Data Structures and Algorithms

Rao Muhammad Umer
Lecturer,
CS and IT Department,
The University of Lahore.
Web: raoumer.com

outline

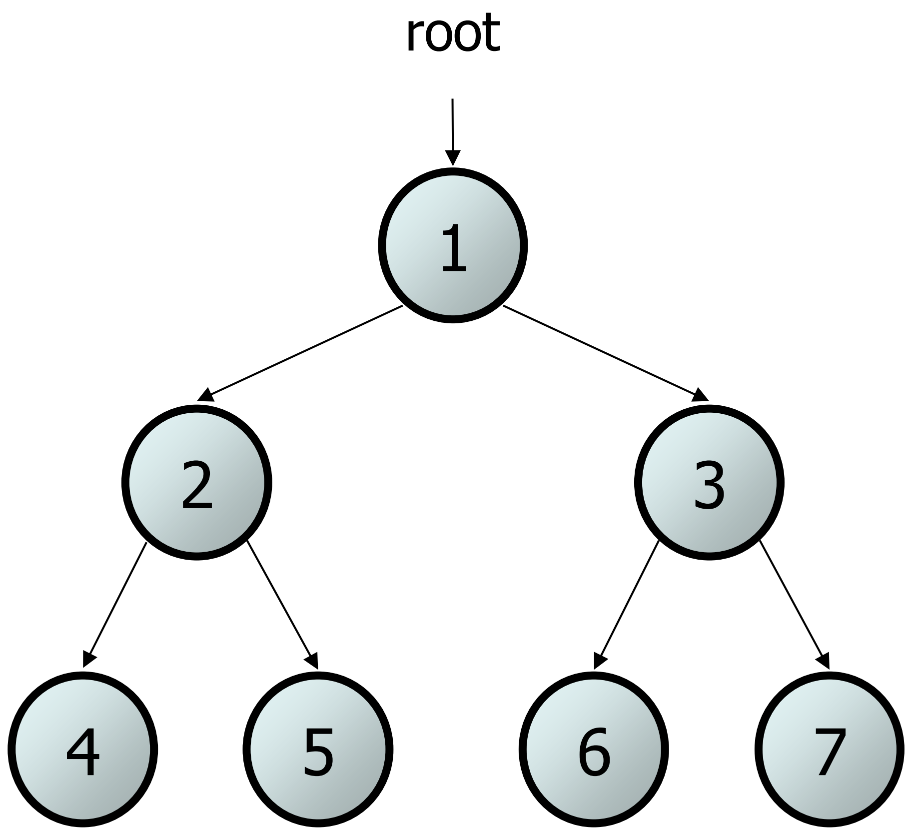
Trees

- What are Trees?
- Binary Trees Concepts
- Binary Search Tree
- Representation of Binary Tree
  - As an Array
  - As a Linked-list
- Operations on a BST
  - Searching, Insertion, Deletion
- Expression Trees
  - Prefix, Postfix, Infix expressions
- Reconstruction Tree
- Balanced Trees
- AVL Trees

Trees

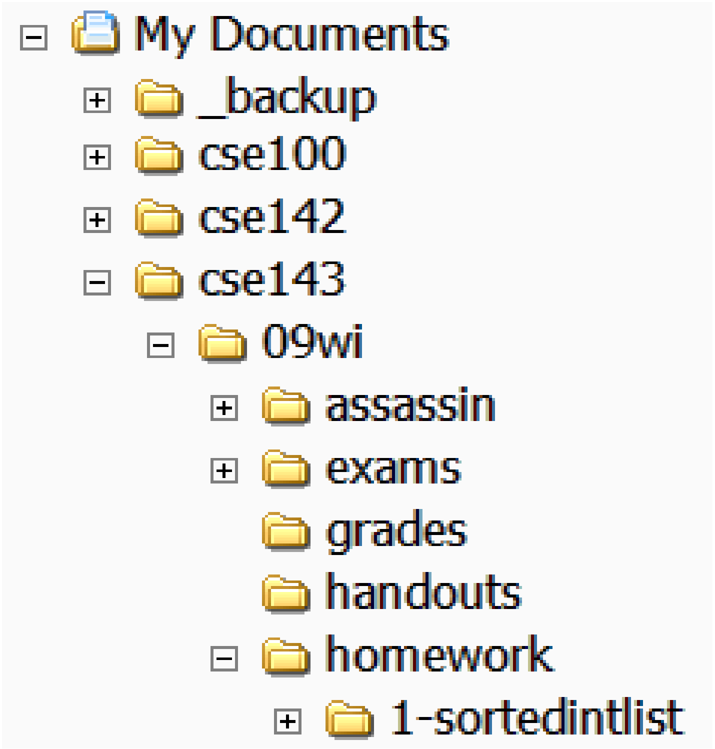
- **tree**: A directed, acyclic structure of linked nodes.
  - ***directed***: Has one-way links between nodes.
  - ***acyclic***: No path wraps back around to the same node twice.
  - **binary tree:** One where each node has at most two children.

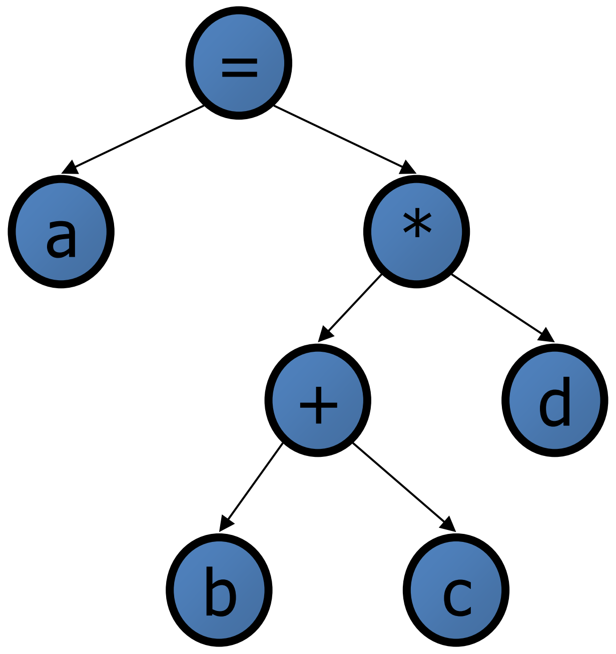
- A binary tree can be defined as either:
  - empty (`null`), or
  - a **root** node that contains:
    - **Data**
    - a **left** subtree and a **right** subtree
      - Either (or both) subtrees could be empty.



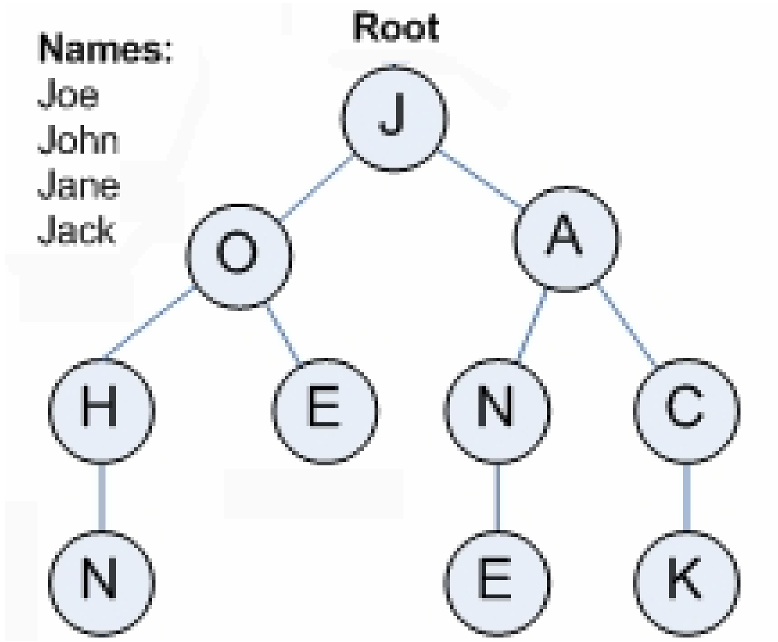
Trees in computer science

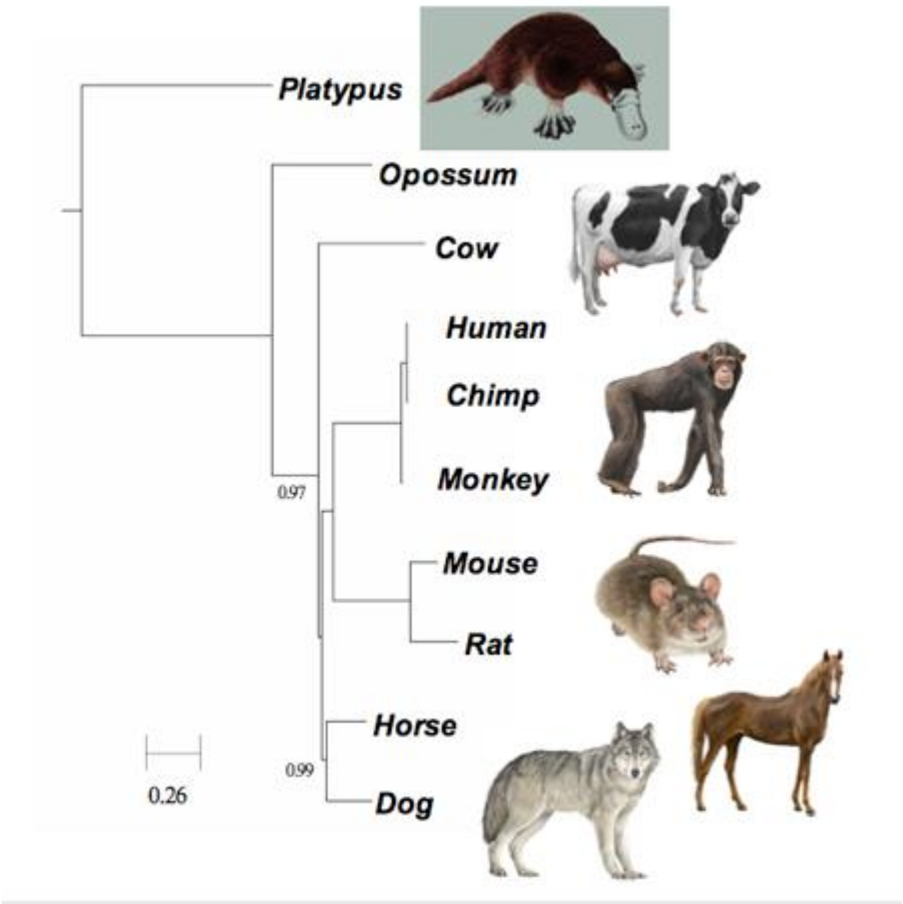
- folders/files on a computer





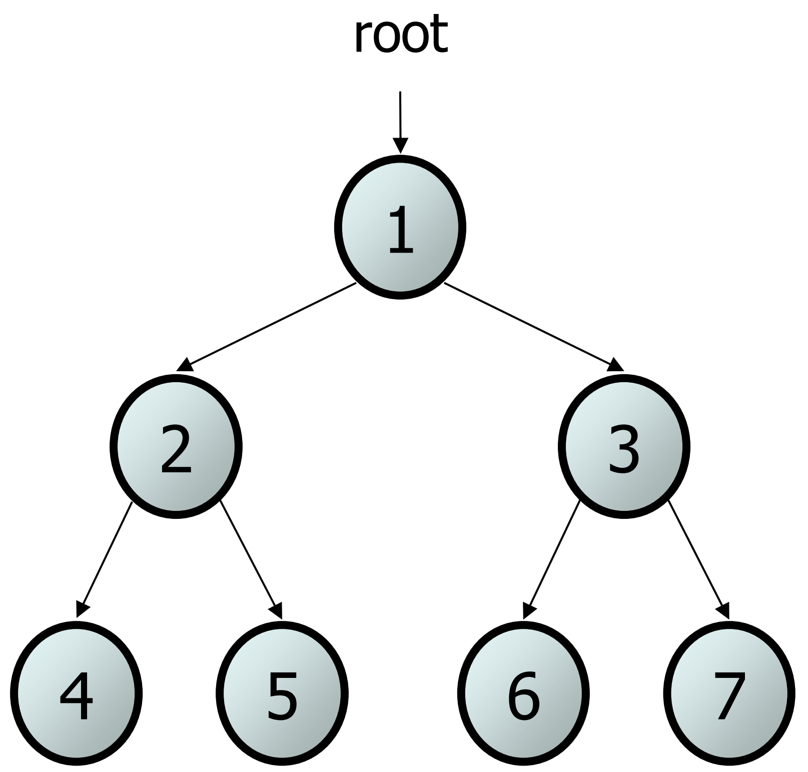
- family genealogy; organizational charts
- AI: decision trees
- compilers: parse tree
  - a = (b + c) * d;
- cell phone T9





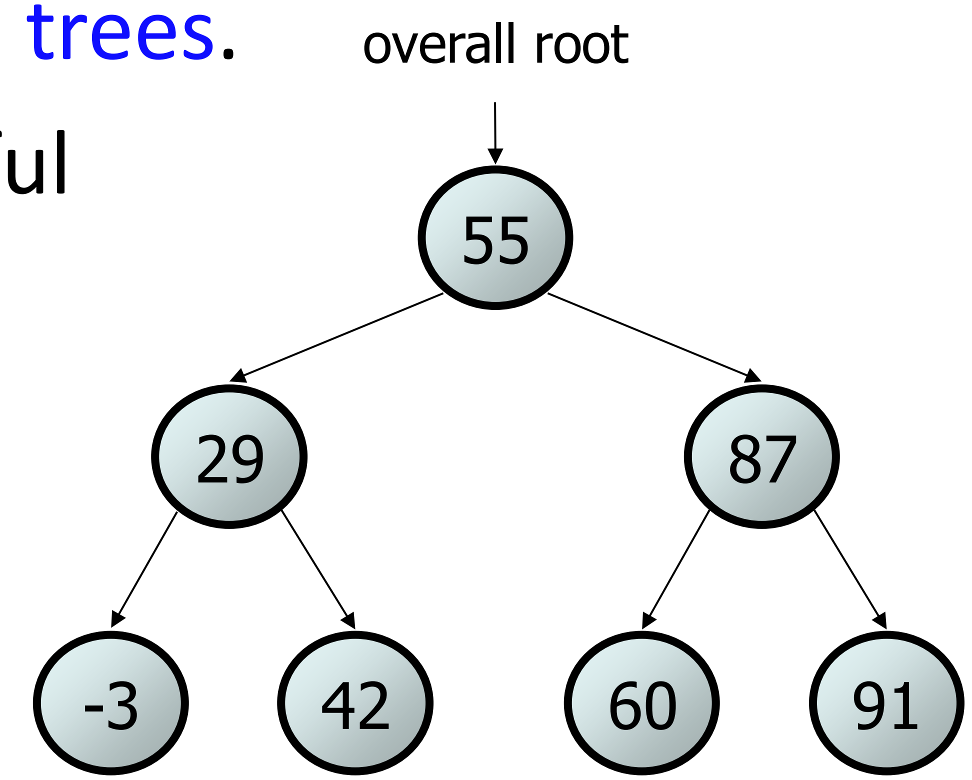
Terminology

- **node**: an object containing a data value and left/right children
- **root**: topmost node of a tree
- **leaf**: a node that has no children
- **branch**: any internal node; neither the root nor a leaf
- **parent**: a node that refers to this one
- **child**: a node that this node refers to
- **sibling**: a node with common parent



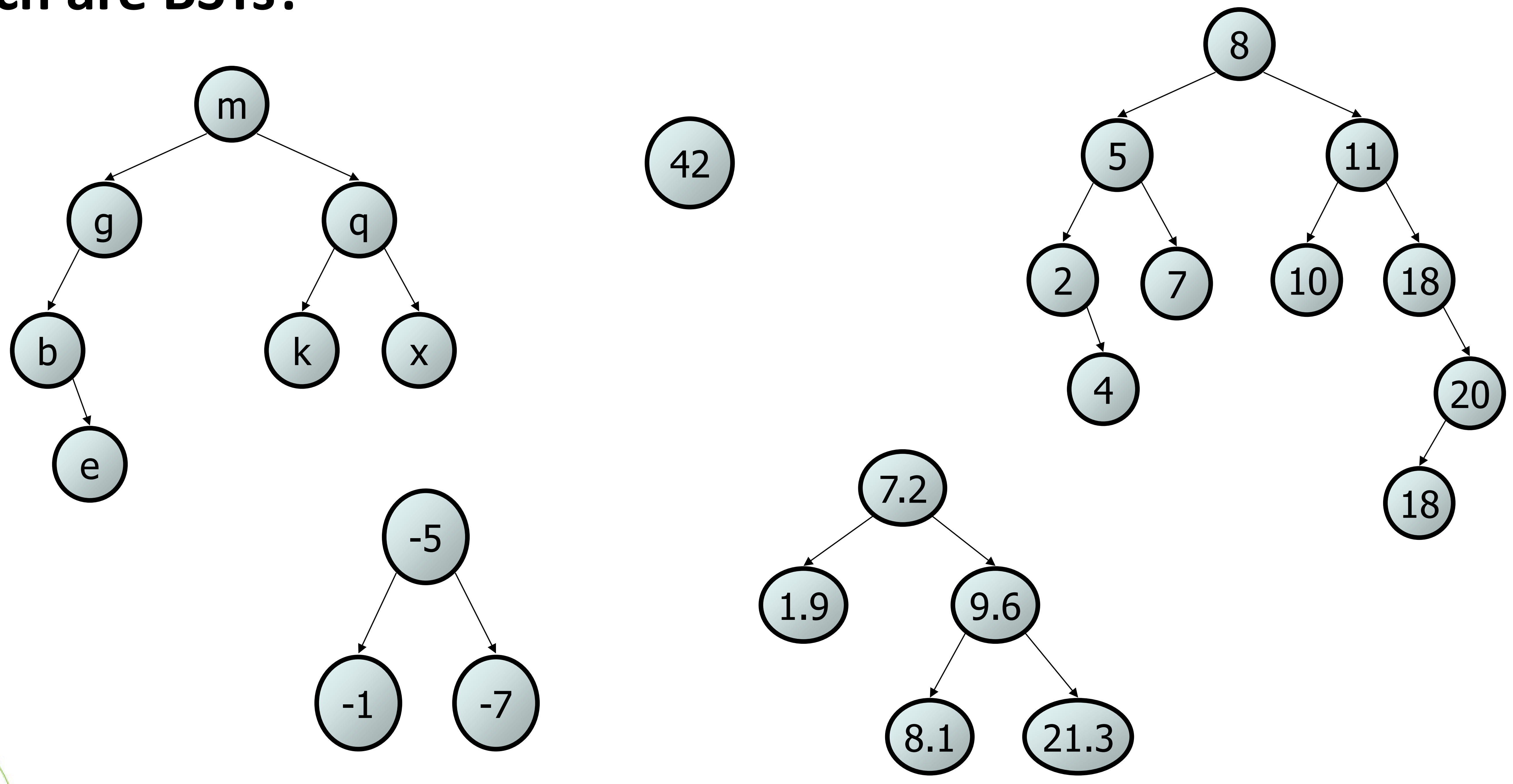
Binary search trees

- **Binary search tree ("BST"):** a binary tree that is either:
  - empty (`null`), or
  - a root node R such that:
    - every element of R's left subtree contains data "less than" R's data,
    - every element of R's right subtree contains data "greater than" R's,
    - R's left and right subtrees are also binary search trees.
- **BSTs** store their elements in sorted order, which is helpful for searching/sorting tasks.
- See [animation](#) of building a BST



Exercise

Which are BSTs?



Programming with Binary Trees

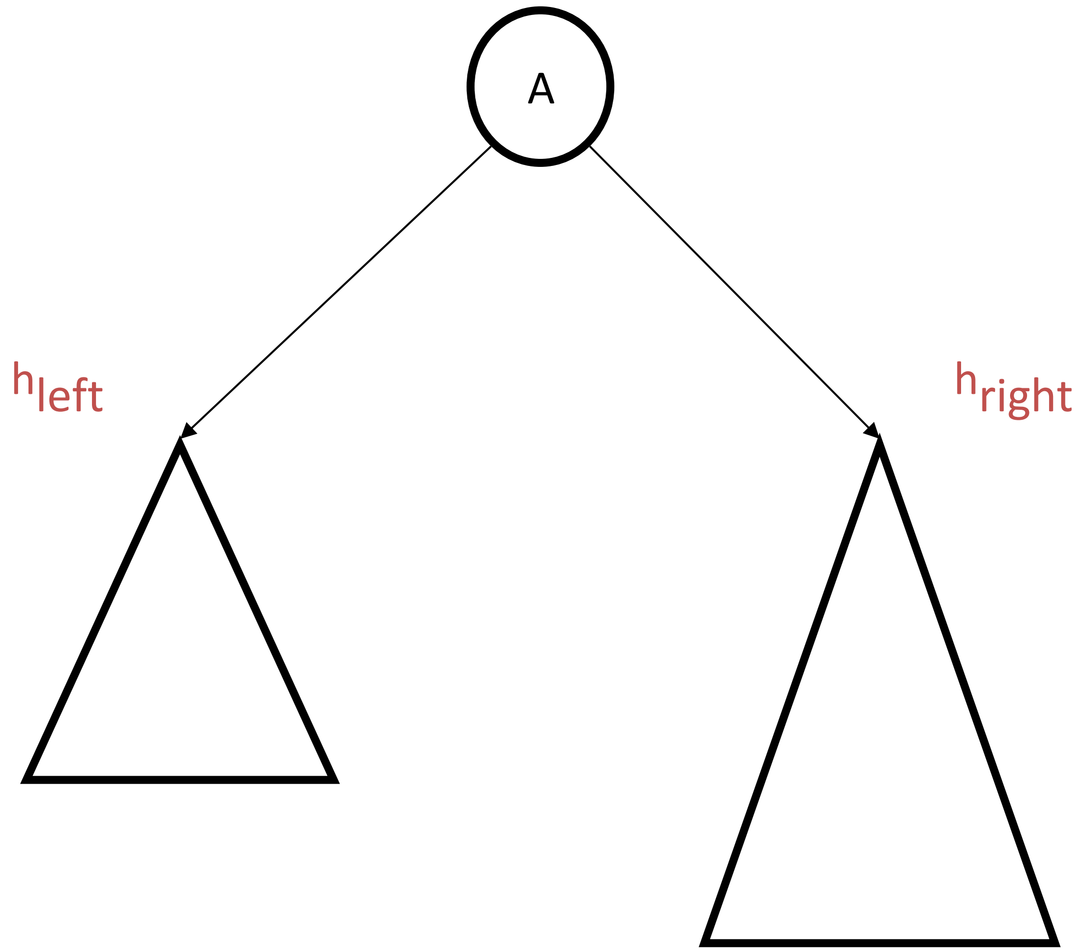
- Many tree algorithms are **recursive**
  - Process current node, recurse on subtrees
  - Base case is usually empty tree (`null`)

- **traversal**: An examination of the elements of a tree.
  - A pattern used in many tree algorithms and methods

- Common orderings for traversals:
  - **pre-order**: process root node, then its left/right subtrees
    - See animation of working of pre-order
  - **in-order**: process left subtree, then root node, then right
    - See animation of working of pre-order
  - **post-order**: process left/right subtrees, then root node
    - See animation of working of pre-order

Tree height calculation

- Height is max number of edges from root to leaf
  - height(null) = -1
  - height(1) = 0
  - height(A)?
    - Hint: it's recursive!



- Height = max (height(left), height(right)) + 1
- Height (null) = -1
- Runtime: O(N) visit each node once.

Binary Trees: Some Numbers

- Recall: **height of a tree = length of longest path from the root to a leaf.**
- For binary tree of height ***h***:
  - **max # of leaves:**

    2^h

  - **max # of nodes:**

    $2^{(h+1)} - 1$

  - **min # of leaves:**

    1

  - **min # of nodes:**

    $h + 1$

Representation of a Binary Trees in Memory

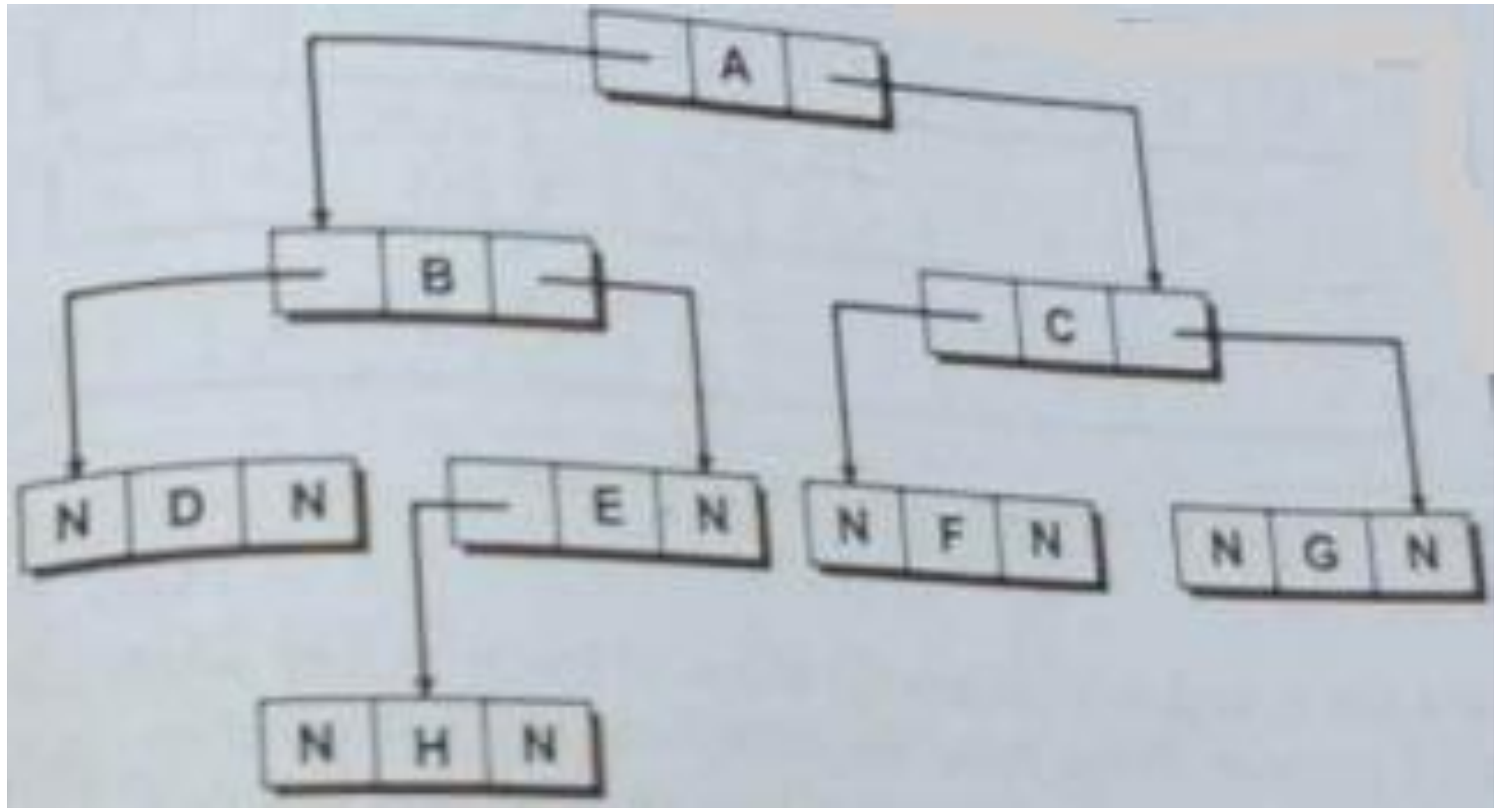
- **Node of Binary Tree:**

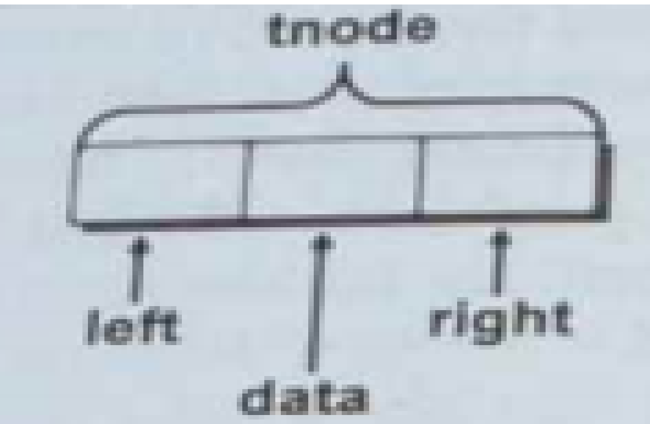
```
struct tnode
    {
        tnode *left;
        int data;
        tnode *rigth;
    }
```

- There are two ways to represent a binary tree:
    - **Linked representation of a binary tree**
    - **Array representation of a binary tree**

Representation of a Binary Trees in Memory

- **Linked representation of a binary tree**

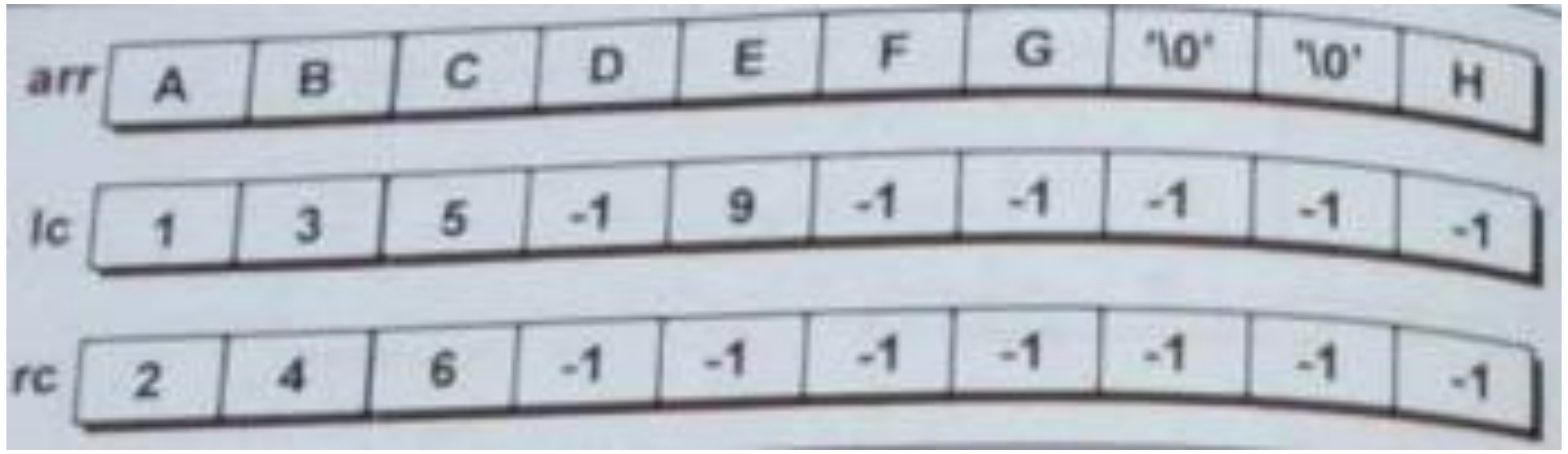




- See animation of building tree using linked list

Representation of a Binary Trees in Memory

- **Array representation of a binary tree**



arr	A	B	C	D	E	F	G	'\0'	'\0'	H
lc	1	3	5	-1	9	-1	-1	-1	-1	-1
rc	2	4	6	-1	-1	-1	-1	-1	-1	-1

- See source code in C++ of building tree using array

Binary Search Tree (BST)

- **Implementation of a binary search tree**
  - See source code in C++ of building binary search tree
- **Operations on a BST**
  - **Searching**
  - **Insertion**
  - **Deletion**

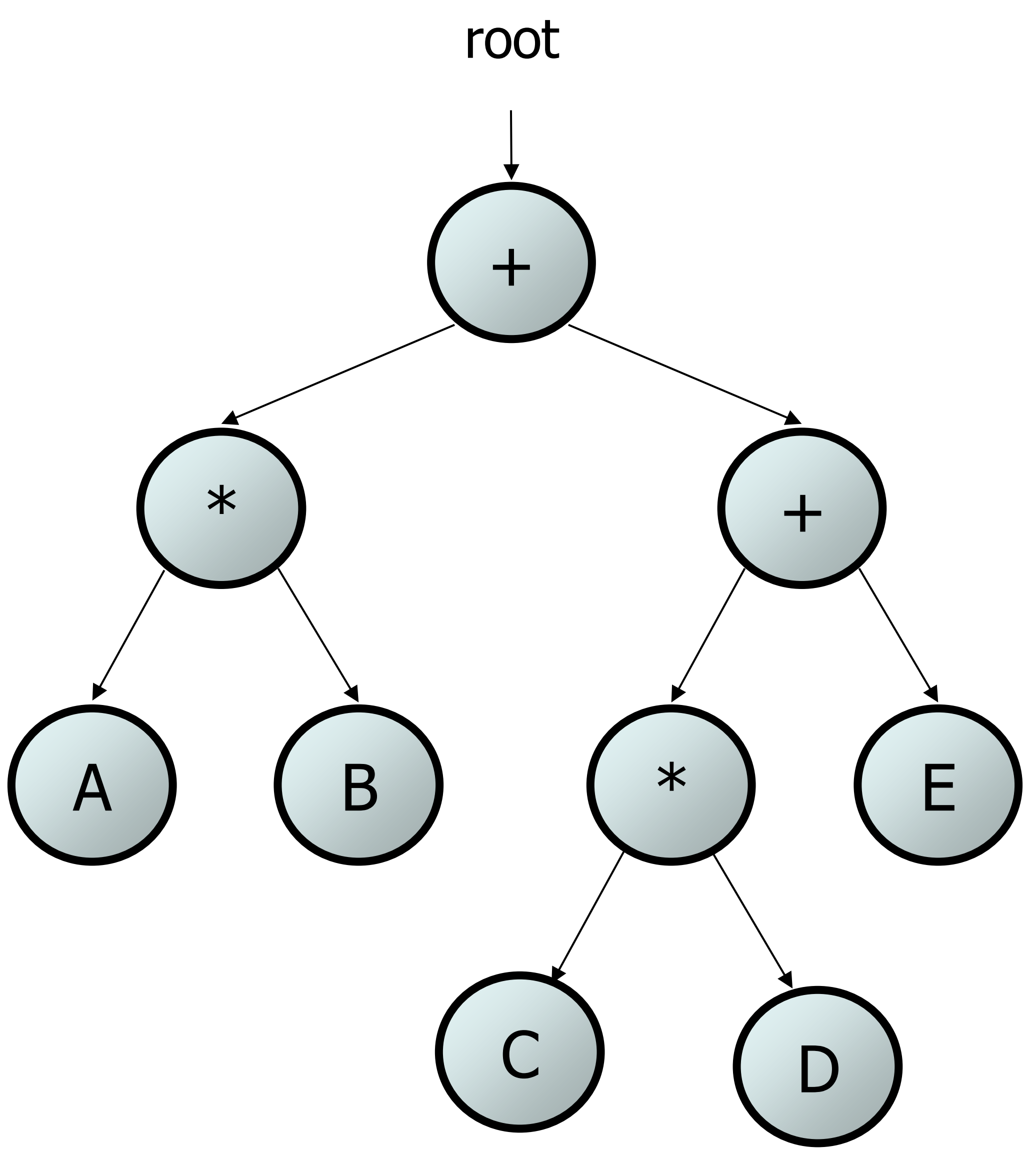
Operations of Binary Search Tree (BST)

- Operations on a binary search tree
  - See animation of operations on a BST
  - See source code of operations on a BST

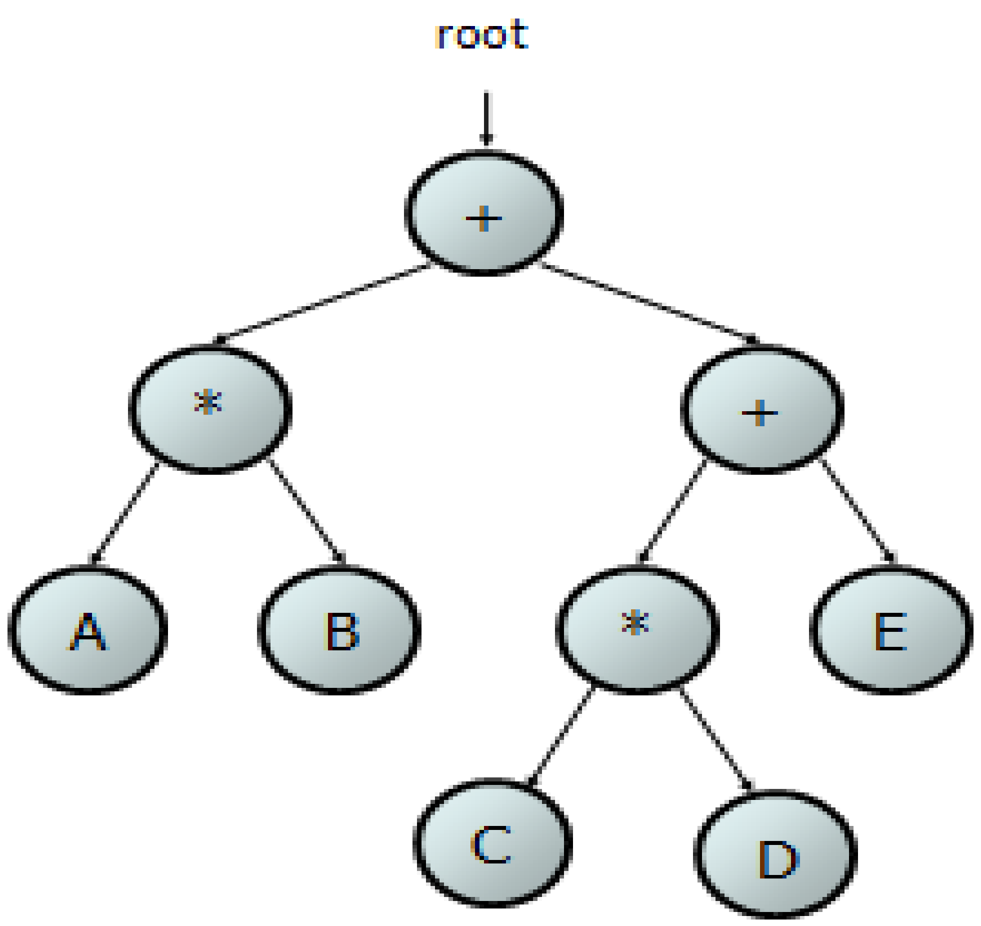
Expression Binary Trees

- **Expression Trees**
  - Arithmetic expression: **A * B + C * D + E**

  - **Prefix form:**
    - **Pre-order** traversal of expression tree
  - **Infix form:**
    - **In-order** traversal of expression tree
  - **Postfix form:**
    - **Post-order** traversal of expression tree



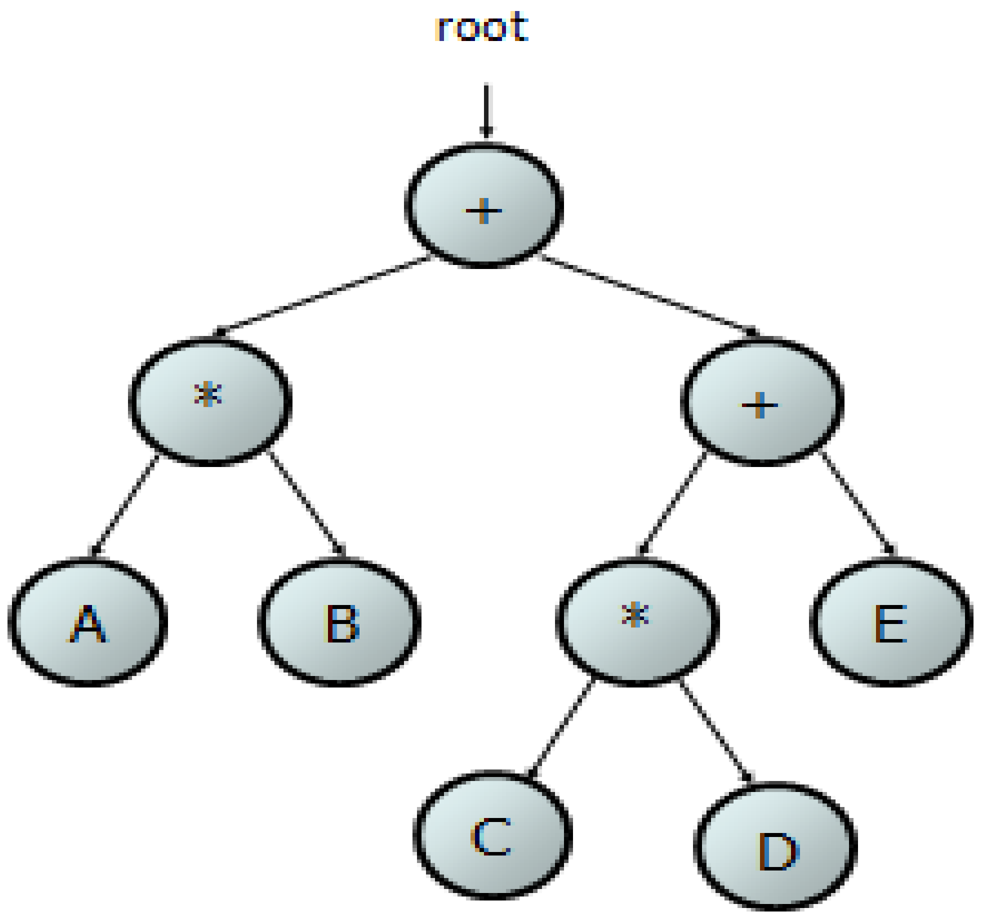
Preorder Of Expression Tree



+ * A B + * C D E

Gives prefix form of expression!

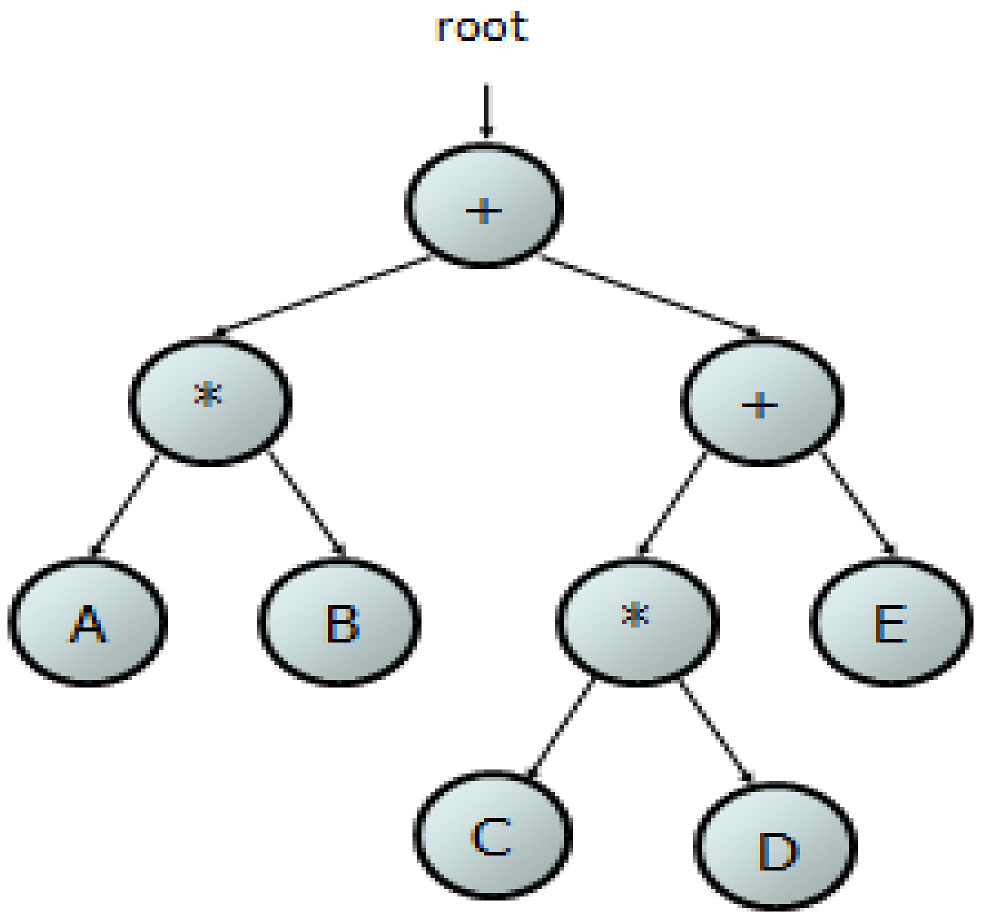
Inorder Of Expression Tree



A * B + C * D + E

Gives infix form of expression!

Postorder Of Expression Tree

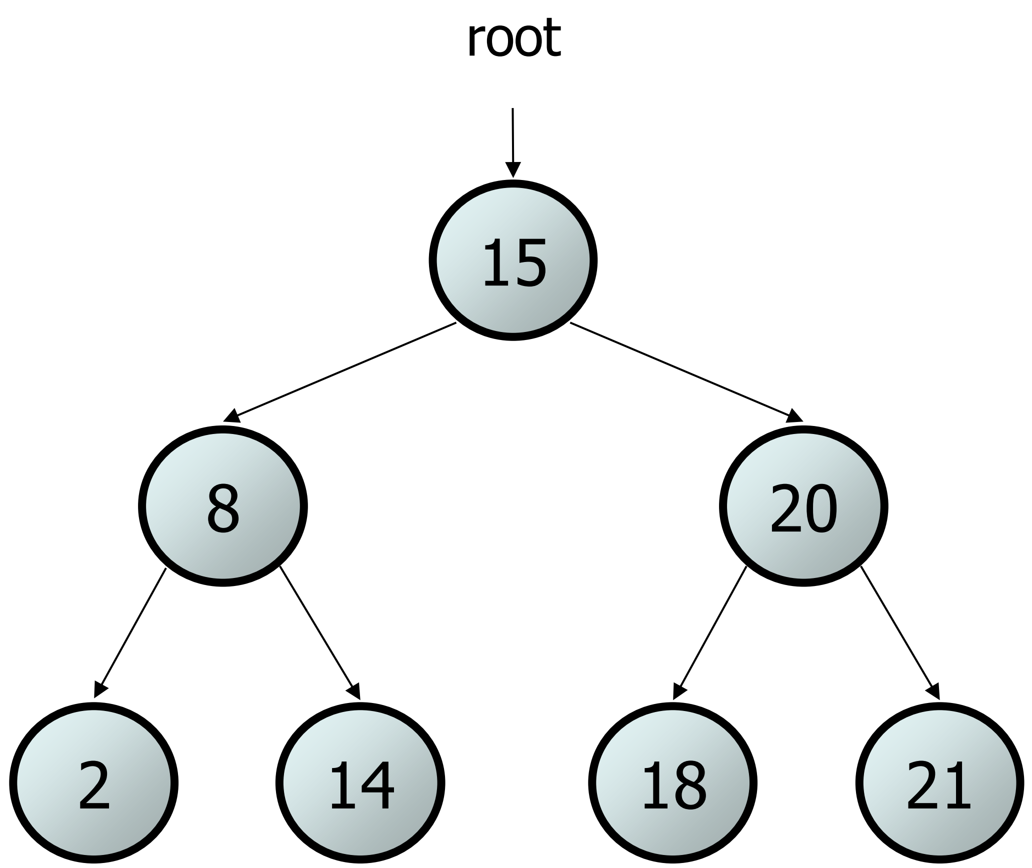


AB*CD*E++

Gives postfix form of expression!

Traversal Applications

- **Make a clone.**
- **Determine height.**
- **Determine number of nodes.**



Binary Tree Construction

- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely.

Some Examples

preorder

= ab

inorder

= ab

postorder

= ab

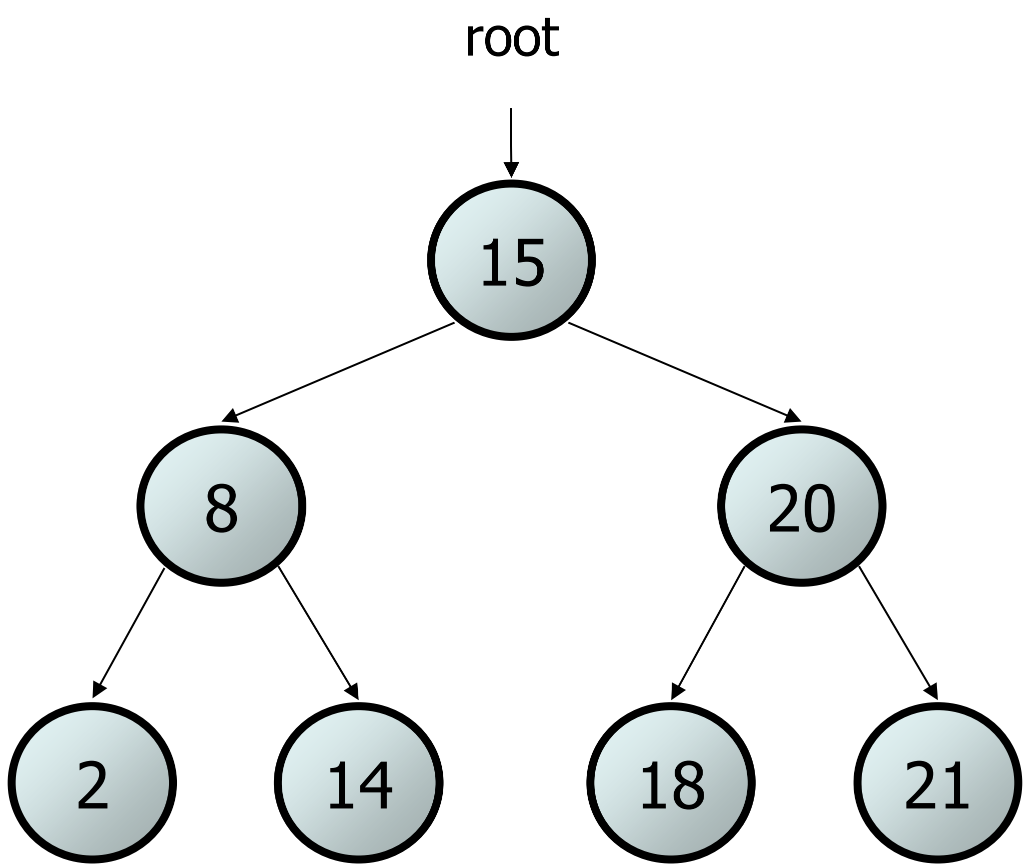
A Balanced Tree

- Values: 2 8 14 15 18 20 21
  - Order added: 15, 8, 2, 20, 21, 14, 18

- Different tree structures possible
  - Depends on order inserted

- **7** nodes, expected height **log 7 ≈ 3**

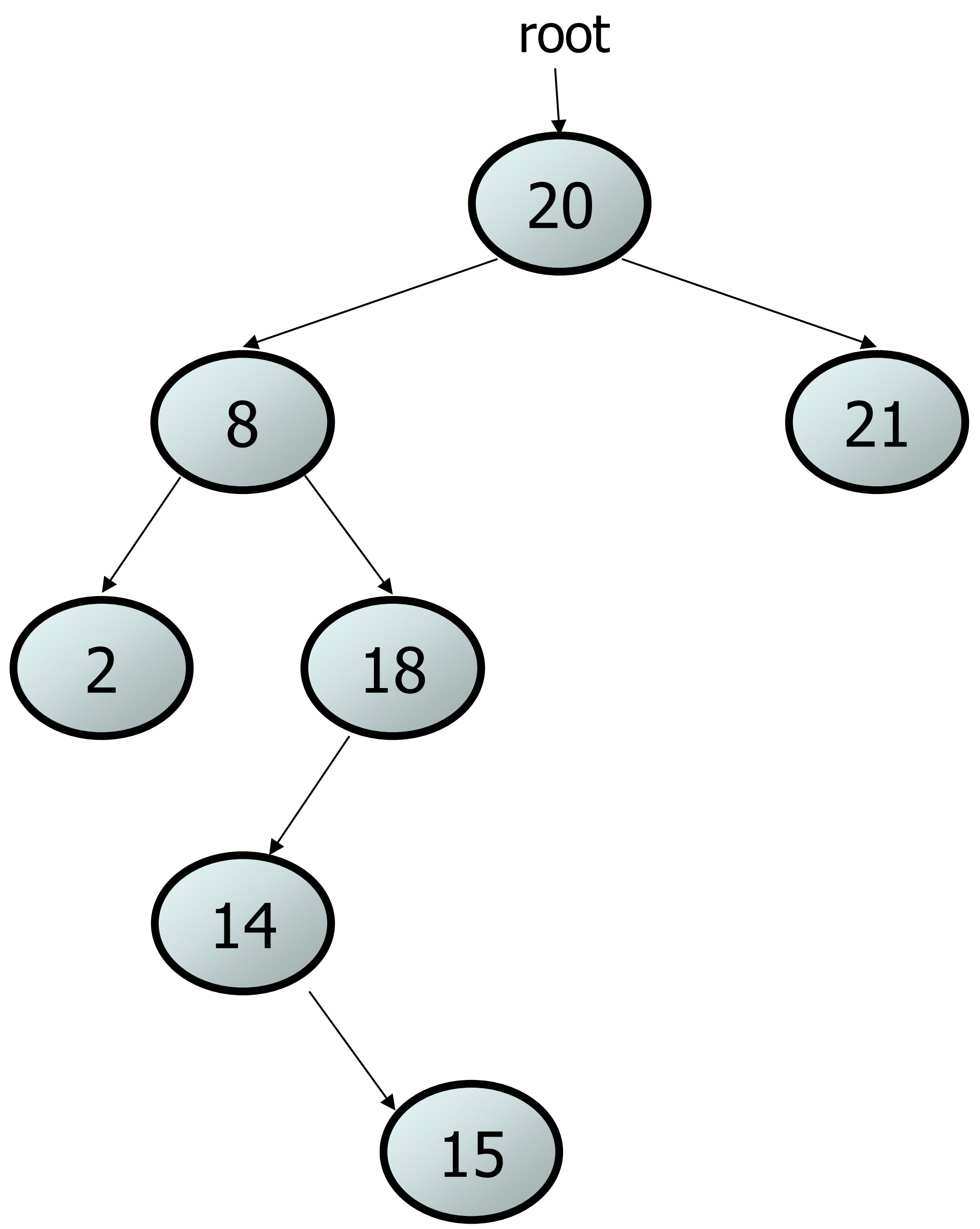
- Perfectly **balanced**



Mostly Balanced Tree

- Same Values: 2 8 14 15 18 20 21
  - Order added: 20, 8, 21, 18, 14, 15, 2

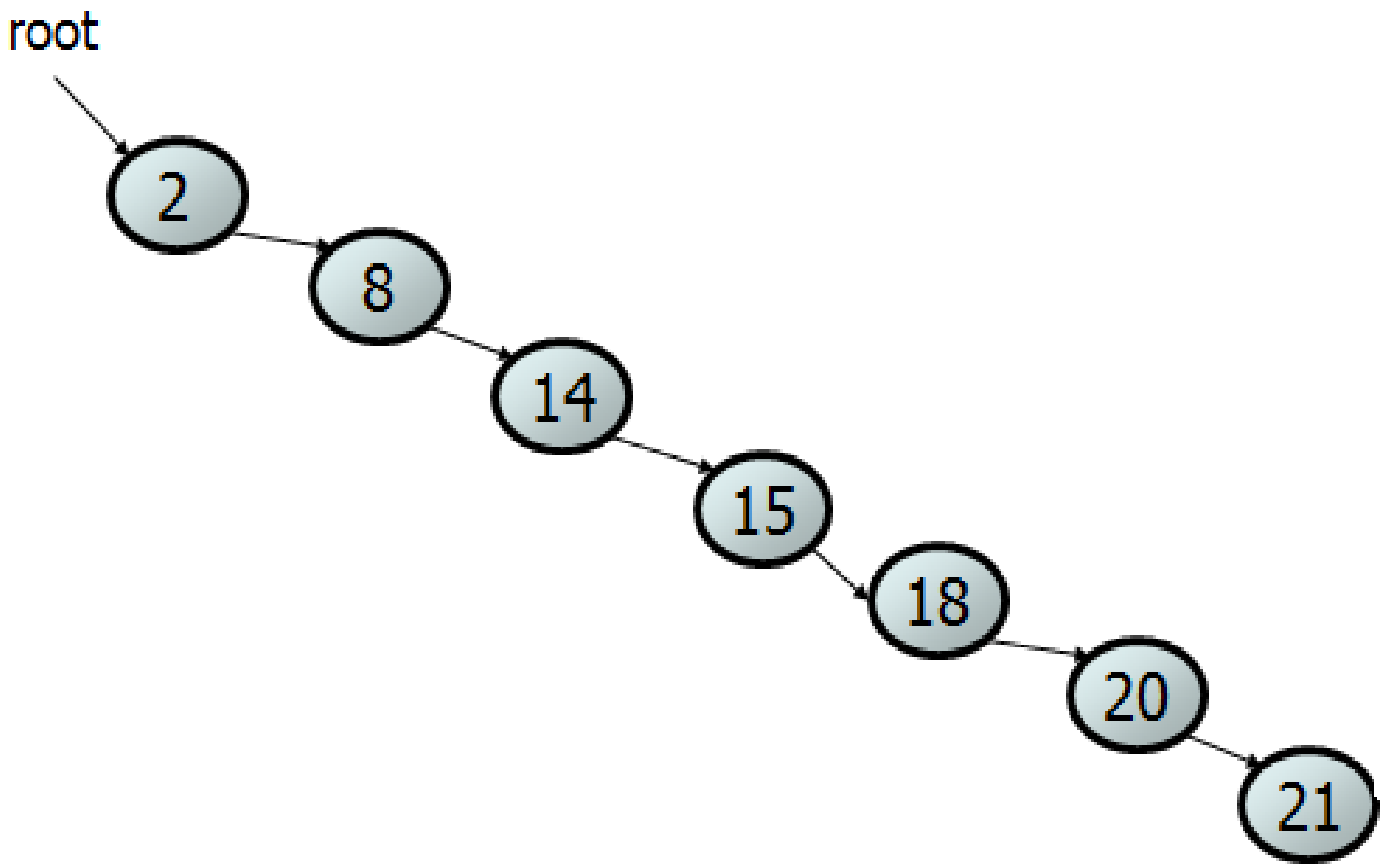
- Mostly **balanced**, height 4



Degenerate Tree

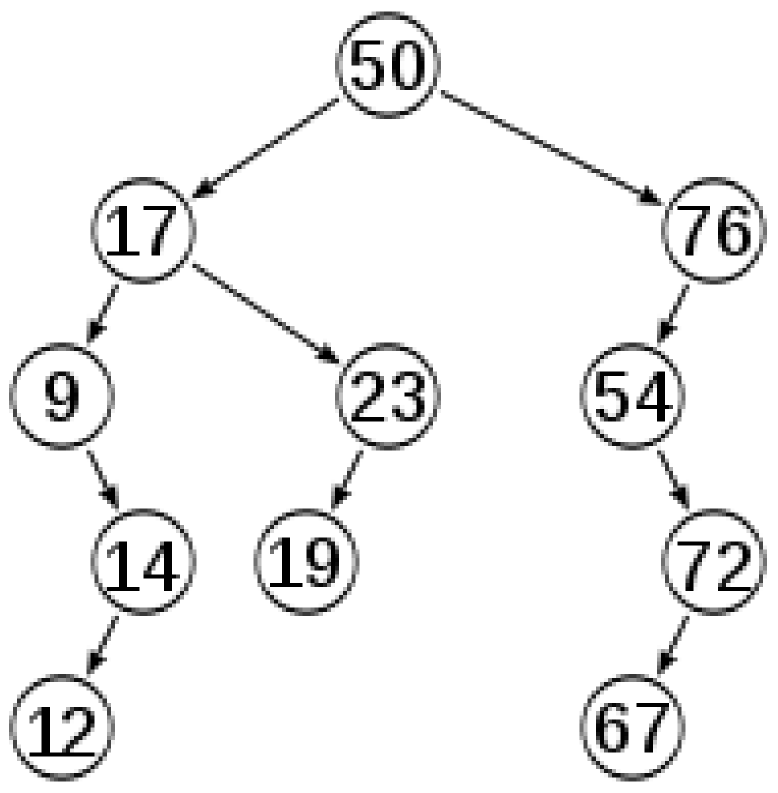
- Same Values: 2 8 14 15 18 20 21
  - Order added: 2, 8, 14, 15, 18, 20, 21

- Totally **unbalanced**, height 6

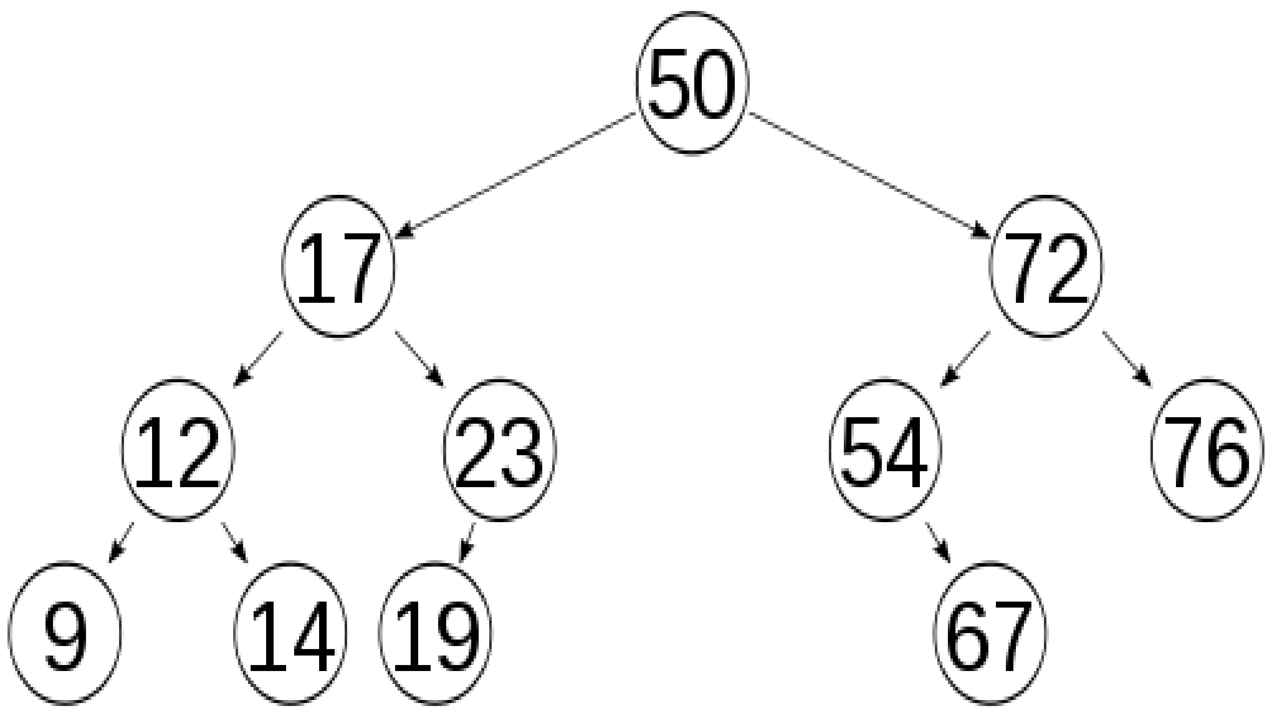


Balanced Tree

- **Balanced Tree:** a tree in which heights of sub-trees are approximately equal



unbalanced tree



balanced tree

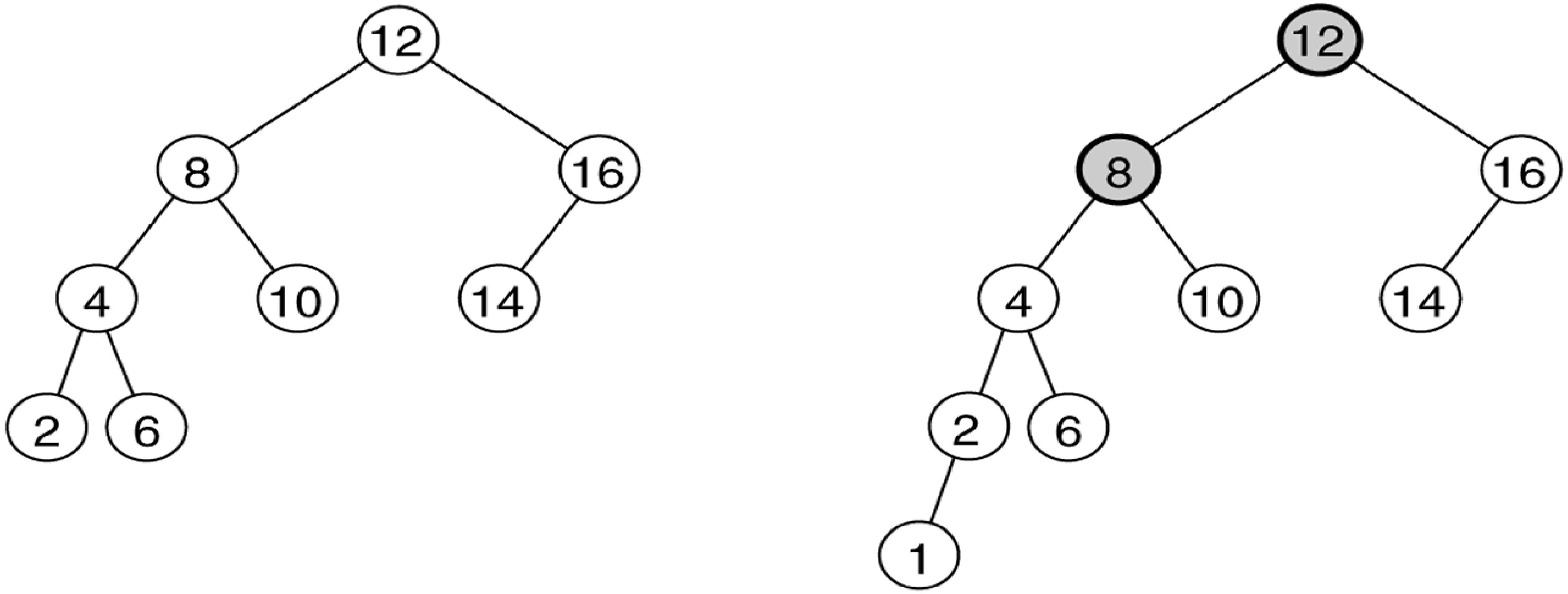
AVL Trees

- **AVL tree:** a binary search tree that uses modified add and remove operations to stay balanced as items are added to and remove from it
  - invented in 1962 by two mathematicians (Adelson-Velskii and Landis)
  - one of several auto-balancing trees (others in book)
  - specifically, maintains a balance factor of each node of **0**, **1**, or **-1**
    - i.e. no node's two child subtrees differ in height by more than **1**

- **balance factor**, for a tree node *n* :
  - height of *n*'s right subtree minus height of *n*'s left subtree
    - $BF_n = Height_{n.right} - Height_{n.left}$
  - start counting heights at n

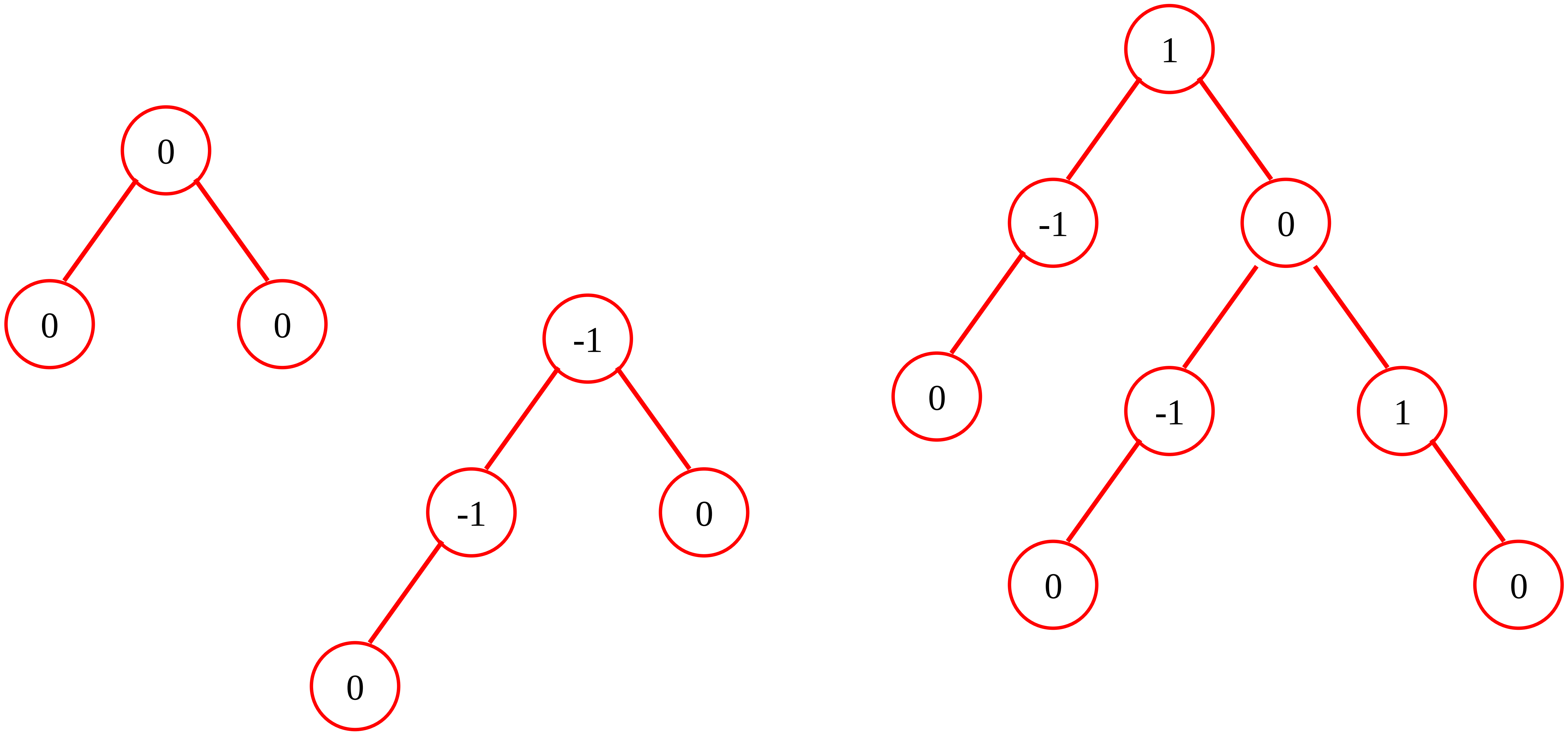
AVL tree examples

- Two binary search trees:
  - (a) an AVL tree
  - (b) not an AVL tree (unbalanced nodes are darkened)

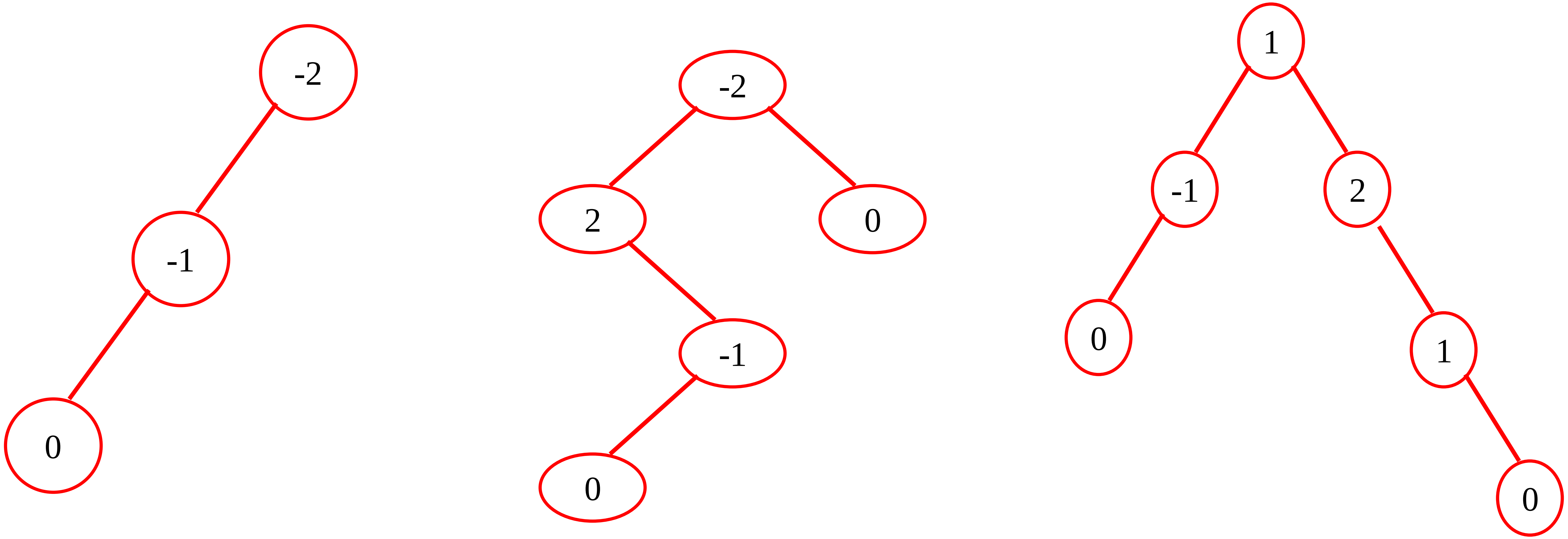


(a) (b)

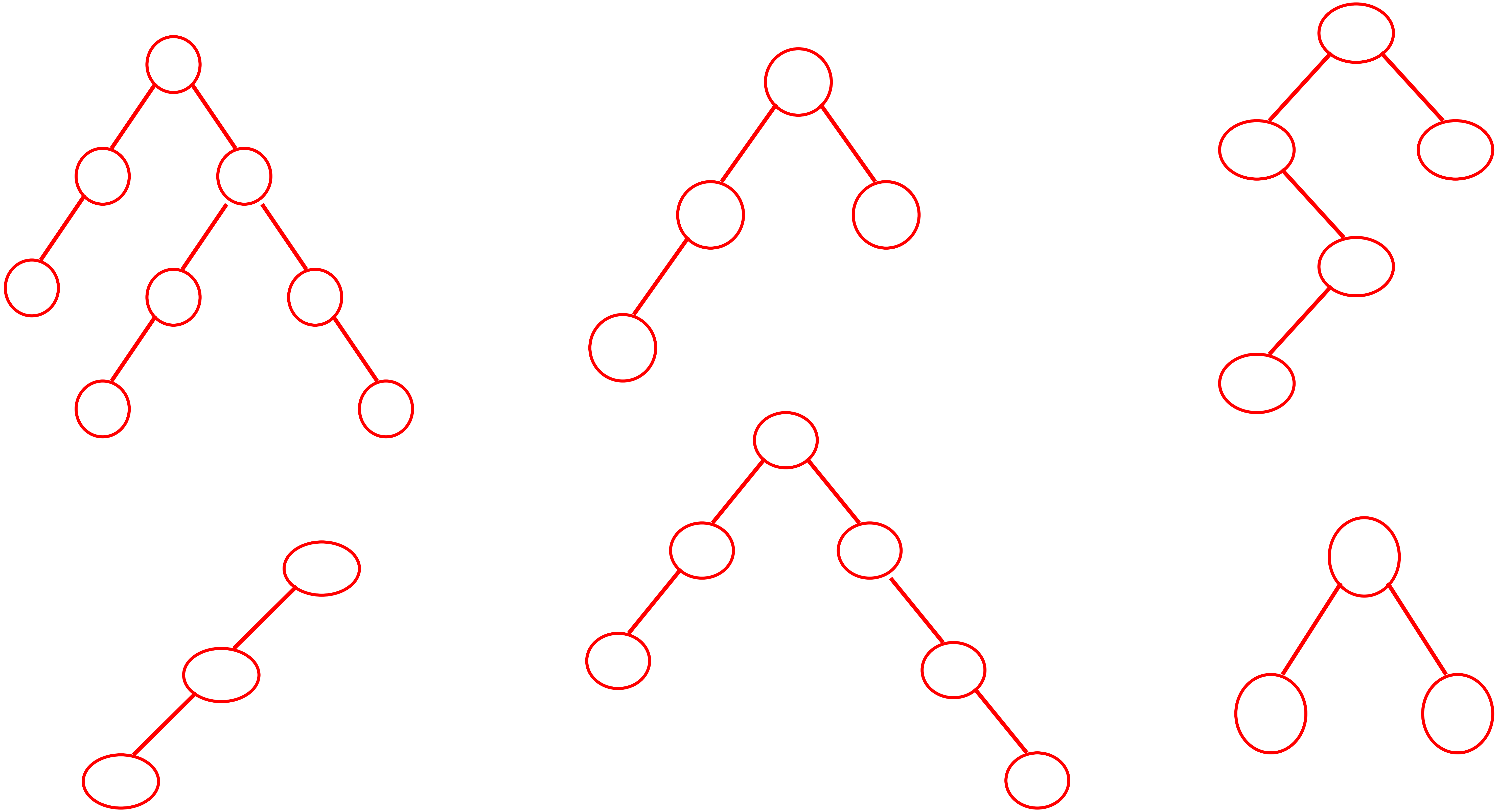
More AVL tree examples



Not AVL tree examples



Which are AVL trees?



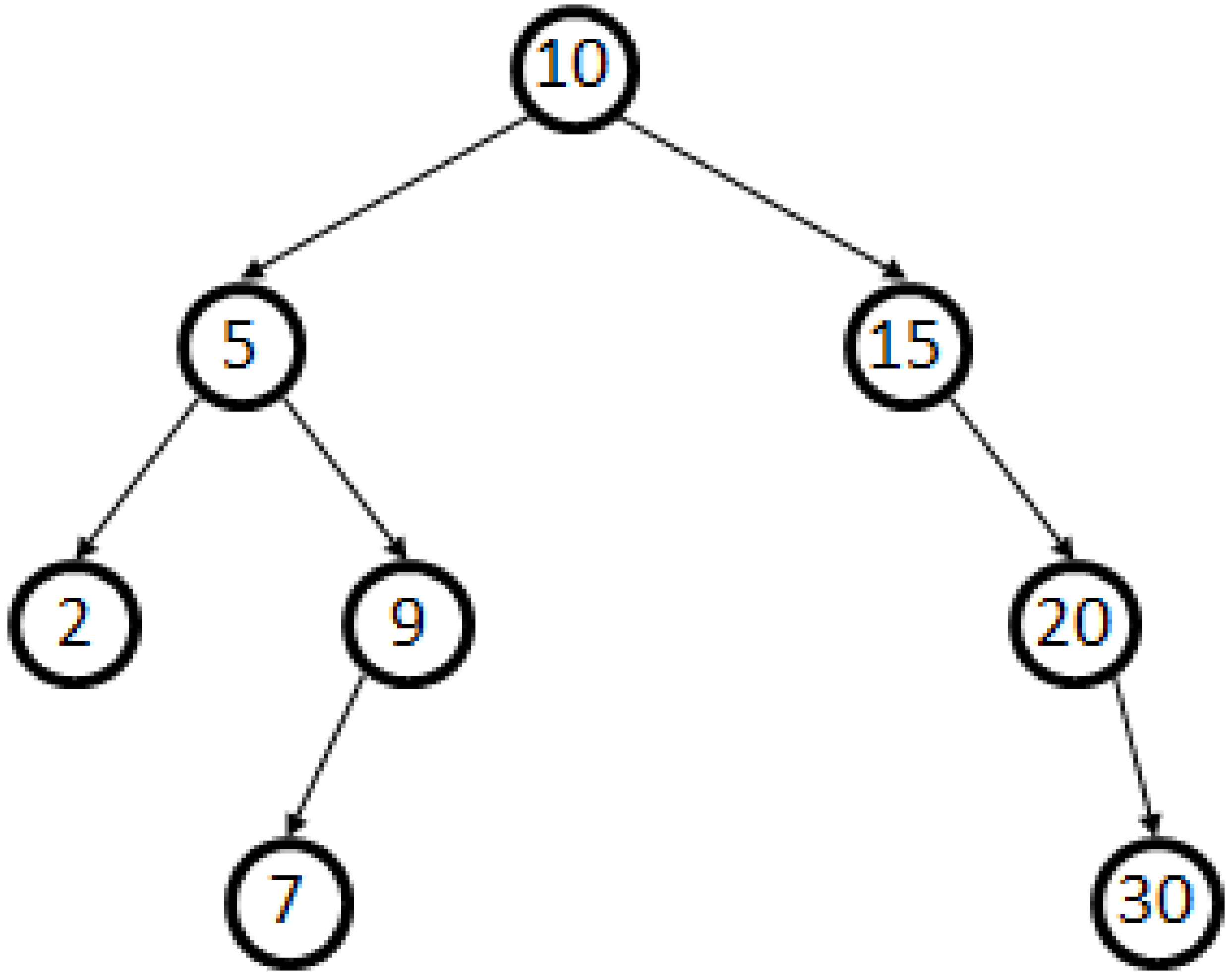
AVL Trees: search, insert, remove

- **AVL search:**
  - Same as BST search.

- **AVL insert:**
  - Same as BST insert, except you need to check your balance and may need to "fix" the AVL tree after the insert.

- **AVL remove:**
  - Remove it, check your balance, and fix it.

Testing the Balance Property

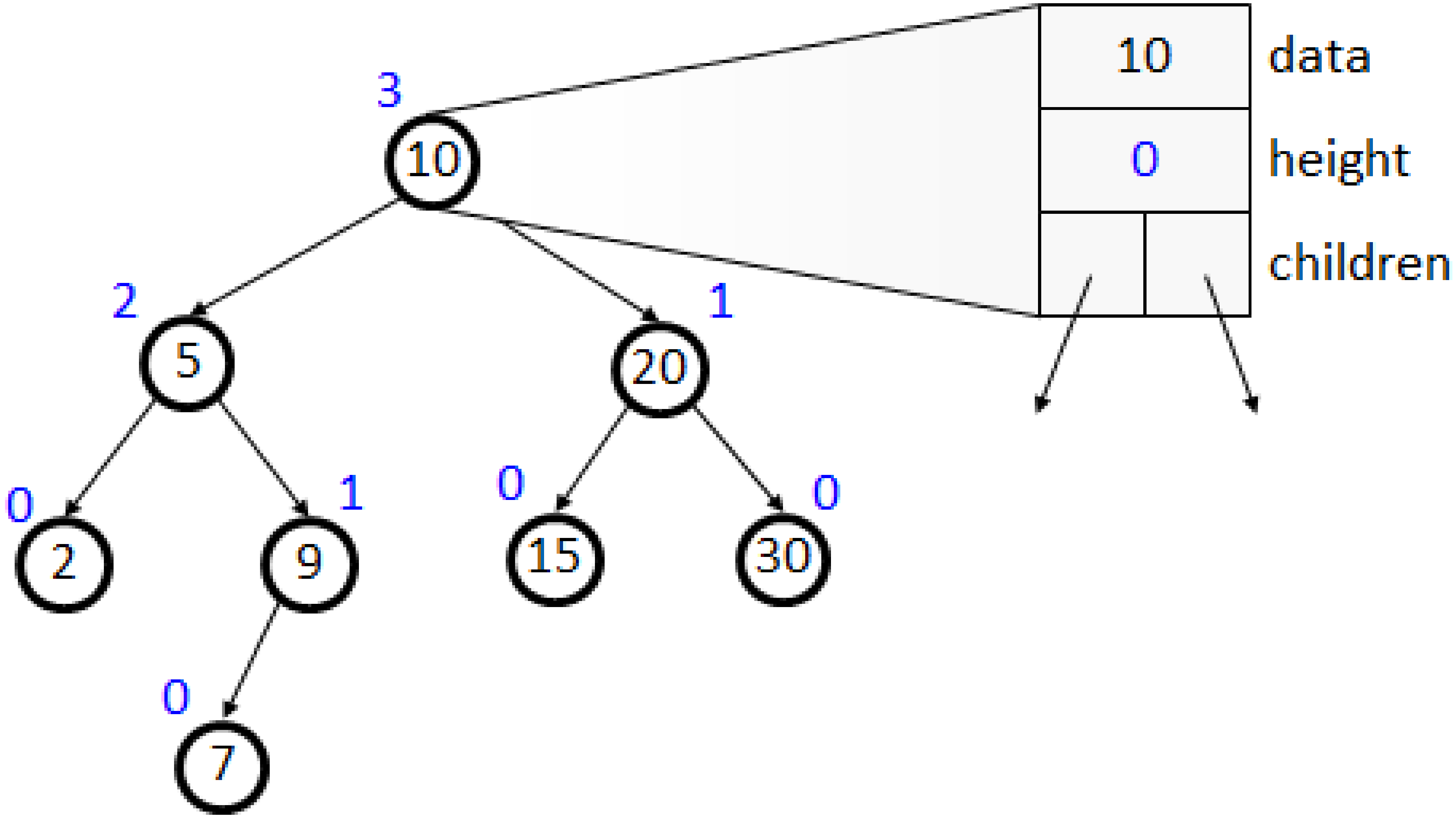


- We need to be able to:

  1. Track Balance Factor
  2. Detect Imbalance
  3. Restore Balance

- How do we accomplish each step?

Tracking Balance



Acknowledgement

- Mostly Slides taken from Book: **“Data Structures through C++”** by Yashavant P. Kanetkar