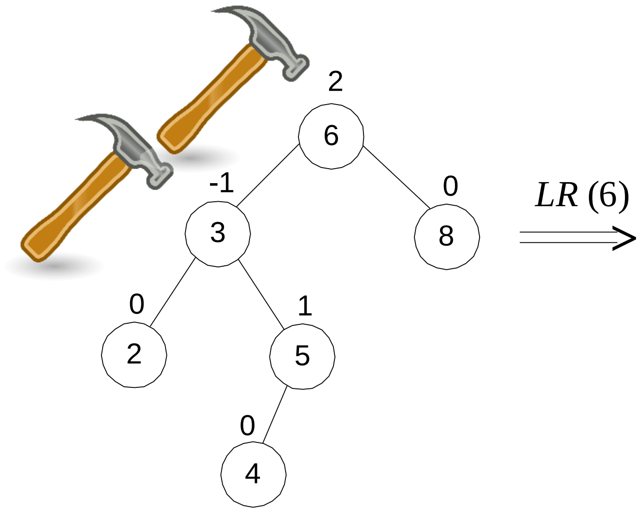
AVL tree construction - an example (cont.)



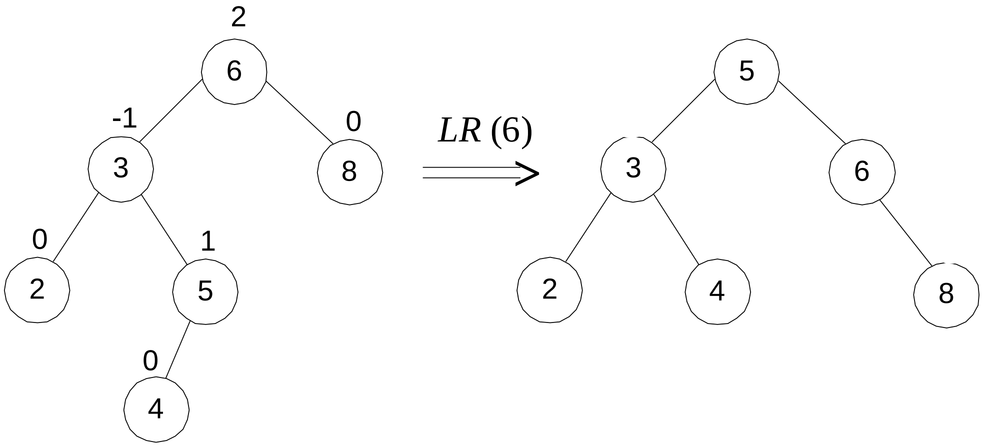
AVL tree construction - an example (cont.)



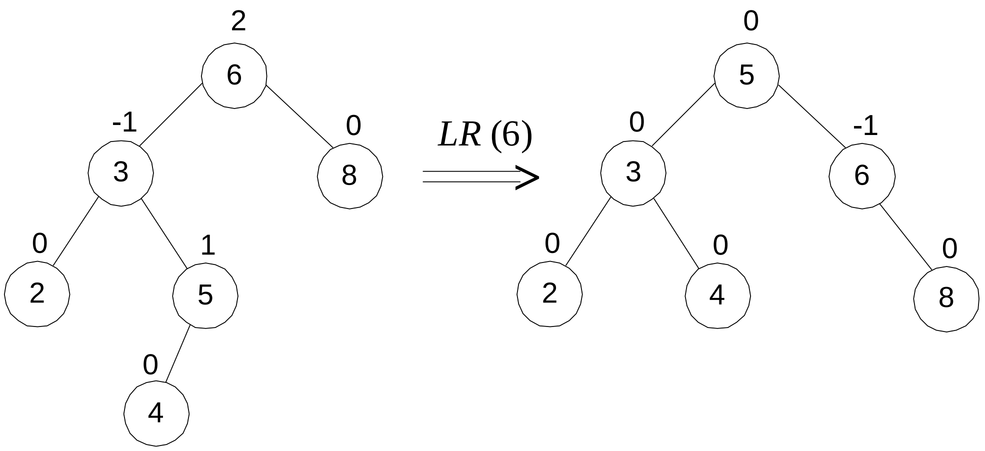
AVL tree construction - an example (cont.)



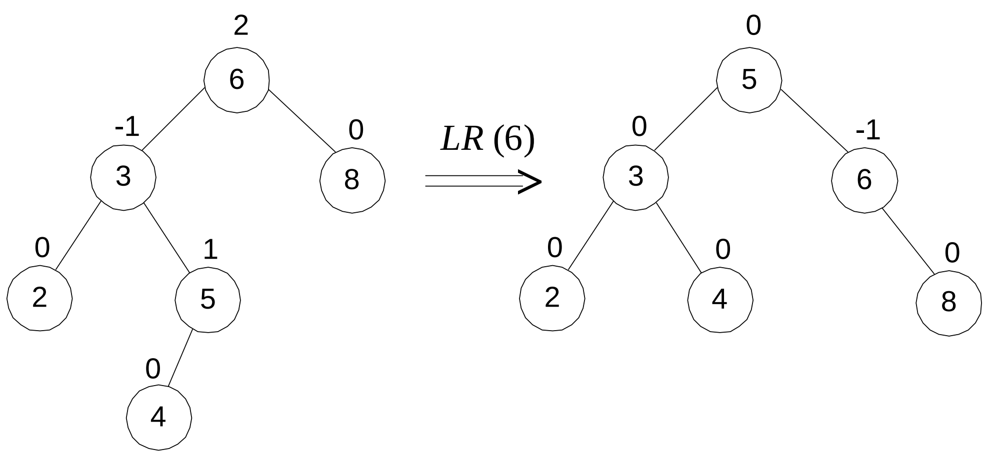
AVL tree construction - an example (cont.)

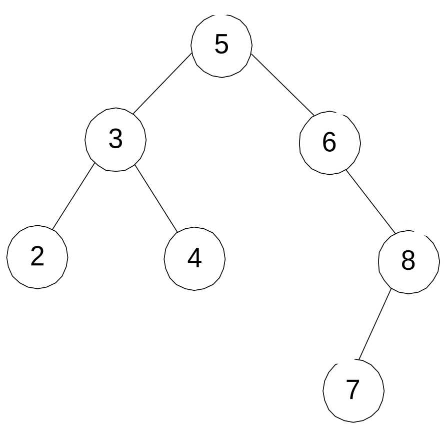


AVL tree construction - an example (cont.)

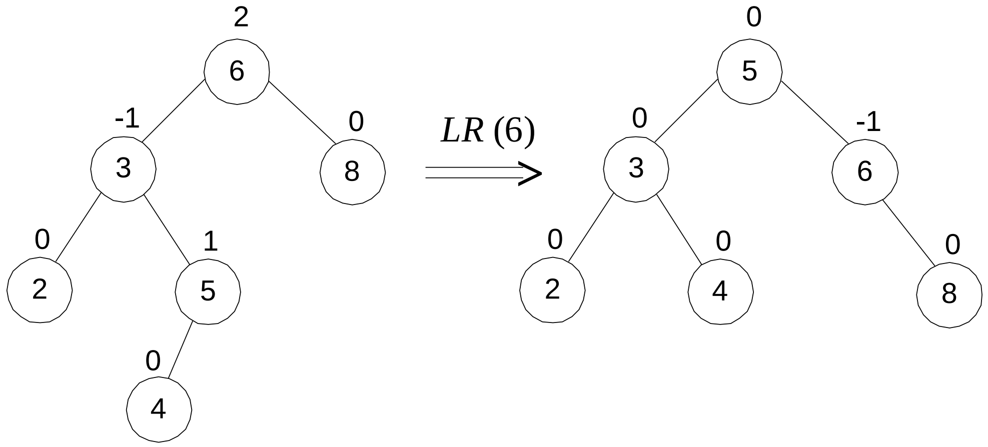


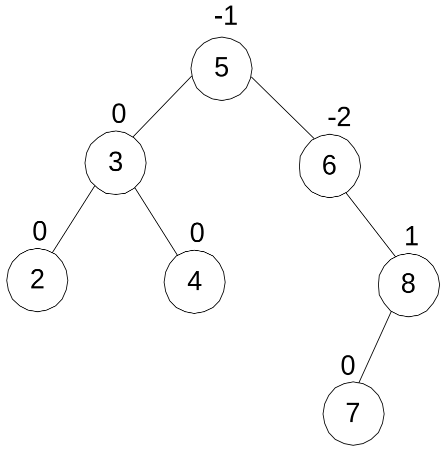
AVL tree construction - an example (cont.)



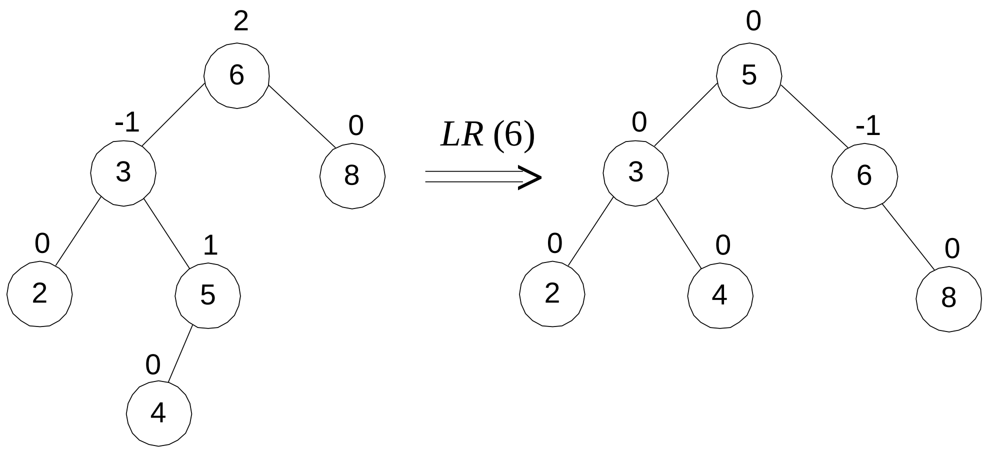


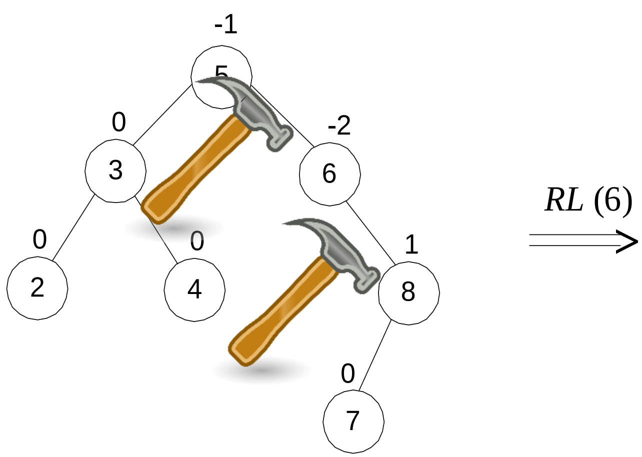
AVL tree construction - an example (cont.)



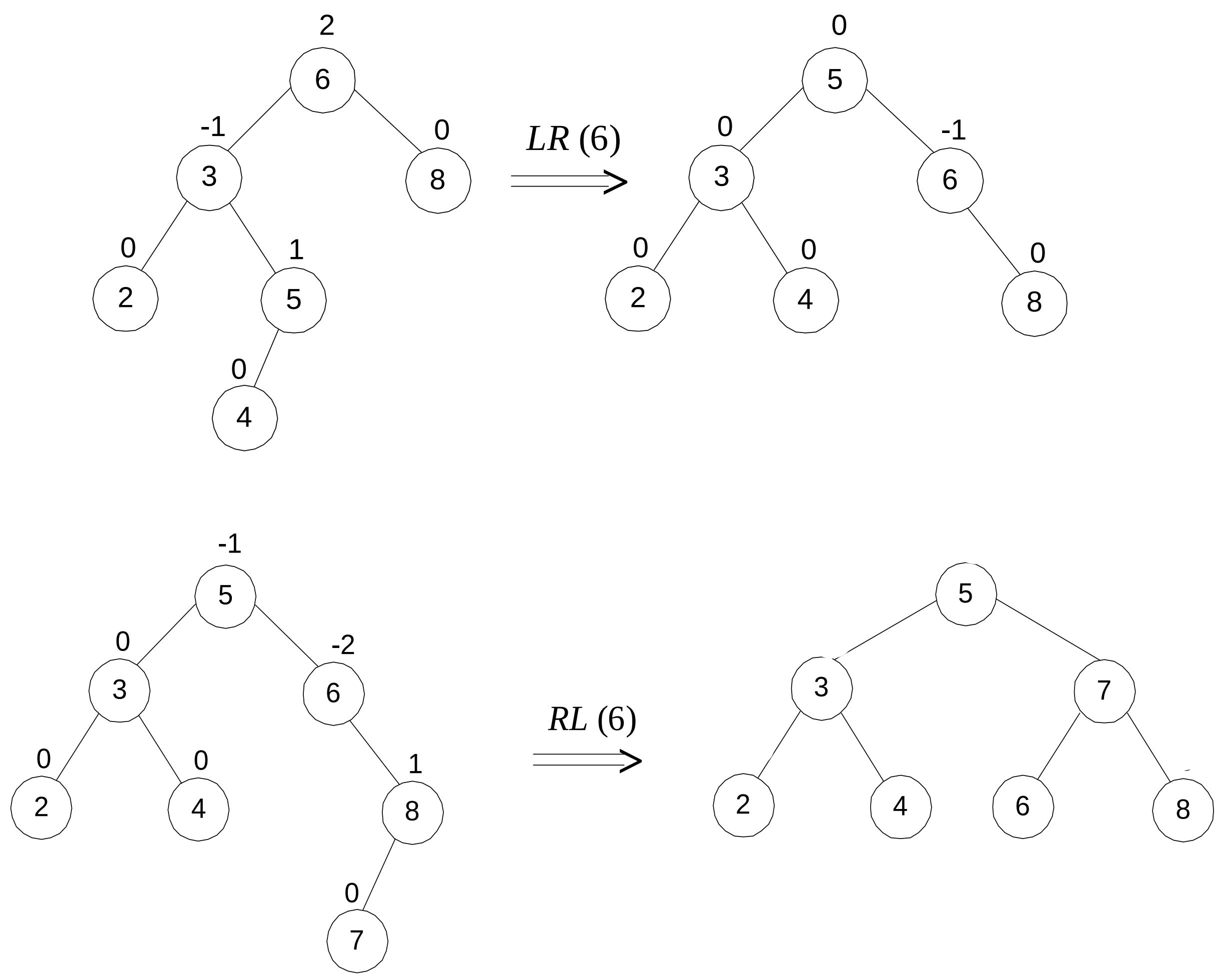


AVL tree construction - an example (cont.)

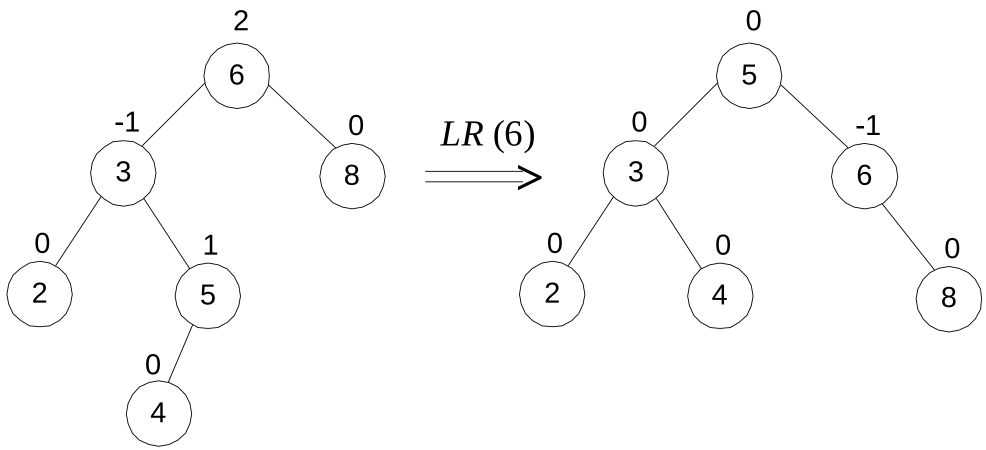


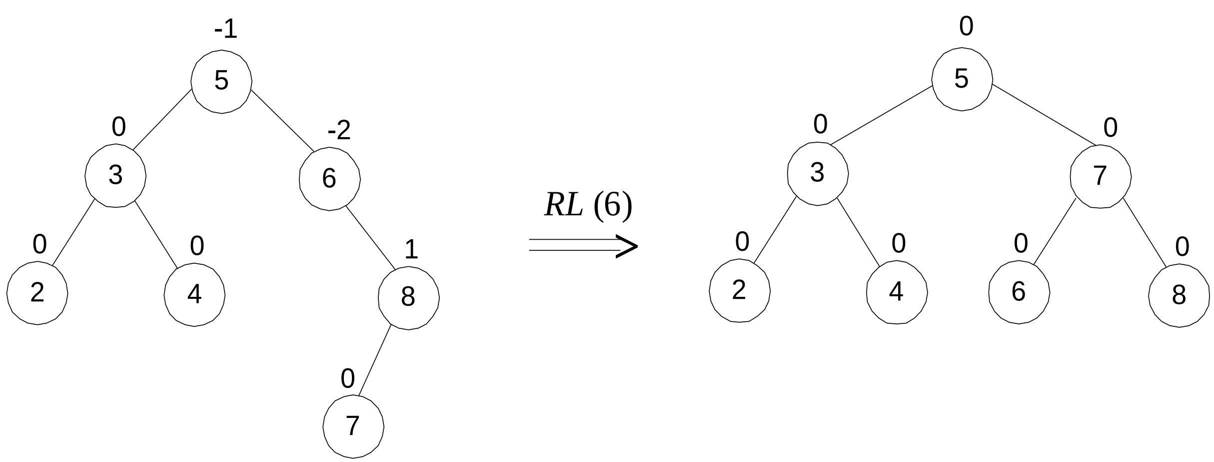


AVL tree construction - an example (cont.)



AVL tree construction - an example (cont.)



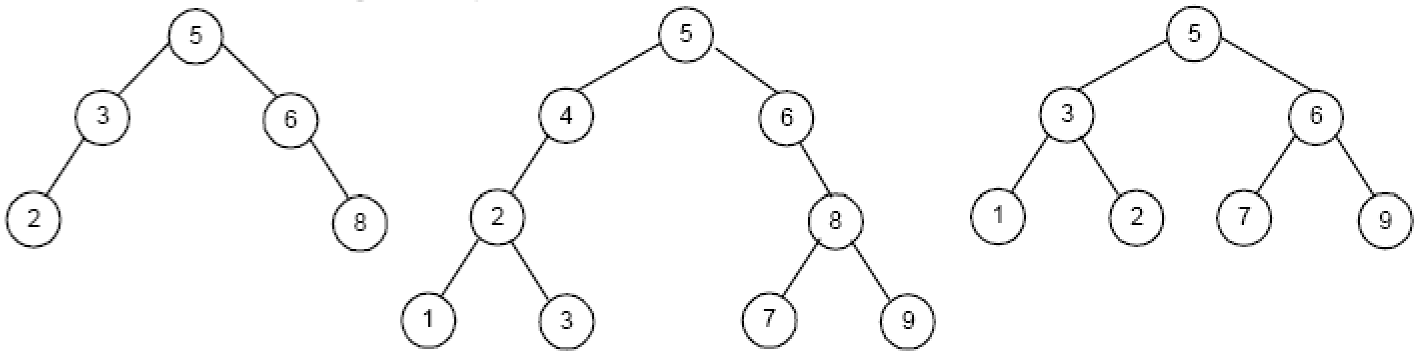


Analysis of AVL trees

- $h \leq 1.4404 \log_2(n+2) - 1.3277$
- 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 sub-trees is allowed to differ by up to a factor of 2)

Discussion

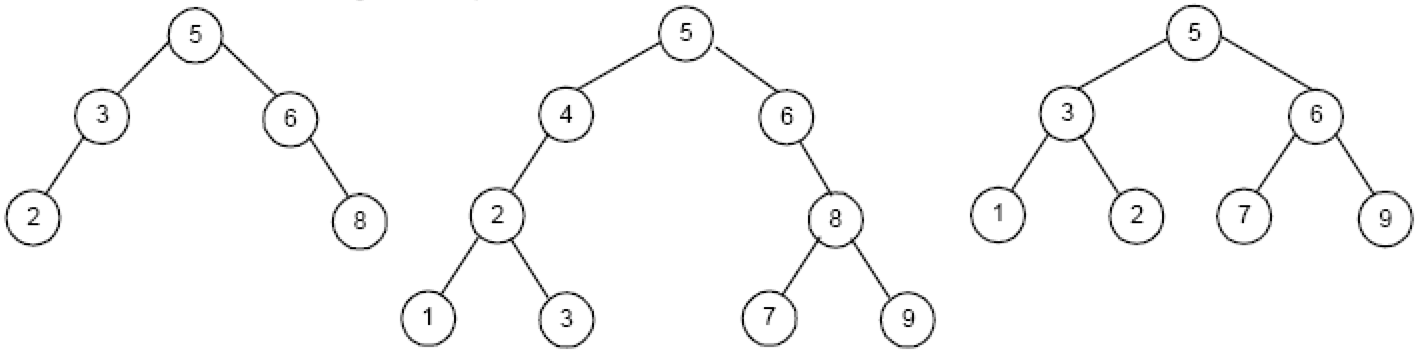
1. Which of the following binary trees are AVL trees?



(a) (b) (c)

Discussion

1. Which of the following binary trees are AVL trees?



(a) (b) (c)

1. Only (a) is an AVL tree; (b) has a node (in fact, there are two of them: 4 and 6) that violates the balance requirement; (c) is not a binary search tree because 2 is in the right subtree of 3 (and 7 is in the left subtree of 6).

Discussion

4. For each of the following lists, construct an AVL tree by inserting their elements successively, starting with the empty tree.

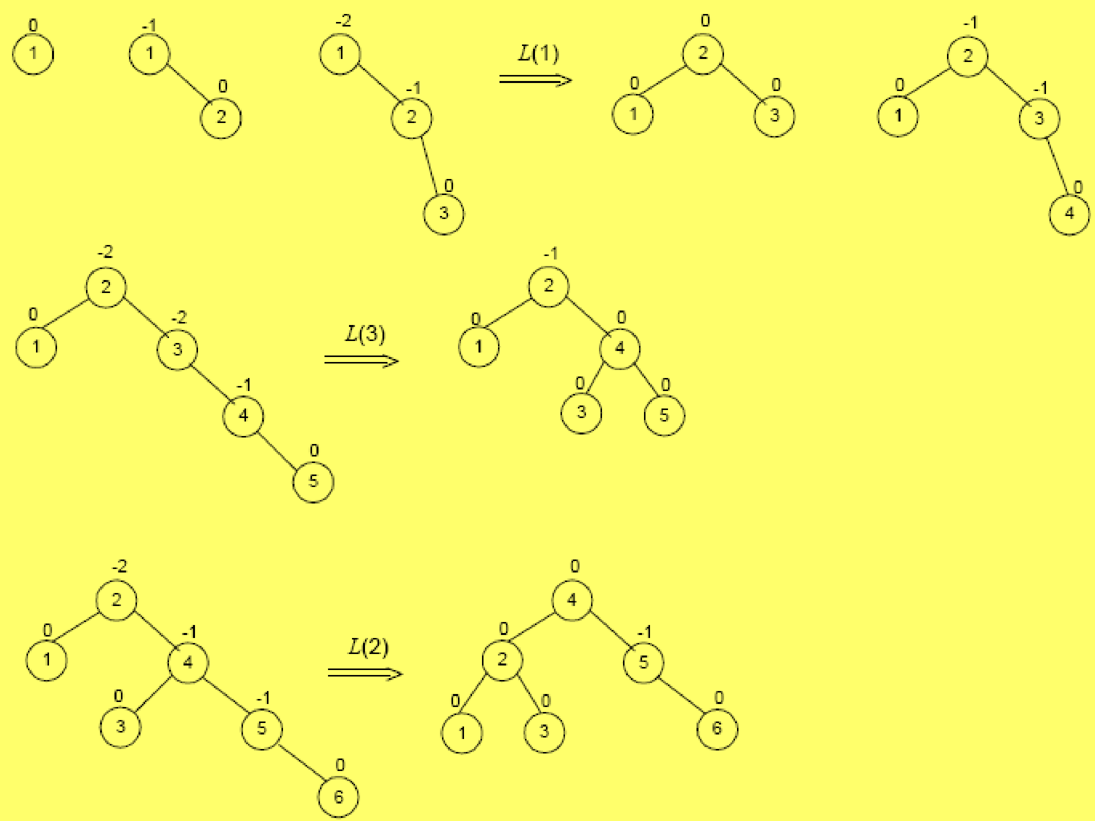
   a. 1, 2, 3, 4, 5, 6

   b. 6, 5, 4, 3, 2, 1

   c. 3, 6, 5, 1, 2, 4

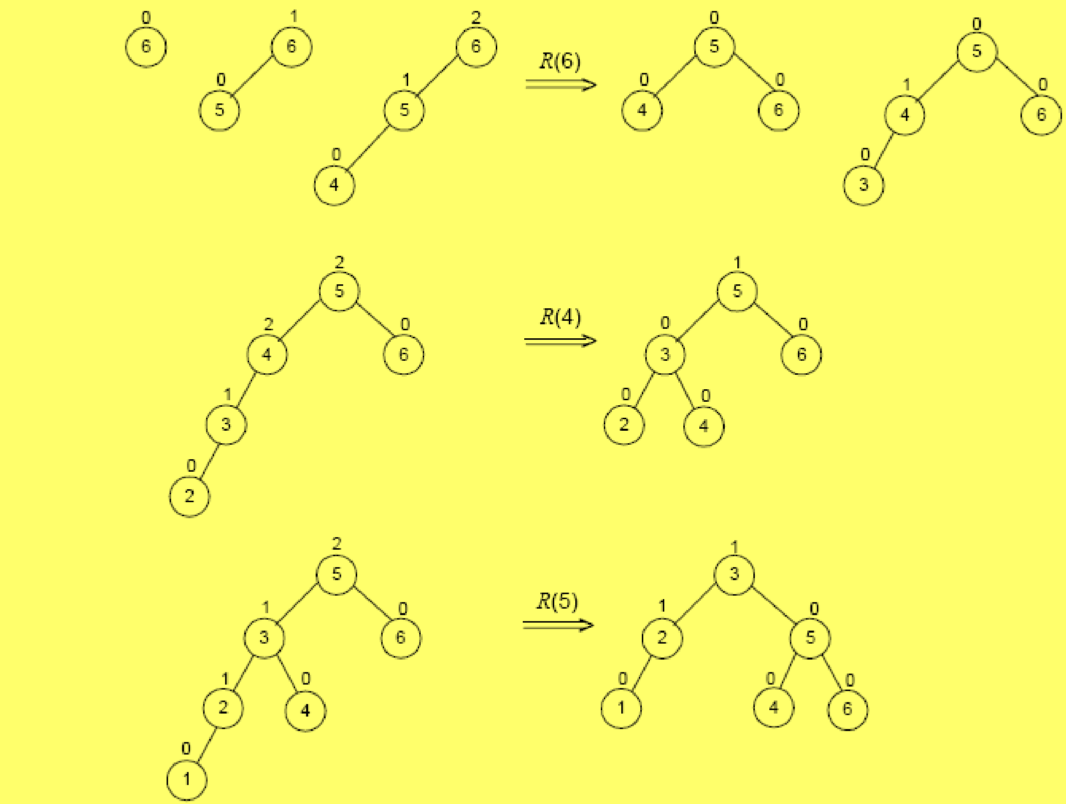
Discussion

a. Construct an AVL tree for the list 1, 2, 3, 4, 5, 6.



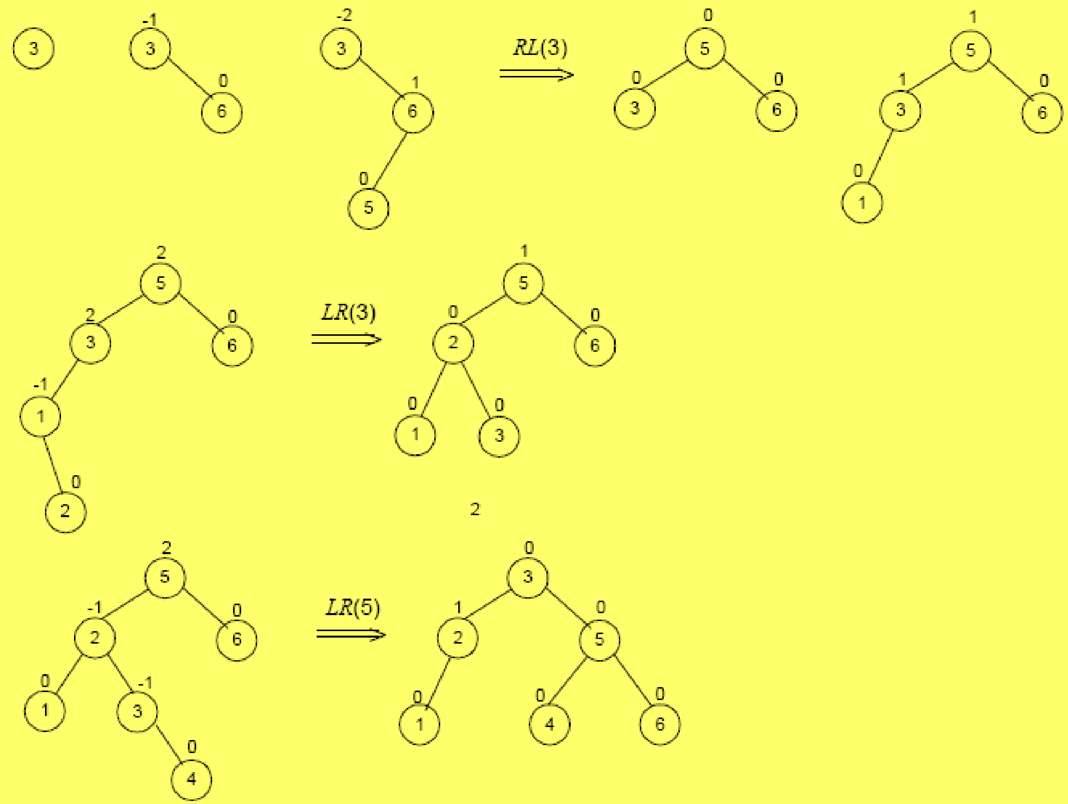
Discussion

b. Construct an AVL tree for the list 6, 5, 4, 3, 2, 1.



Discussion

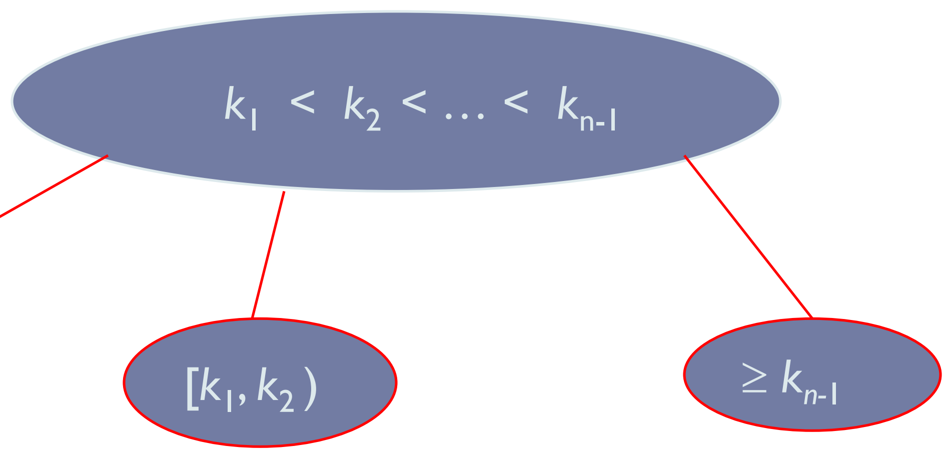
c. Construct an AVL tree for the list 3, 6, 5, 1, 2, 4.



Multi-way Search Trees

A multi-way search tree is a search tree that allows more than one key in the same node of the tree.

A node of a search tree is called an n-node if it contains $n - 1$ ordered keys (which divide the entire key range into n intervals pointed to by the node's n links to its children):

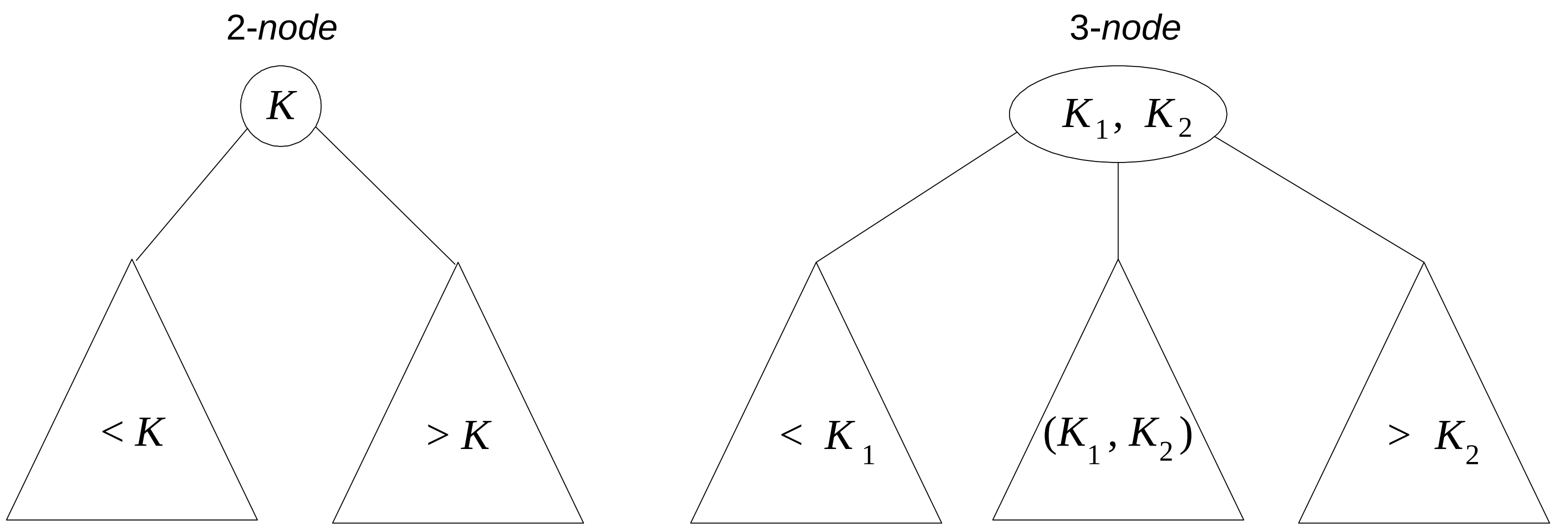


Every node in a classical binary search tree is a 2-node

2-3 Tree

A 2-3 tree is a search tree that

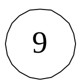
- may have 2-nodes and 3-nodes
- height-balanced (all leaves are on the same level)



A 2-3 tree is constructed by successive insertions of keys given, with a new key always inserted into a leaf of the tree. If the leaf is a 3-node, it's split into two with the middle key promoted to the parent.

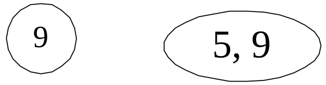
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



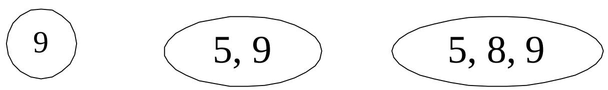
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



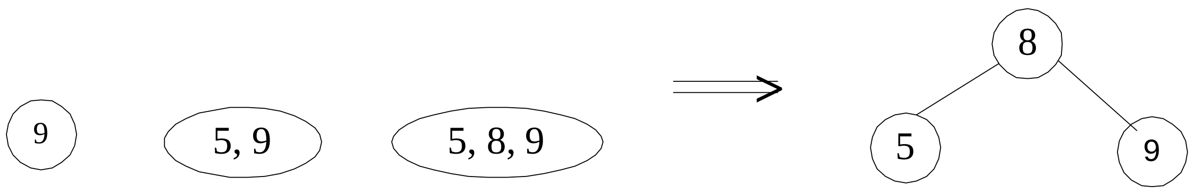
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



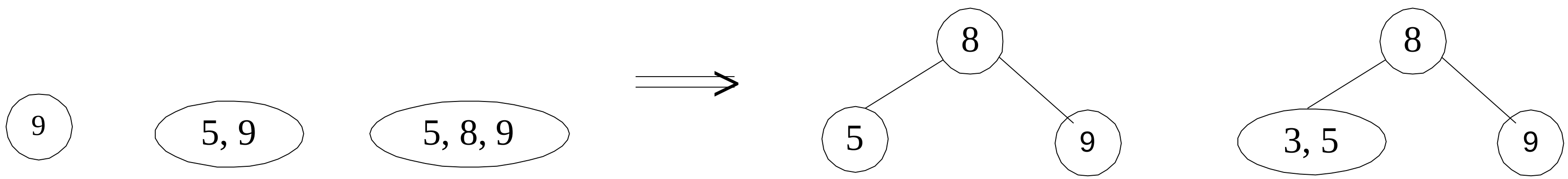
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



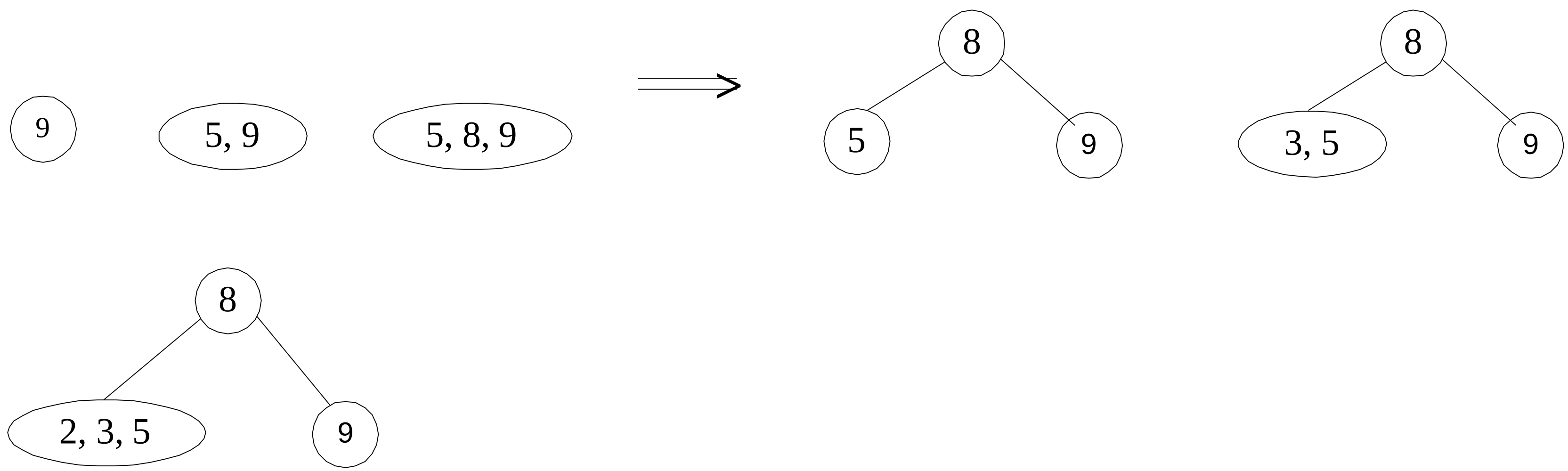
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



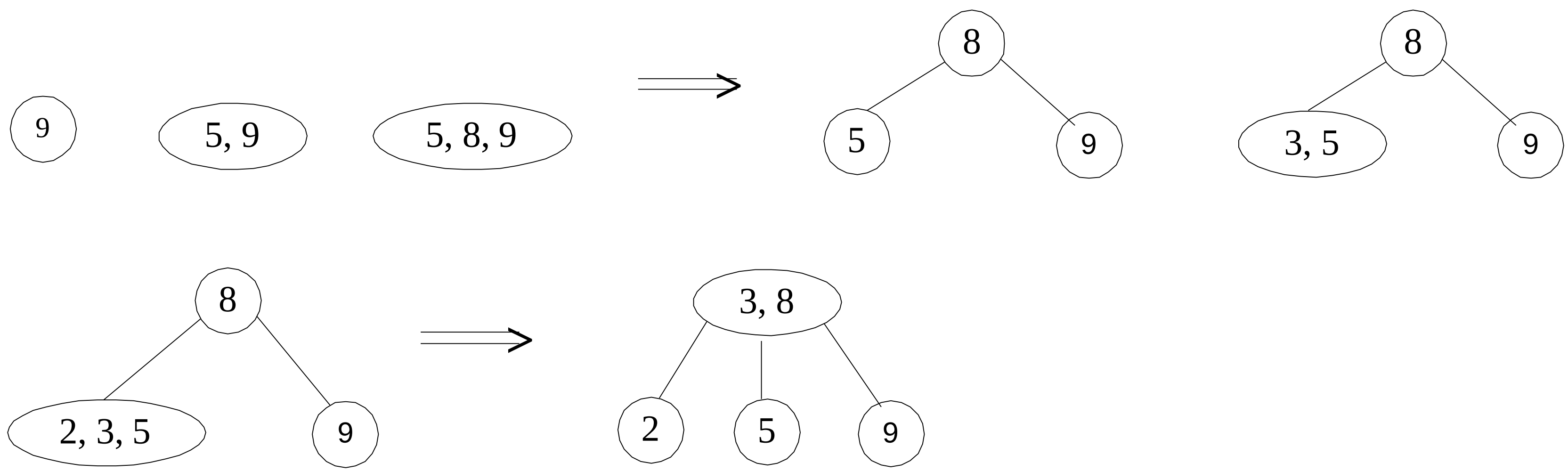
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



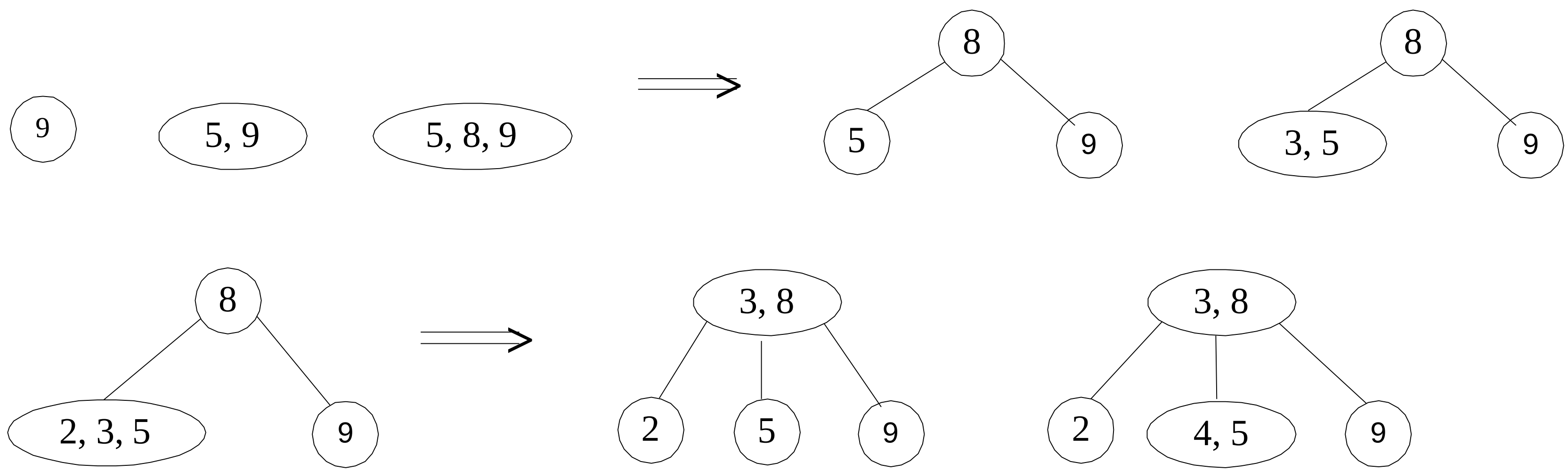
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



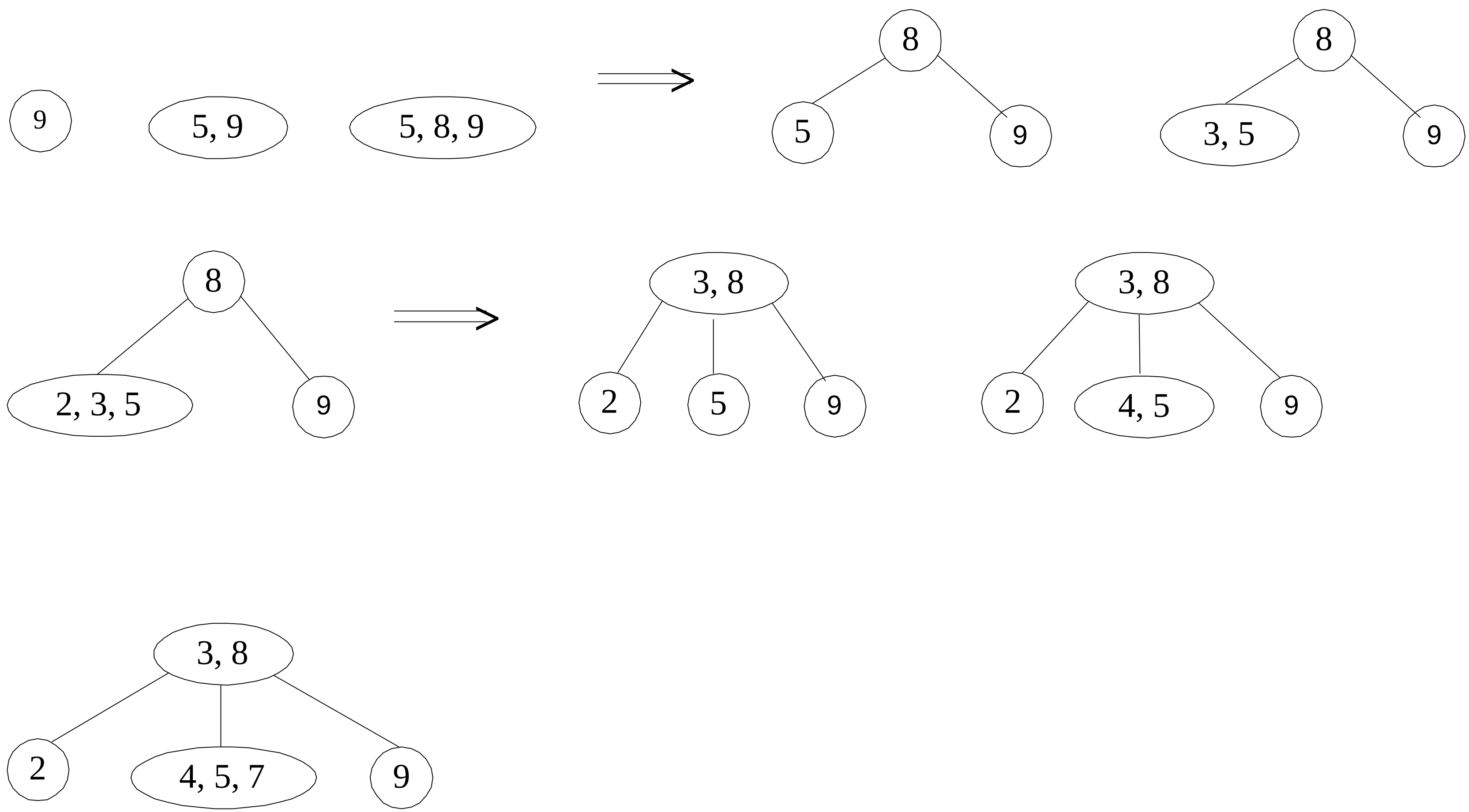
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



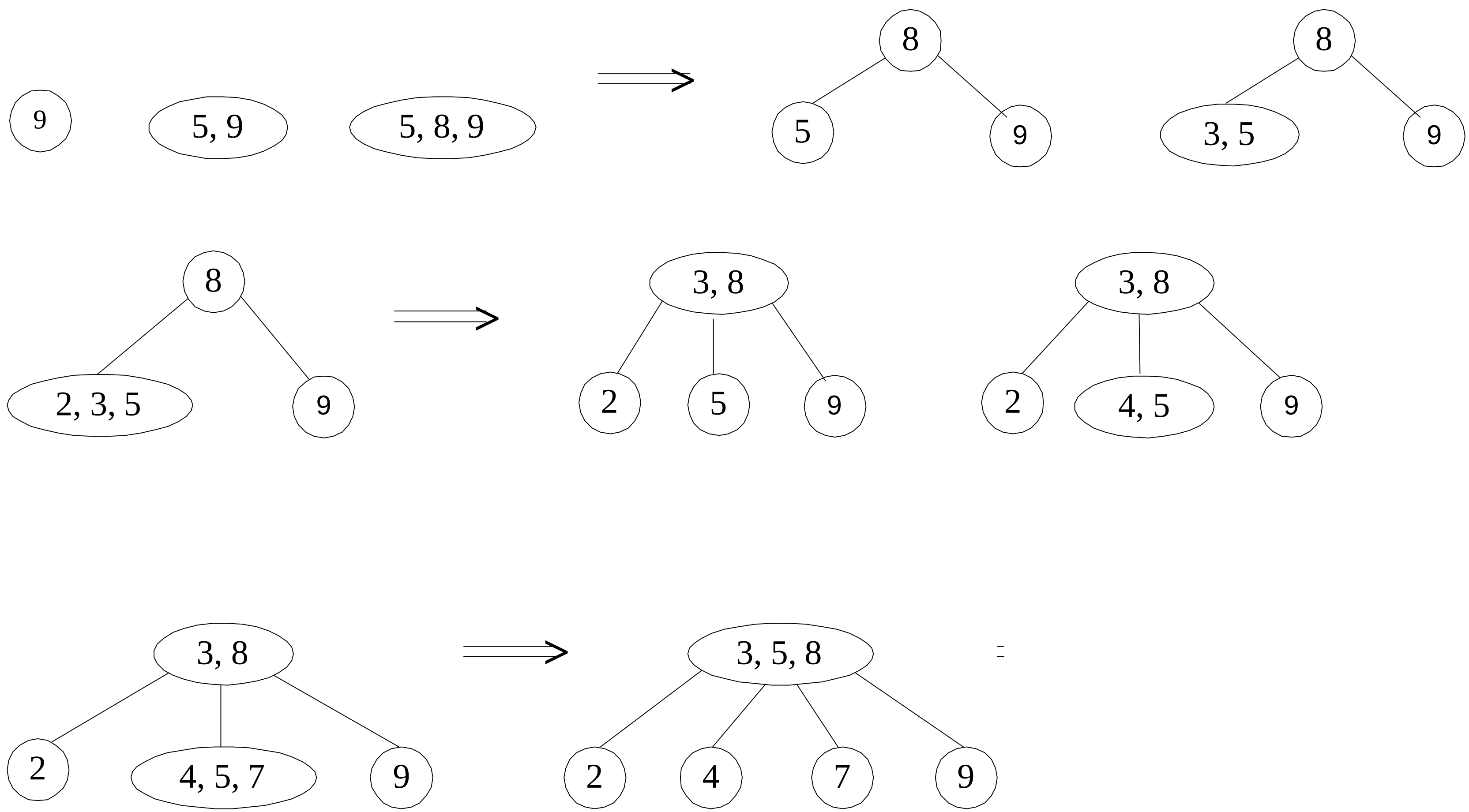
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



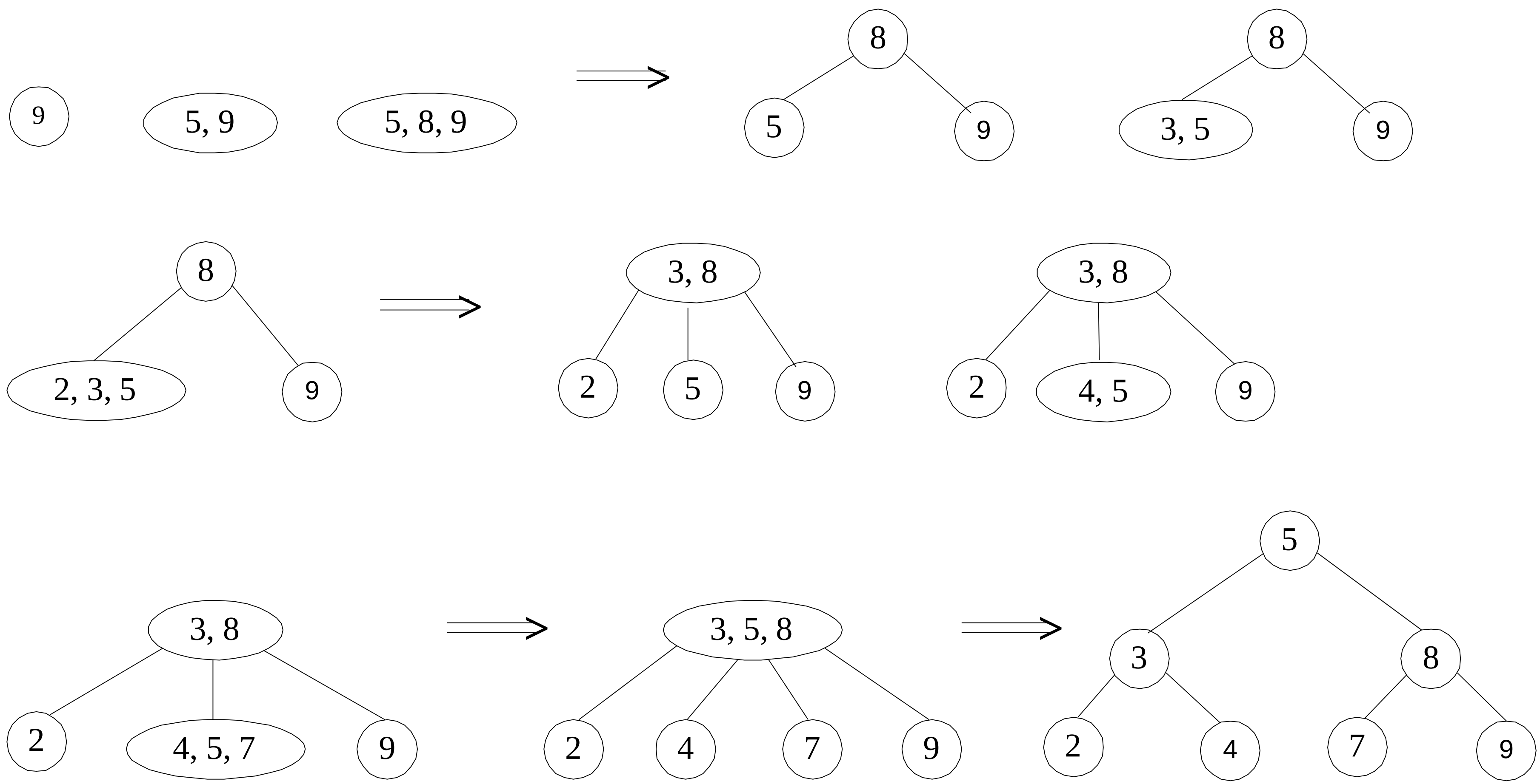
2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



2-3 tree construction – an example

Construct a 2-3 tree the list 9, 5, 8, 3, 2, 4, 7



Analysis of 2-3 trees

- $\log_3(n+1) - 1 \leq h \leq \log_2(n+1) - 1$
- Search, insertion, and deletion are in $\theta(\log n)$
- The idea of 2-3 tree can be generalized by allowing more keys per node
  - 2-3-4 trees
  - B-trees