CSE 373: Data Structures and Algorithms

Binary Search and BSTs

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Administrivia

Due dates:

- HW2 Part 2 due Friday 11:59pm
- HW1 grades will be released later today

Modeling recursion: Unfolding Method

$$T(n) = \begin{cases} C_1 & \text{when} \quad n = 0, 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

$$T(n) = C_2 + T(n-1)$$

$$T(n) = C_2 + C_2 + T(n-2)$$

$$T(n) = C_2 + C_2 + C_2 + T(n-3)$$

$$T(n) = C_2 + C_2 + C_2 + C_2 + \cdots + C_2 + T(2)$$

$$T(n) = C_2 + C_2 + C_2 + C_2 + \cdots + C_2 + C_2 + T(1)$$

$$T(n) = C_2 + C_2 + C_2 + C_2 + \cdots + C_2 + C_2 + C_1$$

$$T(n) = \sum_{i=0}^{n-2} C_2 + C_1 \qquad T(n) = (n-1)C_2 + C_1$$

Modeling binary search recursion

Question: Find the closed form for T(N)

$$T(n) = \begin{cases} C_1 & \text{when} \quad n = 1 \\ C_2 + T\left(\frac{n}{2}\right) & \text{when} \quad n > 1 \end{cases}$$

$$T(n) = C_2 + T\left(\frac{n}{2}\right)$$

$$T(n) = C_2 + \left(C_2 + T\left(\frac{n}{4}\right)\right)$$

$$T(n) = C_2 + C_2 + \left(C_2 + T\left(\frac{n}{8}\right)\right)$$

$$T(n) = \underbrace{C_2 + C_2 + \cdots + C_2}_{\text{t times}} + T\left(\frac{n}{2^t}\right)$$

We want to find a t such that $\frac{n}{2^t} = 1$

Solving for t we get $t = log_2 n$

$$T(n) = C_2 log_2 n + C_1$$

Storing Sorted Items in an Array

get() – O(logn)

put() – O(n)

remove() – O(n)

Can we do better with insertions and removals?

Trees!

A **tree** is a collection of nodes
- Each node has at most 1 parent and 0 or more children

Root node: the single node with no parent, “top” of the tree

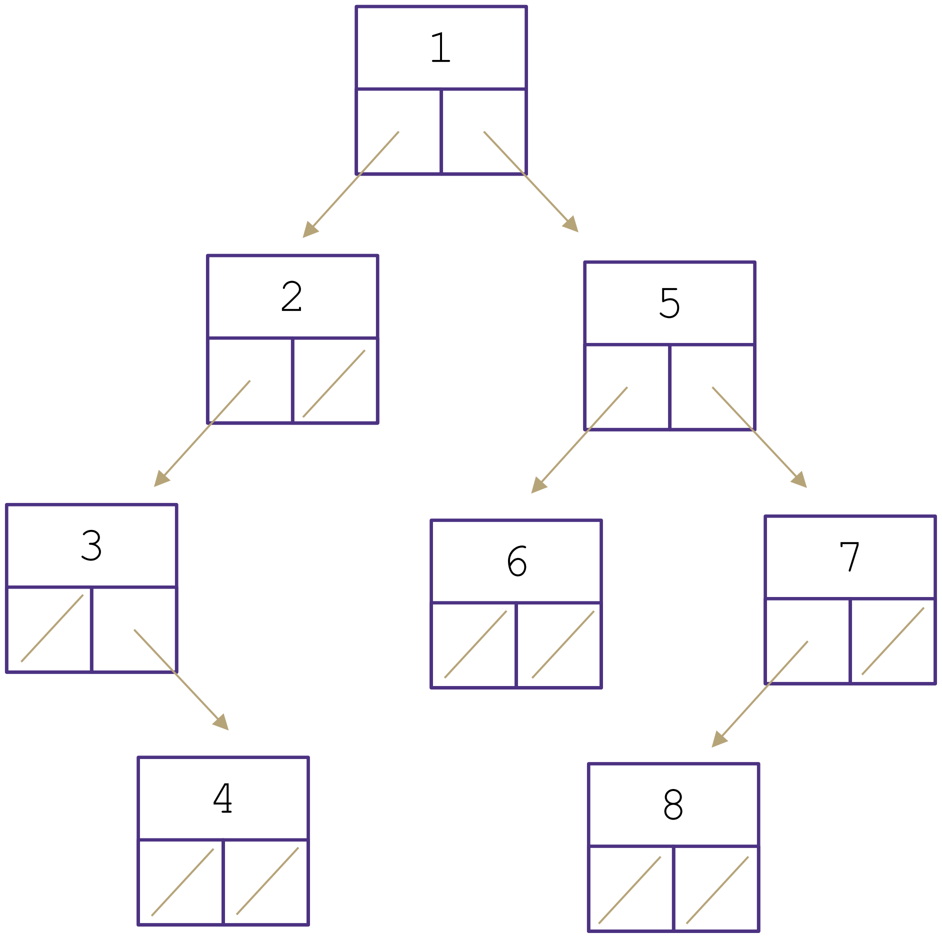
Branch node: a node with one or more children

Leaf node: a node with no children

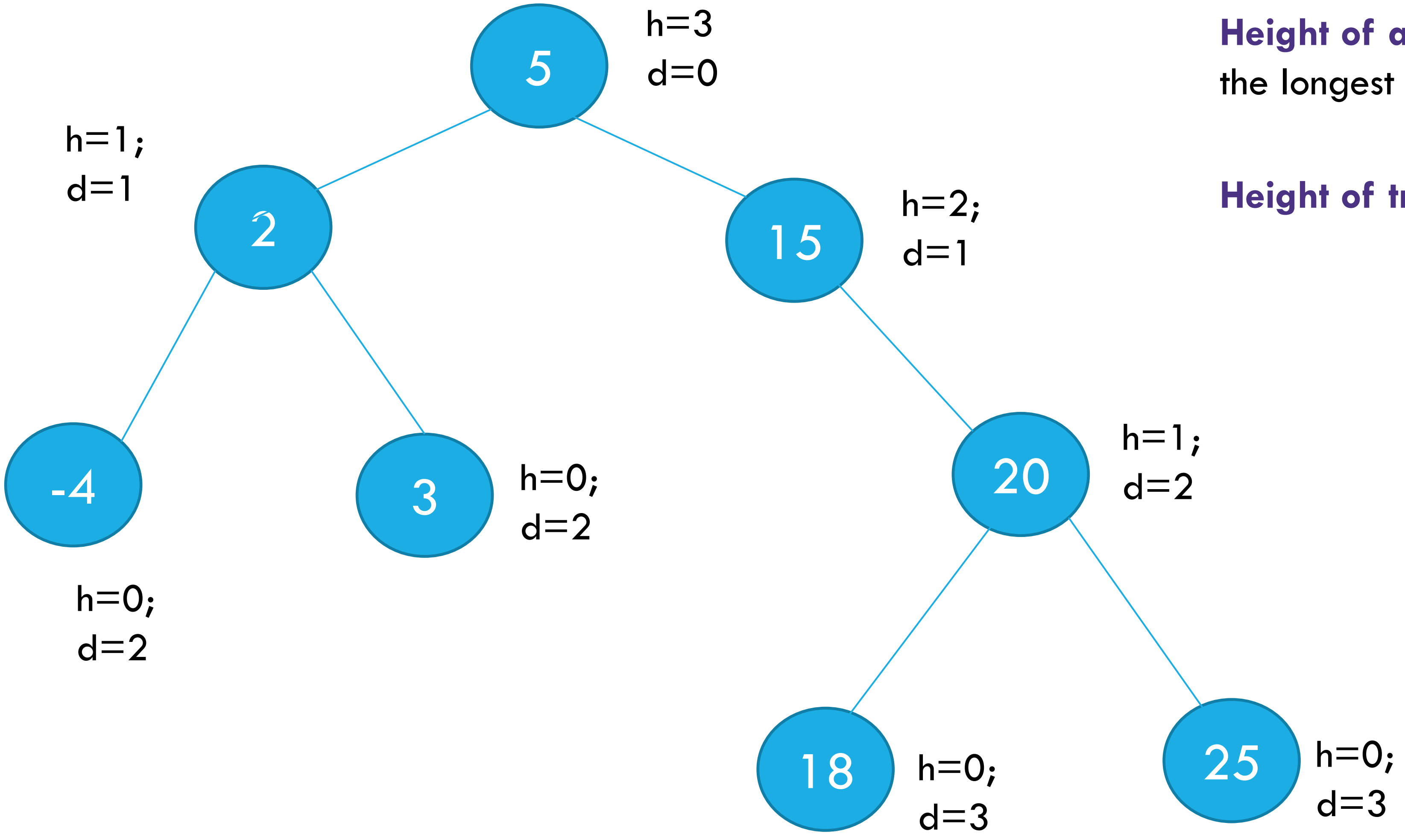
Edge: a pointer from one node to another

Subtree: a node and all it descendants

Height: the number of edges contained in the longest path from root node to some leaf node



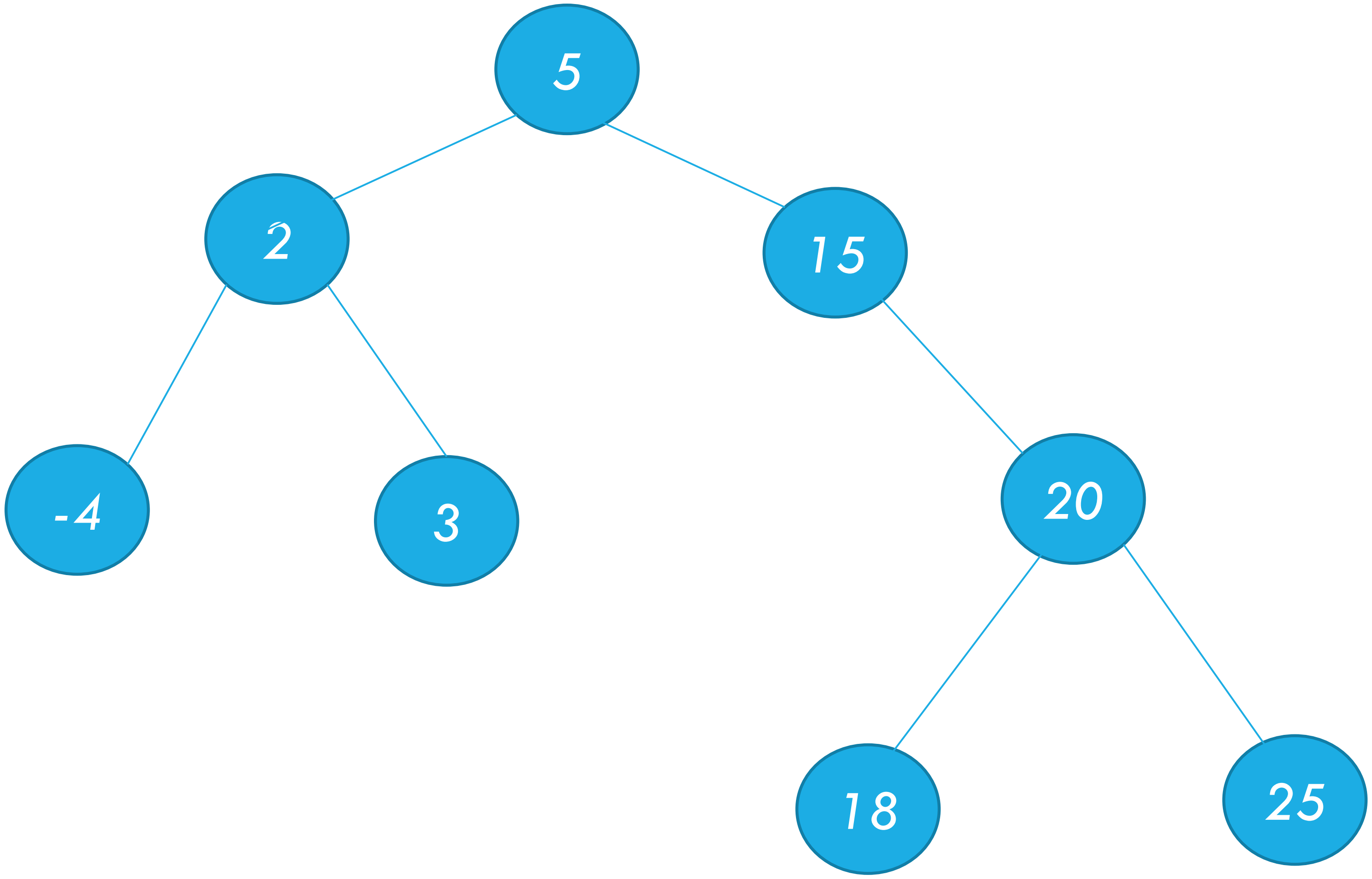
Binary Tree – Height and Depth



Height of a node: the number of edges contained in the longest path from the node to some leaf node

Height of tree = height of root node

Binary Search Tree – O(h) search



Unbalanced Trees

Is this a valid Binary Search Tree?

Yes, but…

We call this a degenerate tree

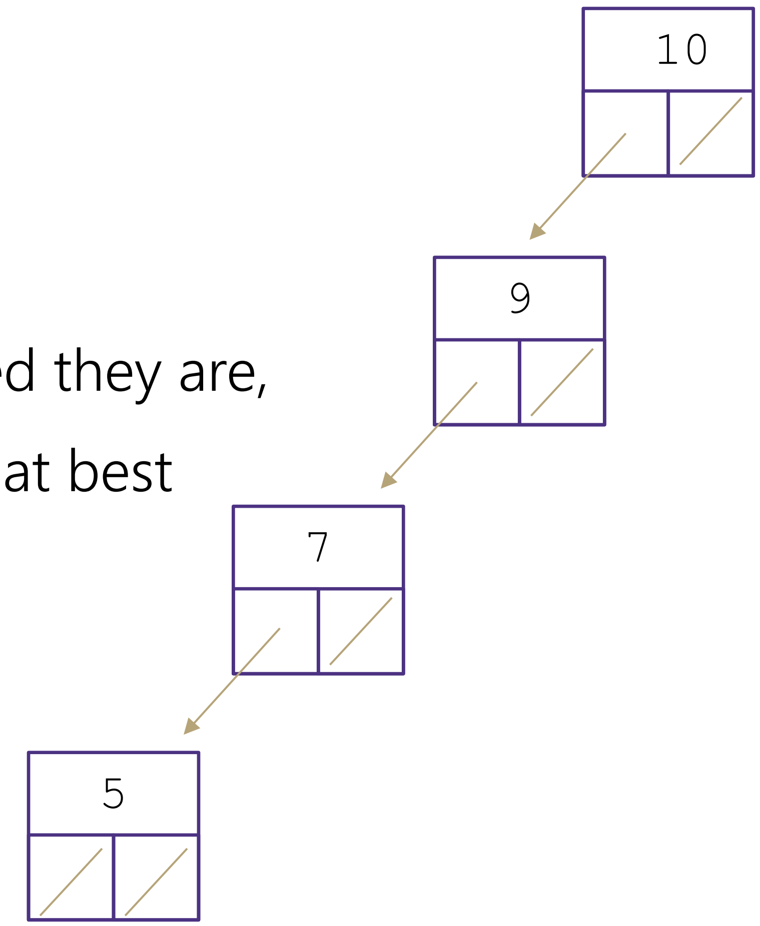
For trees, depending on how balanced they are,

Operations at worst can be O(n) and at best

can be O(logn)

How are degenerate trees formed?

- insert(10)
- insert(9)
- insert(7)
- insert(5)



Unbalanced Trees

Is this a valid Binary Search Tree?

Yes, but…

We call this a **degenerate tree**

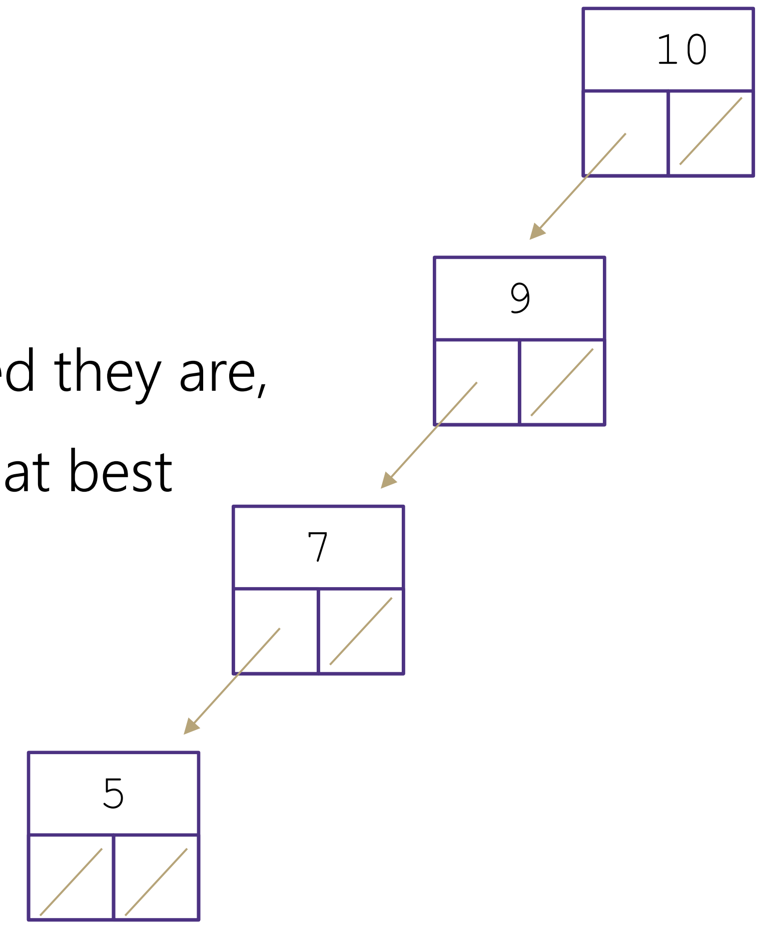
For trees, depending on how balanced they are,

Operations at worst can be O(n) and at best

can be O(logn)

How are degenerate trees formed?

- insert(10)
- insert(9)
- insert(7)
- insert(5)

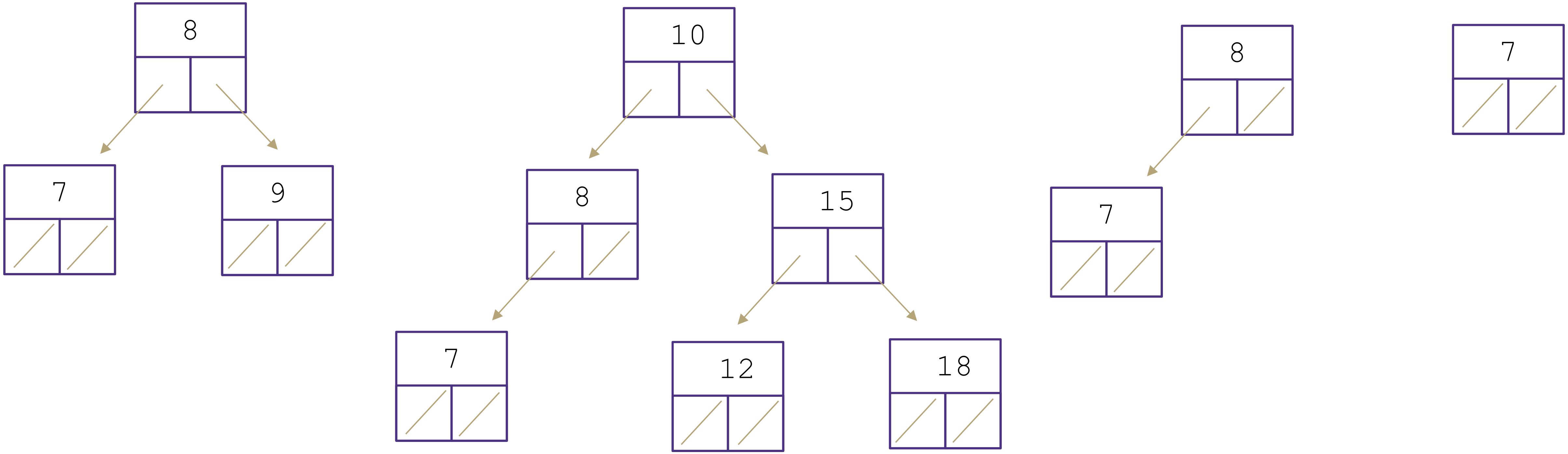


Measuring Balance

Measuring balance:

For each node, compare the heights of its two sub trees

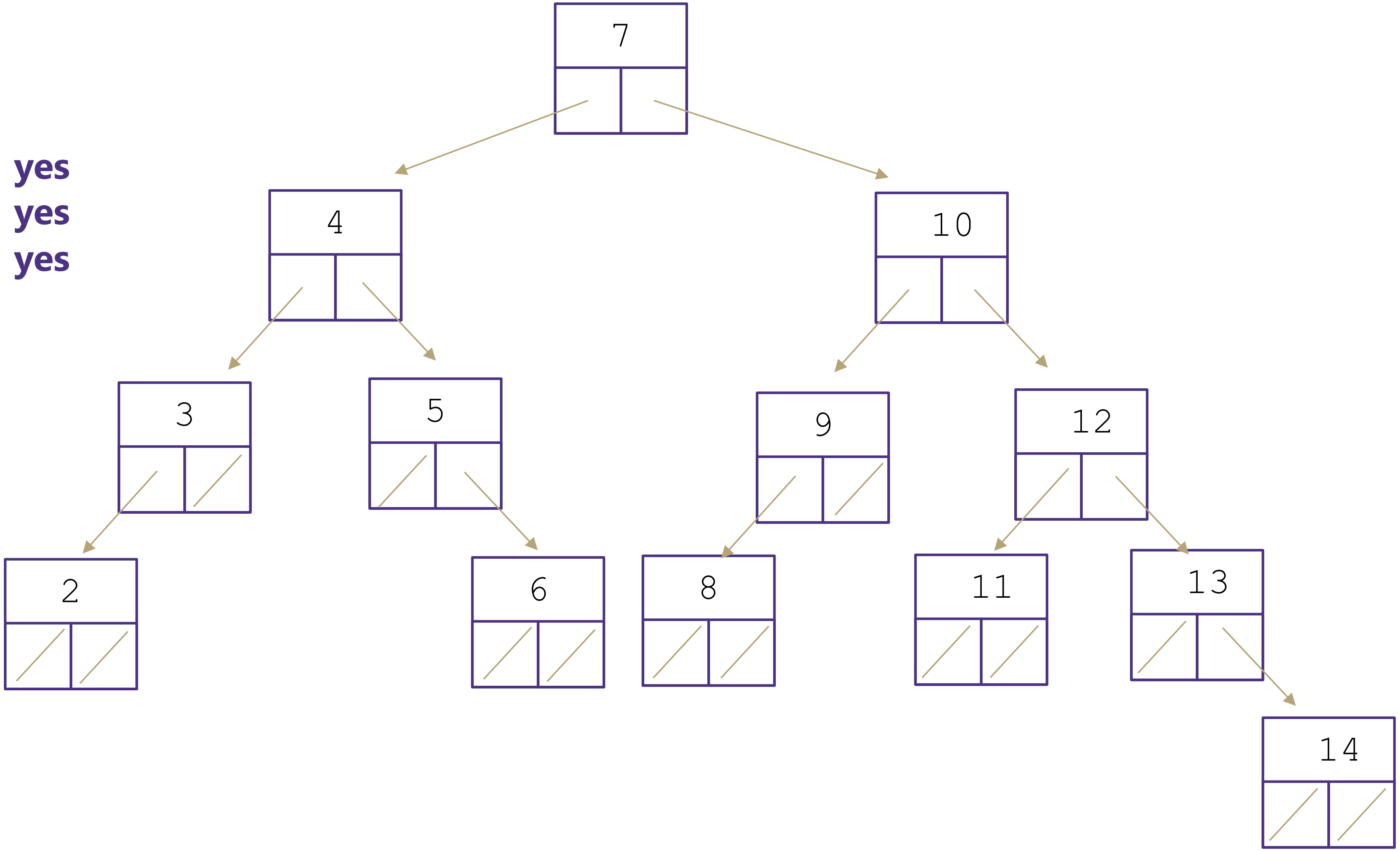
Balanced when the difference in height between sub trees is no greater than 1



Is this a valid AVL tree?

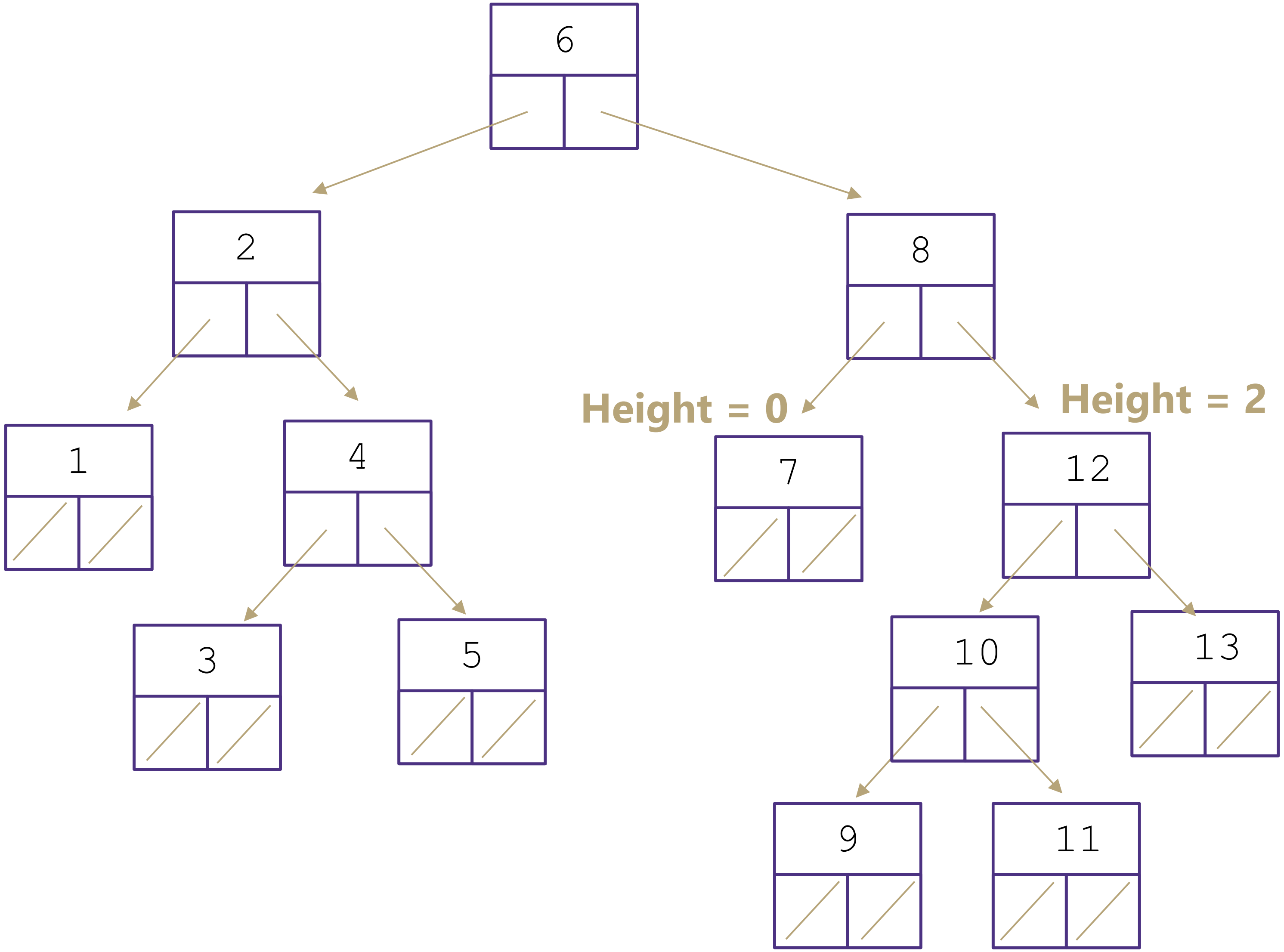
Is it…

- Binary **yes**
- BST **yes**
- Balanced? **yes**



Is this a valid AVL tree?

Is it…
- Binary **yes**
- BST **yes**
- Balanced? **no**



Is this a valid AVL tree?

Is it…
- Binary **yes**
- BST **no**
- Balanced? **yes**