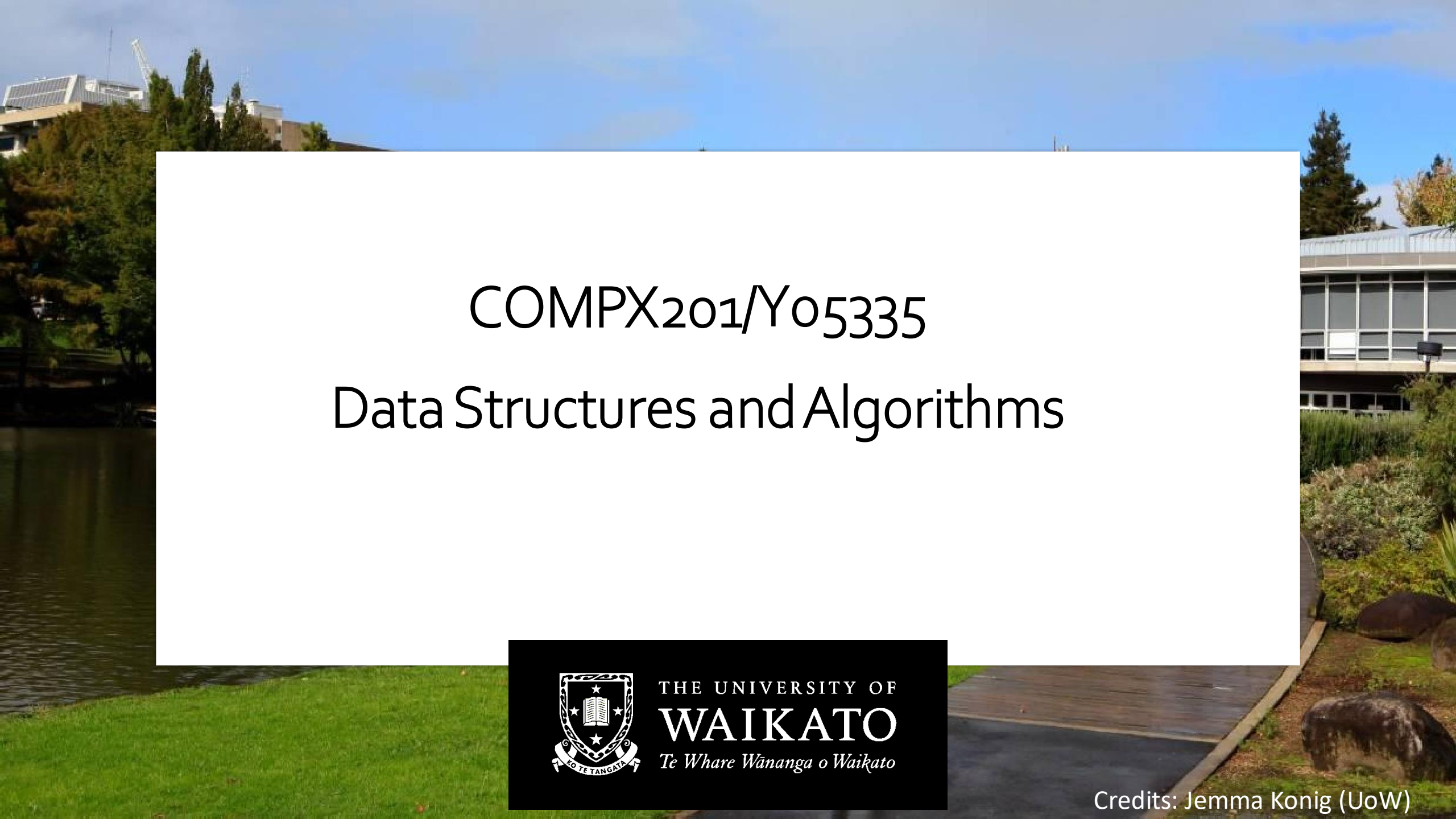


COMPX201/Y05335

Data Structures and Algorithms

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Credits: Jemma Konig (UoW)

Binary Search Trees
Definition and Structure

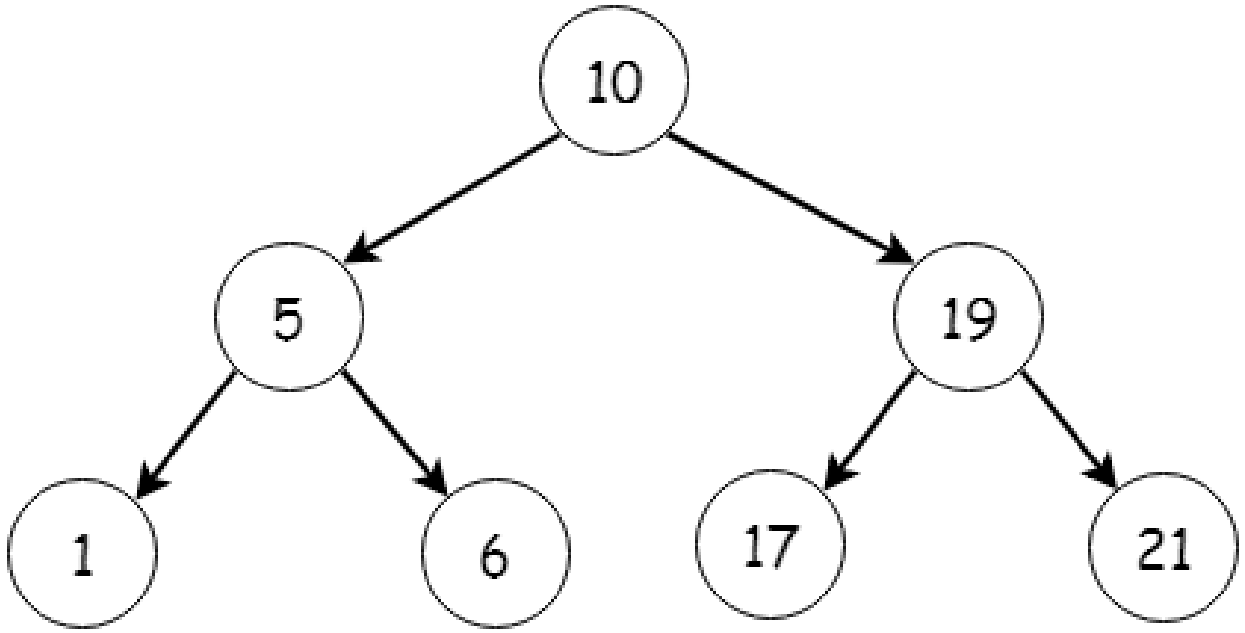
COMPX201/Y05335

Overview

- What is a Binary Search Tree?
- BST Structure

What is a Binary Search Tree (BST)?

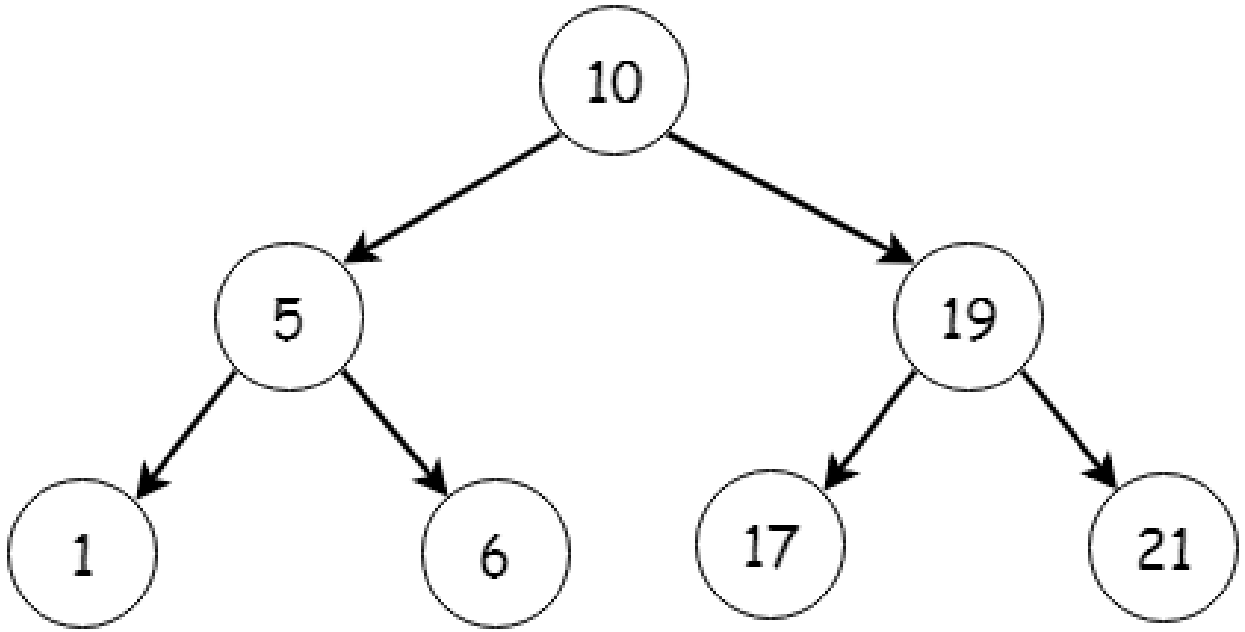
- Abstract data type that supports logarithmic search times.
- Organisation of data generally “halves” search space.



What is a Binary Search Tree?

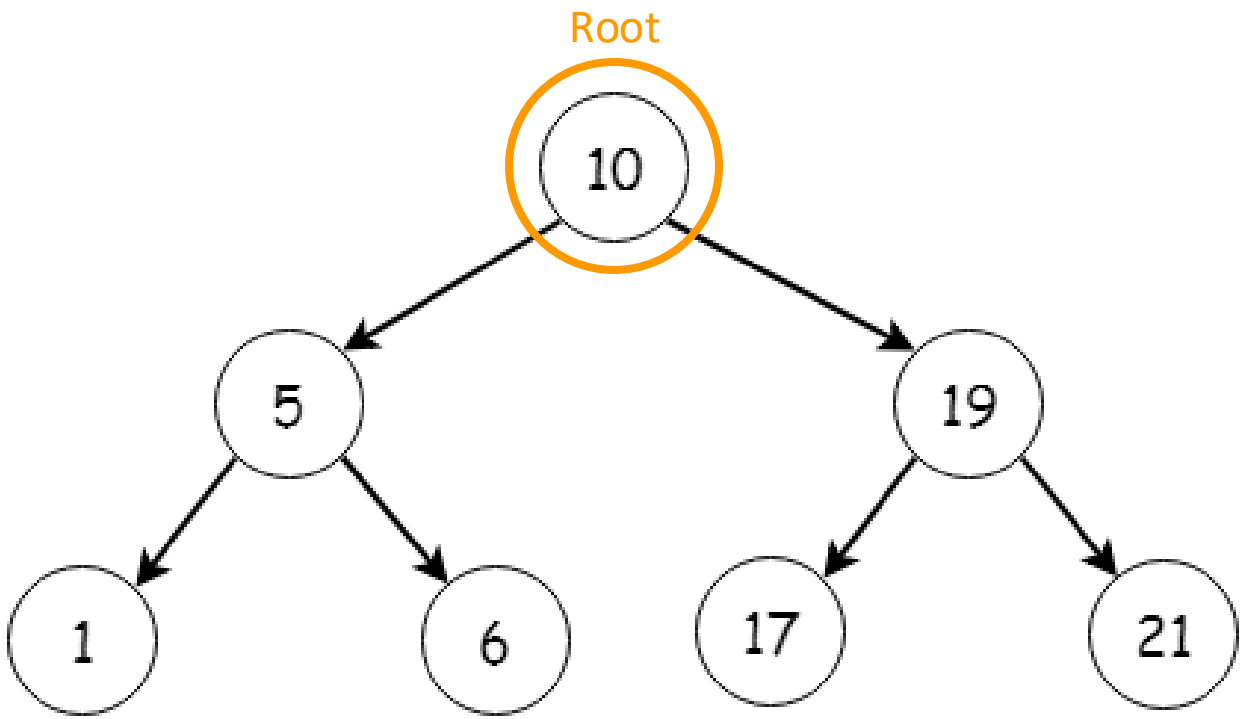
- The top node is the 'root'
- Each node has a value and two subtrees (left and right).
- Each subtree is another BST.
- Left subtree has all values less than those at the root.
- Right subtree has all values greater than those as the root.
- What about equal to?
- The root node is the "parent" of the two subtrees. The two nodes at the root of the two subtrees are called "children".
- A node with no children is called a "leaf" node.
- All nodes in the subtrees are "descendants" of a particular root node.
- All nodes along the search path between the root and a given node are "ancestors".

BST structure



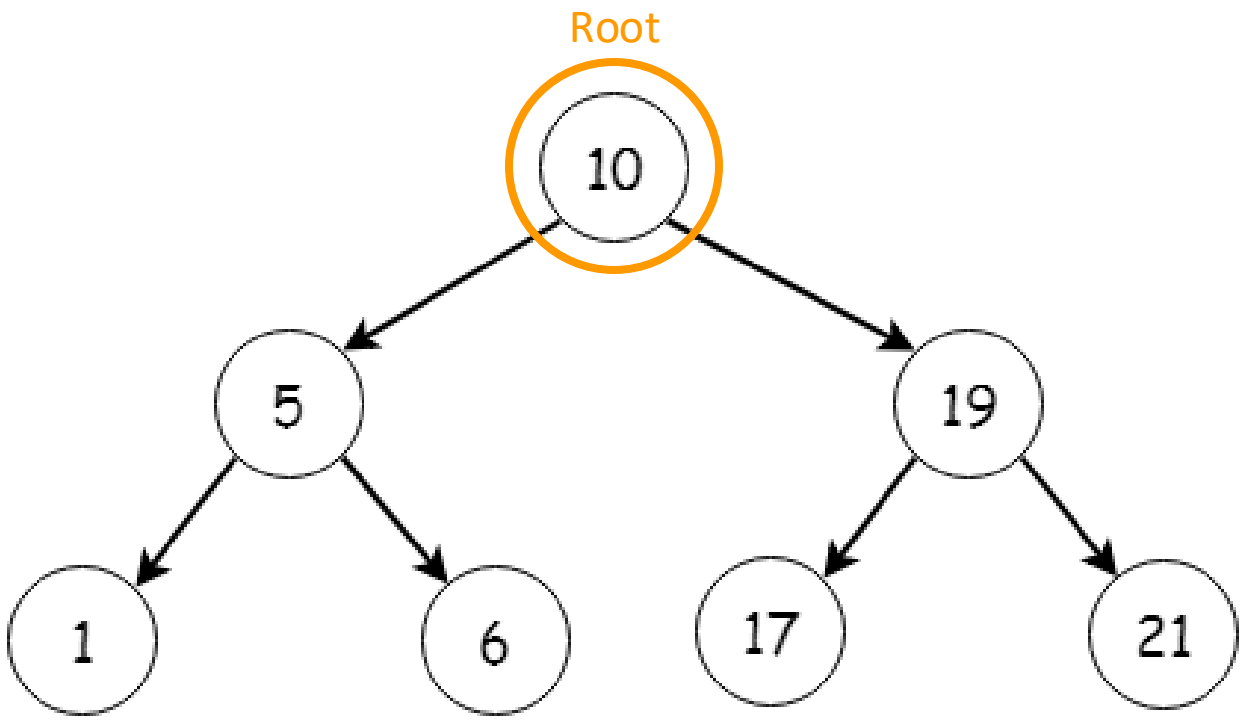
BST structure

- The top node is the ‘root’



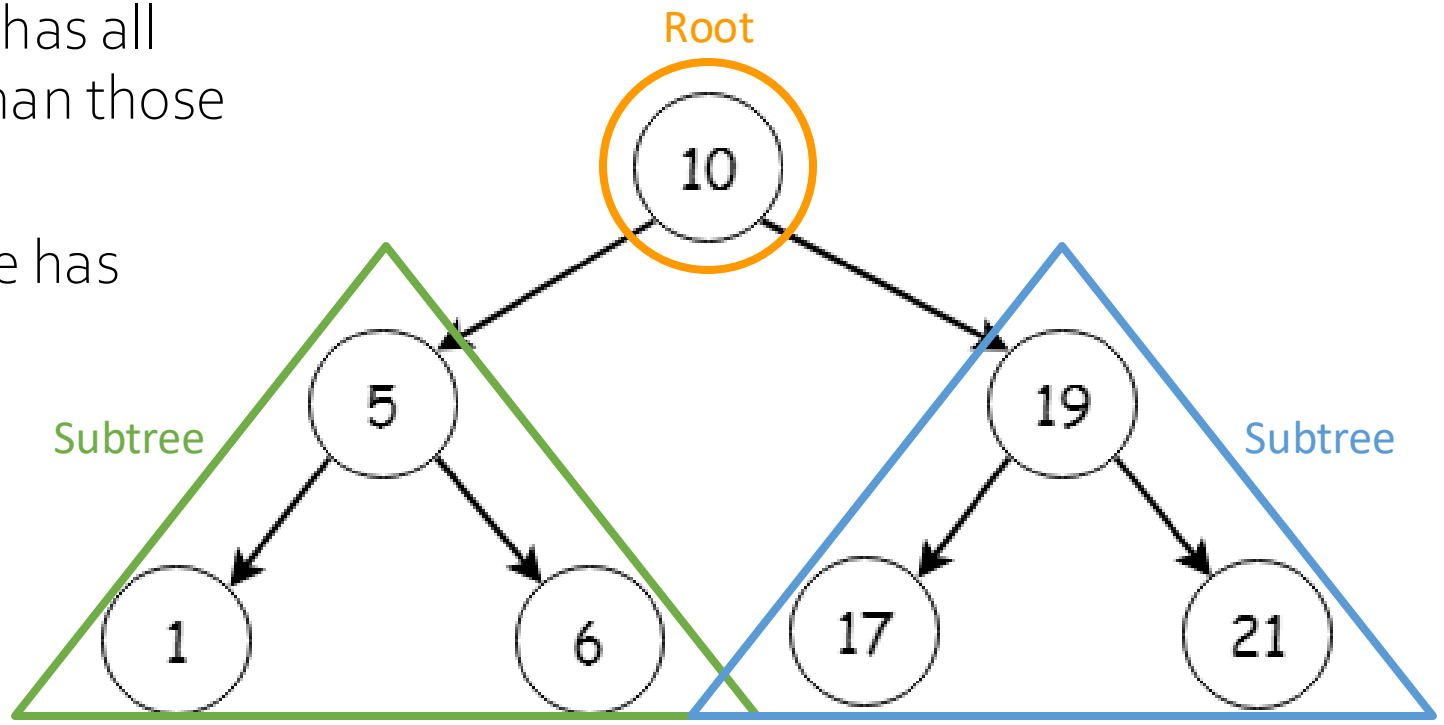
BST structure

- Each node has a value and two subtrees (left and right).



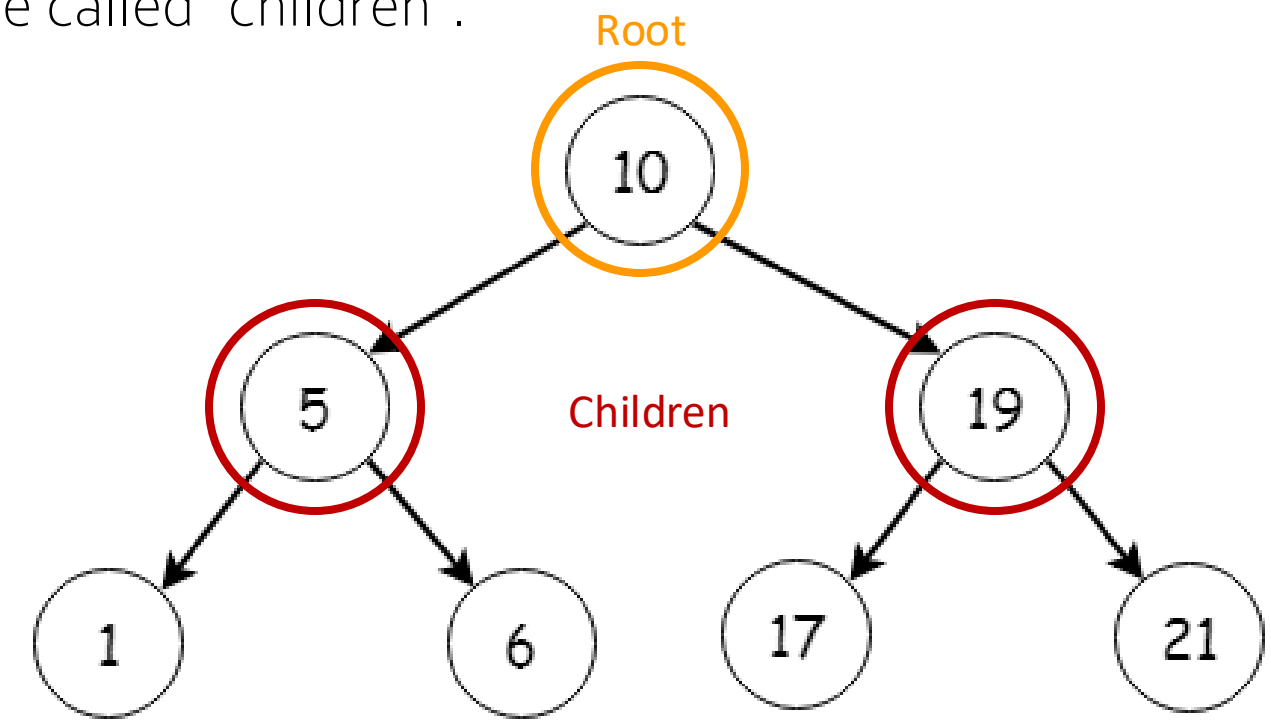
BST structure

- Each subtree is another BST.
- Left subtree has all values less than those at the root.
- Right subtree has all values greater than those as the root.



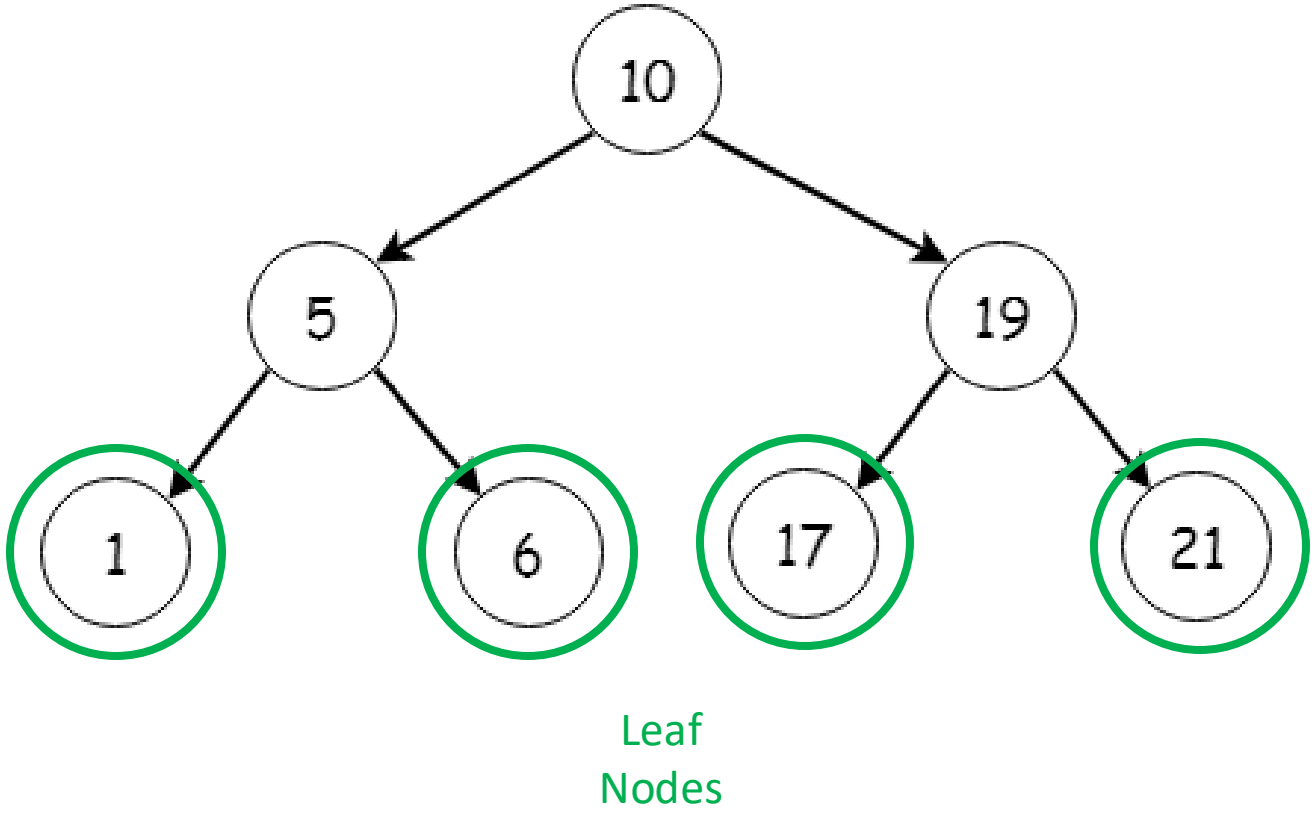
BST structure

- The root node is the “parent” of the two subtrees. The two nodes at the root of the two subtrees are called “children”.



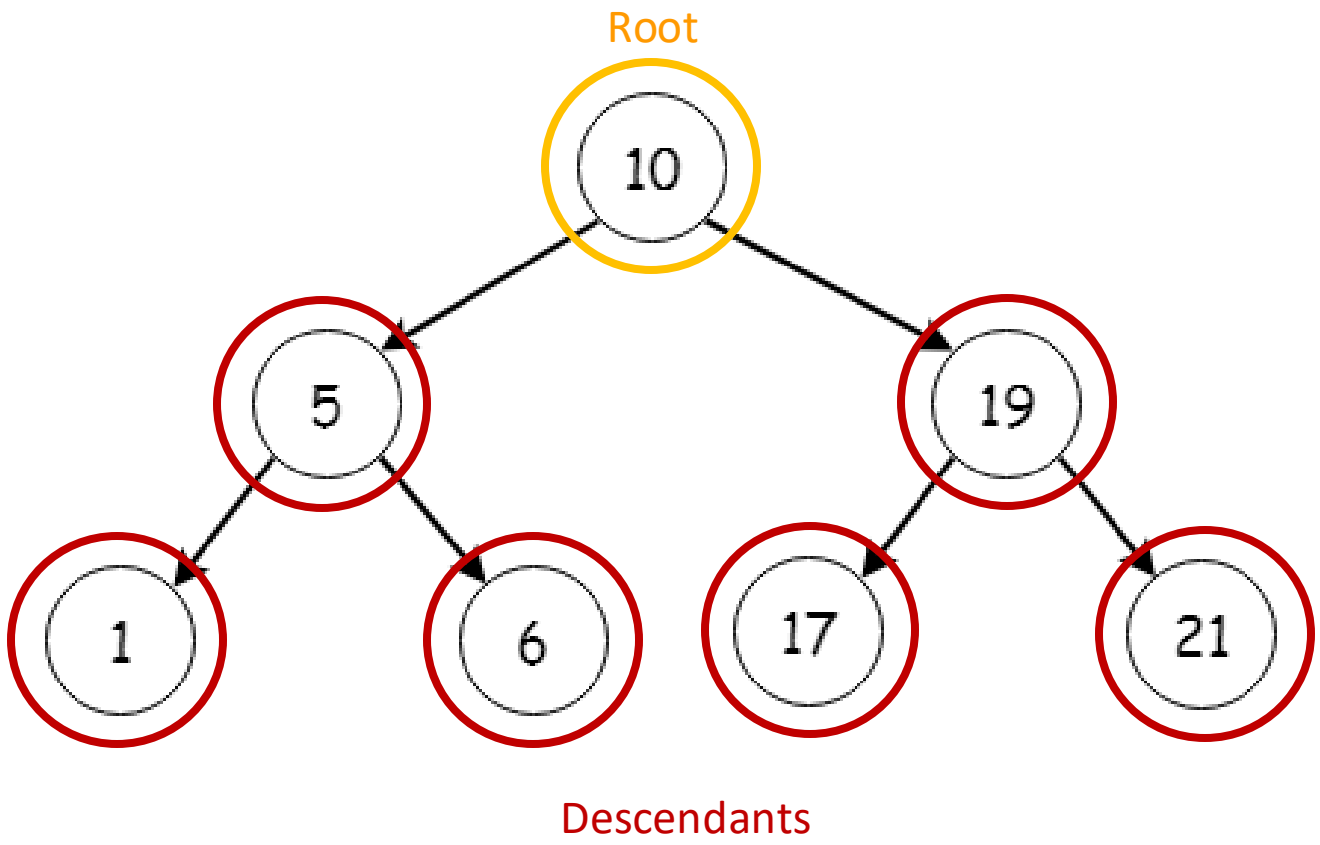
BST structure

- A node with no children is called a “leaf” node.



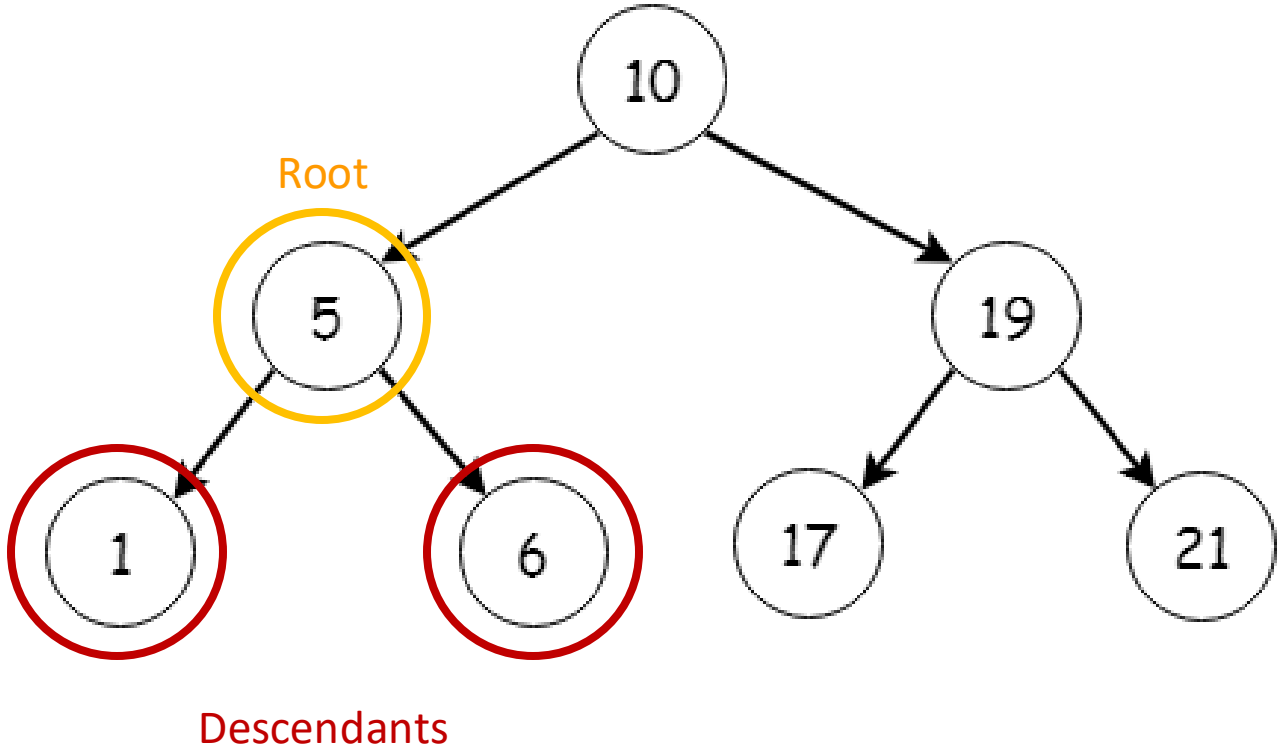
BST structure

- All nodes in the subtrees are “descendants” of a particular root node.



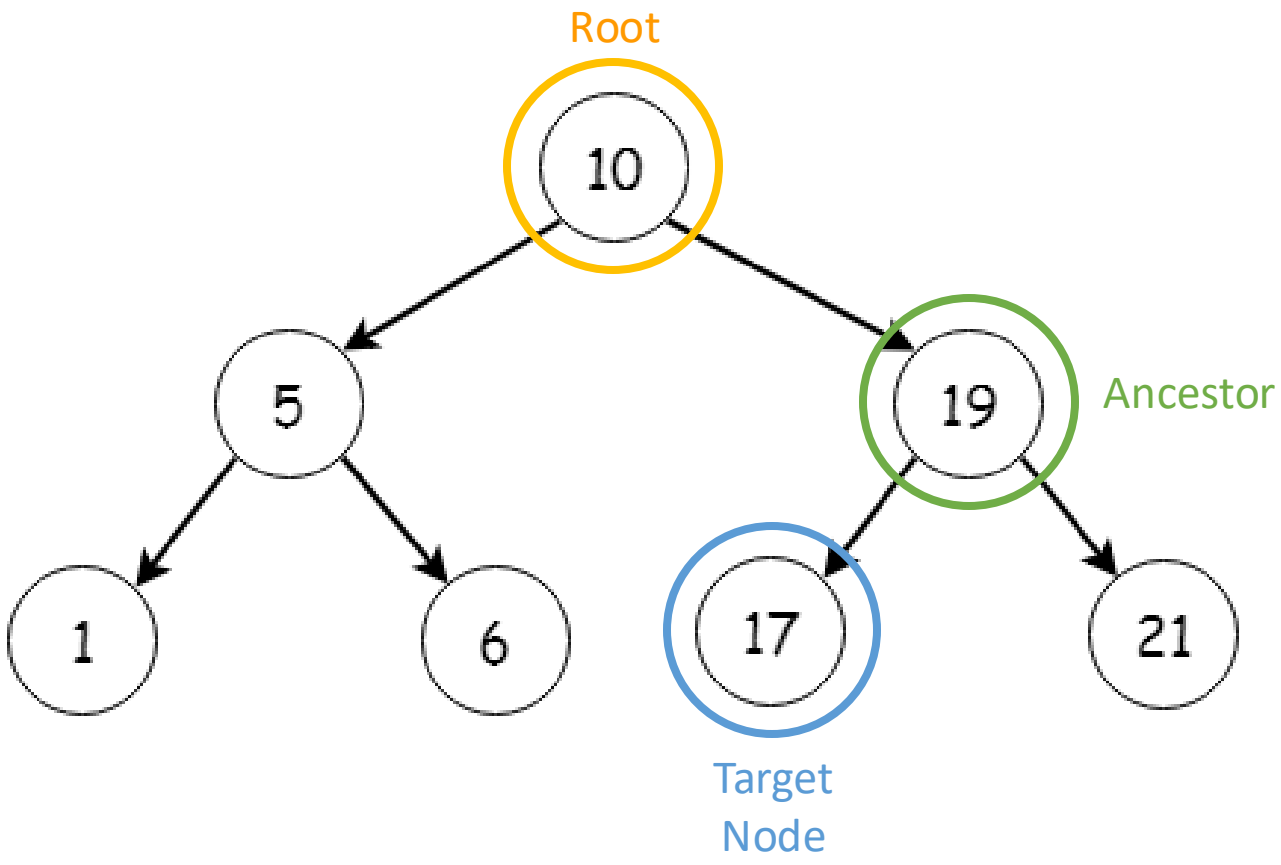
BST structure

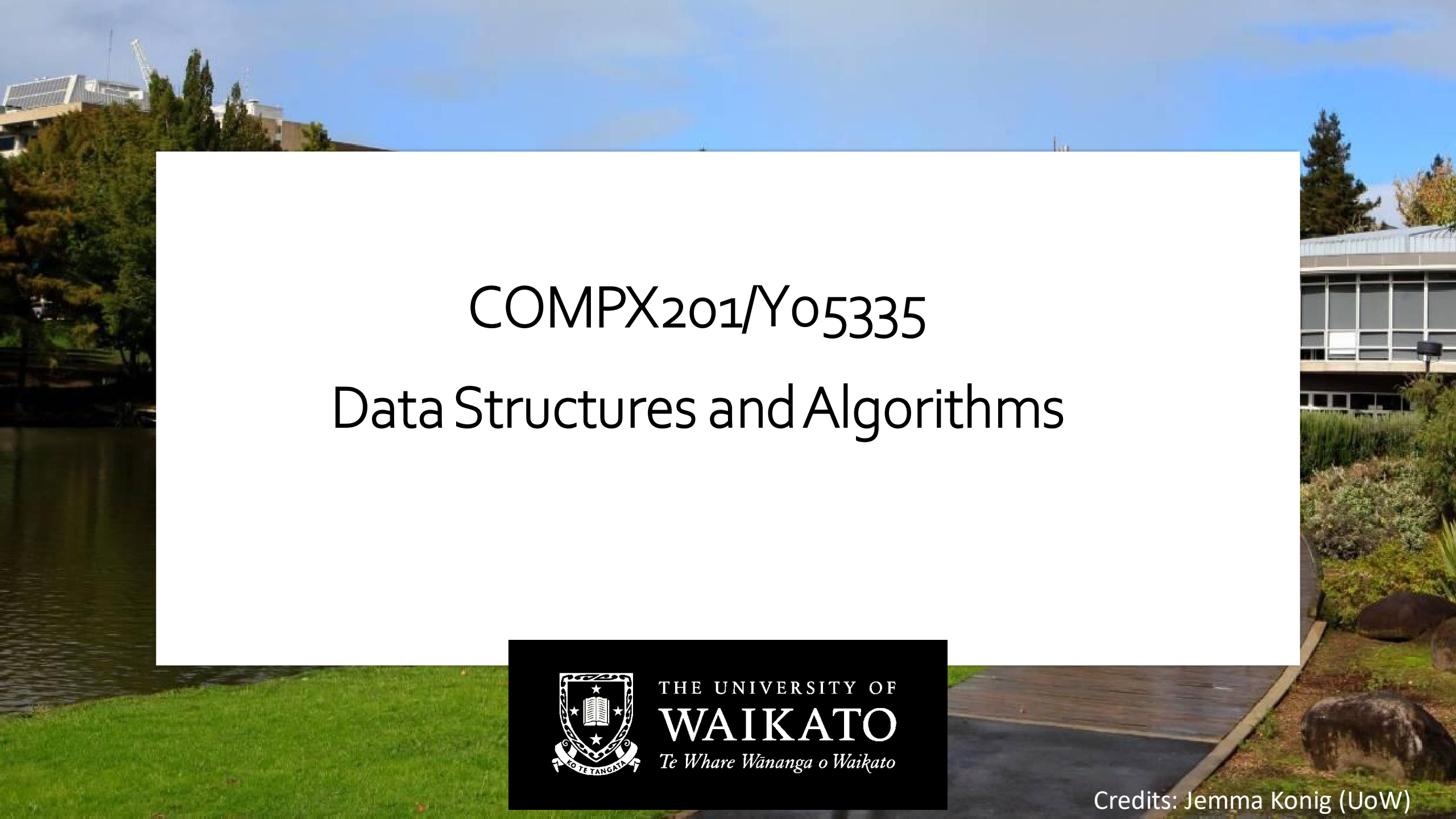
- All nodes in the subtrees are “descendants” of a particular root node.



BST structure

- All nodes along the search path between the root and a given node are “ancestors”.





COMPX201/Y05335

Data Structures and Algorithms

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Credits: Jemma Konig (UoW)

Binary Search Trees
Operations

COMPX201/Y05335

Overview

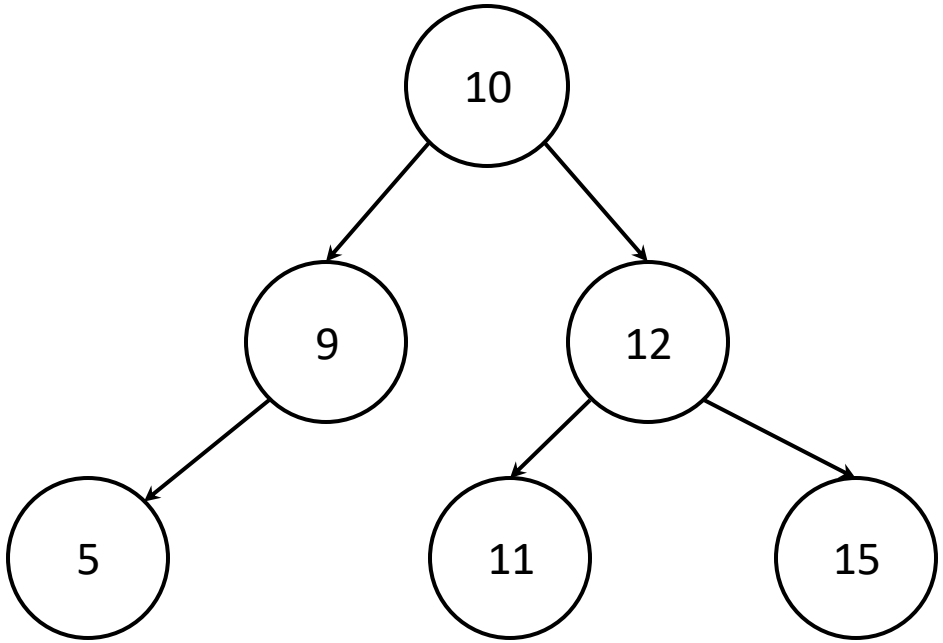
Operations

- Searching
- Insertion
- Deletion

BST searching

- Start at the top of the tree (the “root” node, like the “head” of a list).
- If root node is empty, the tree is empty
- If the value matches our target value, we have the correct node
- If our value is smaller repeat the process in the left subtree
- If our value is greater repeat the process in the right subtree
- Makes it easy to use recursion!

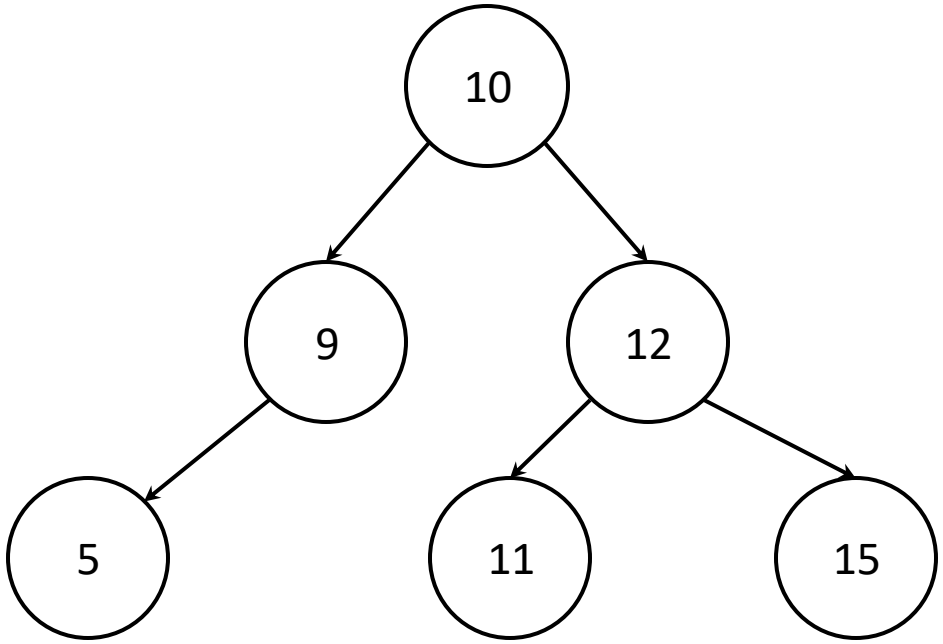
BST searching



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

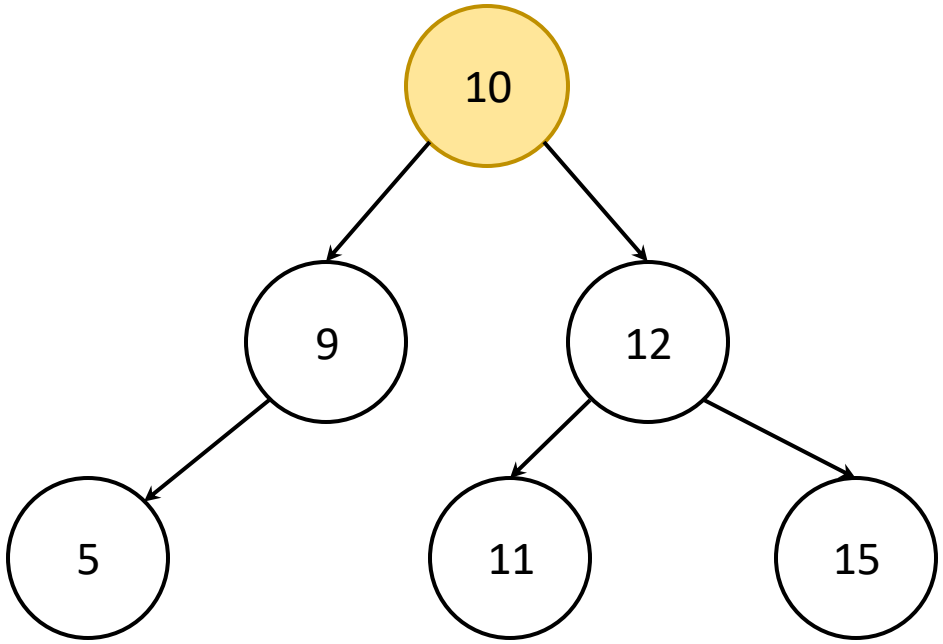
Find
100?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

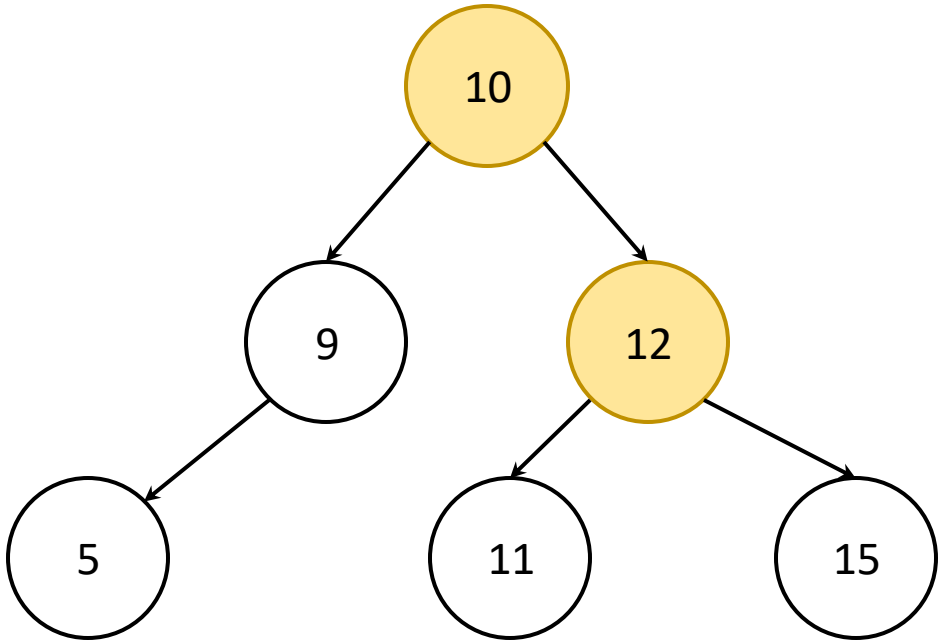
Find
100?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

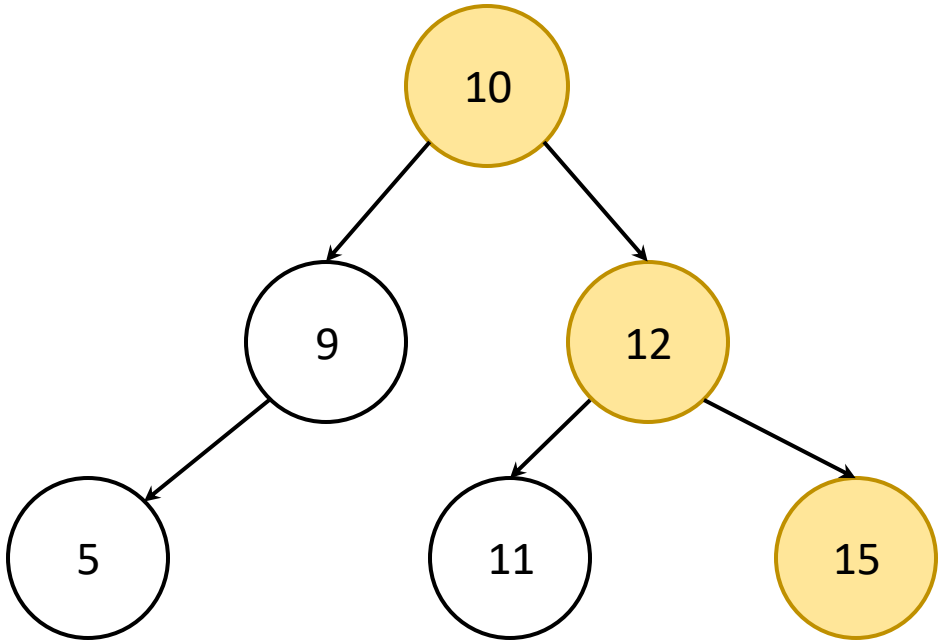
Find
100?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

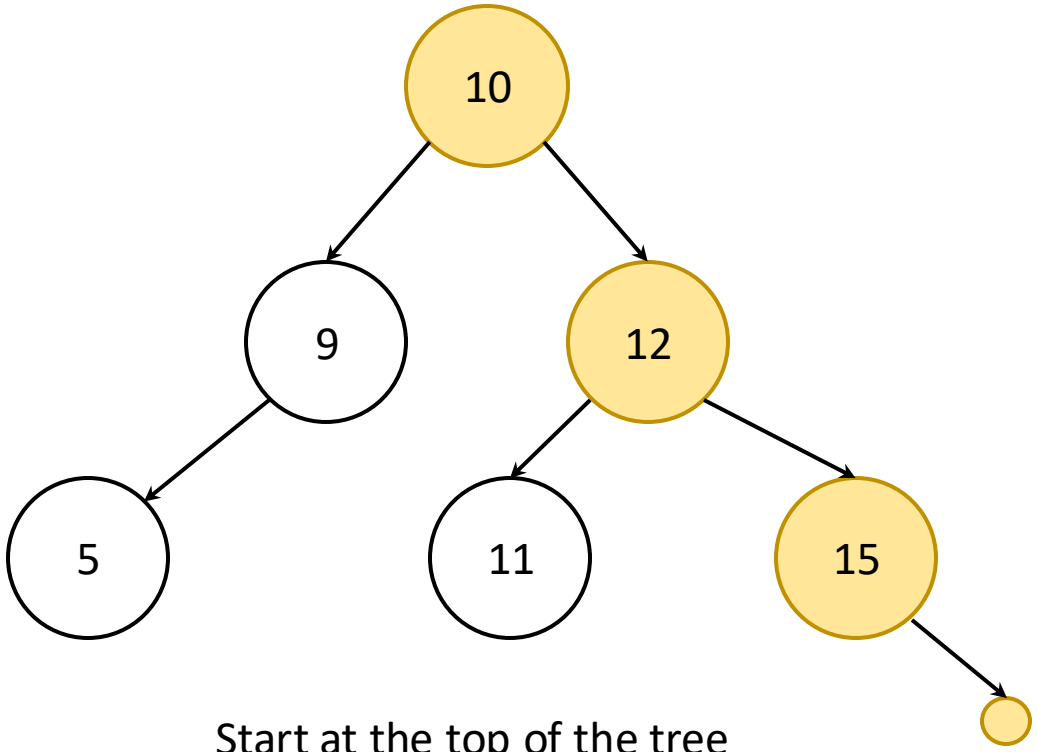
Find
100?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

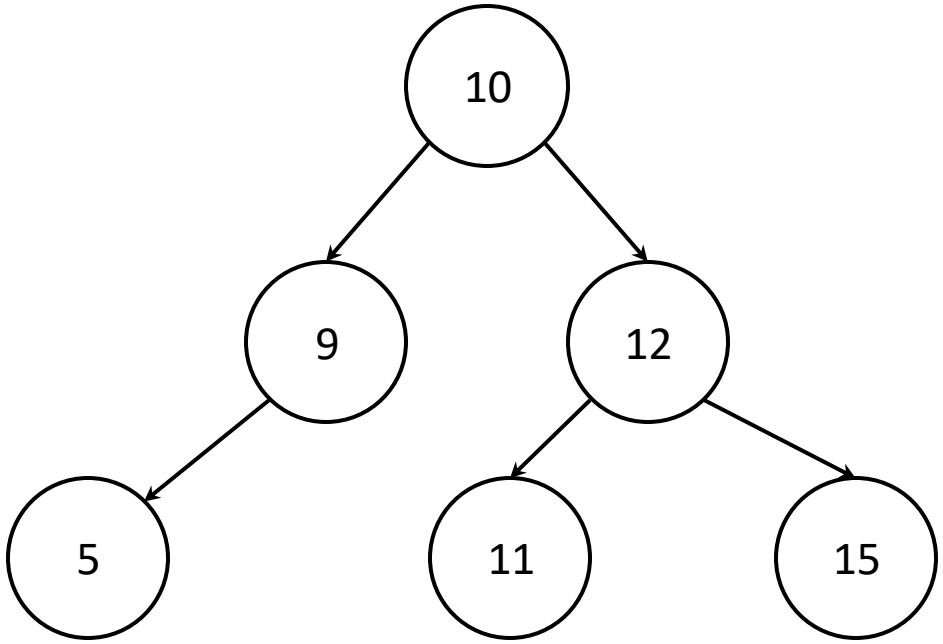
Find
100?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

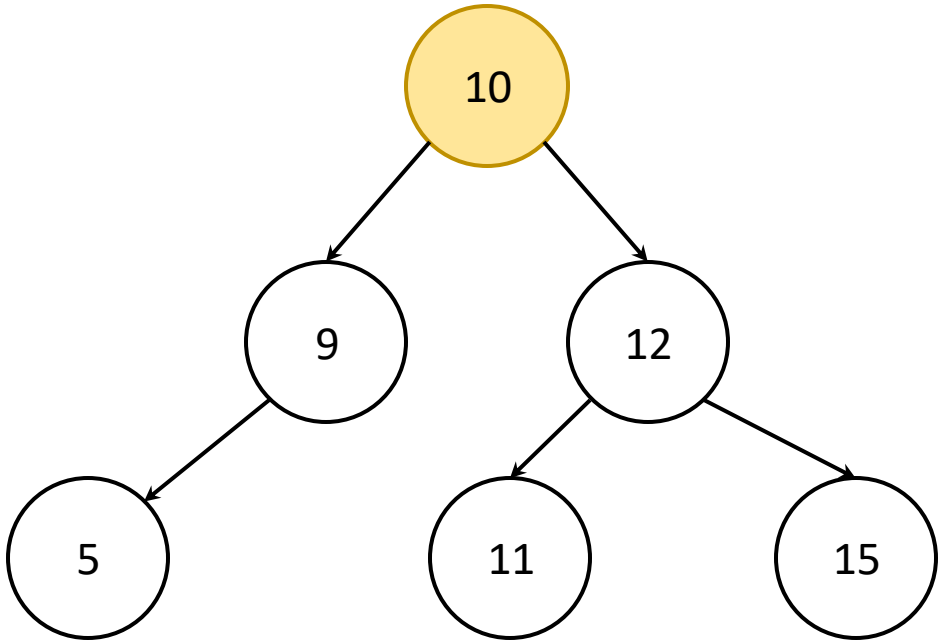
Find 10?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

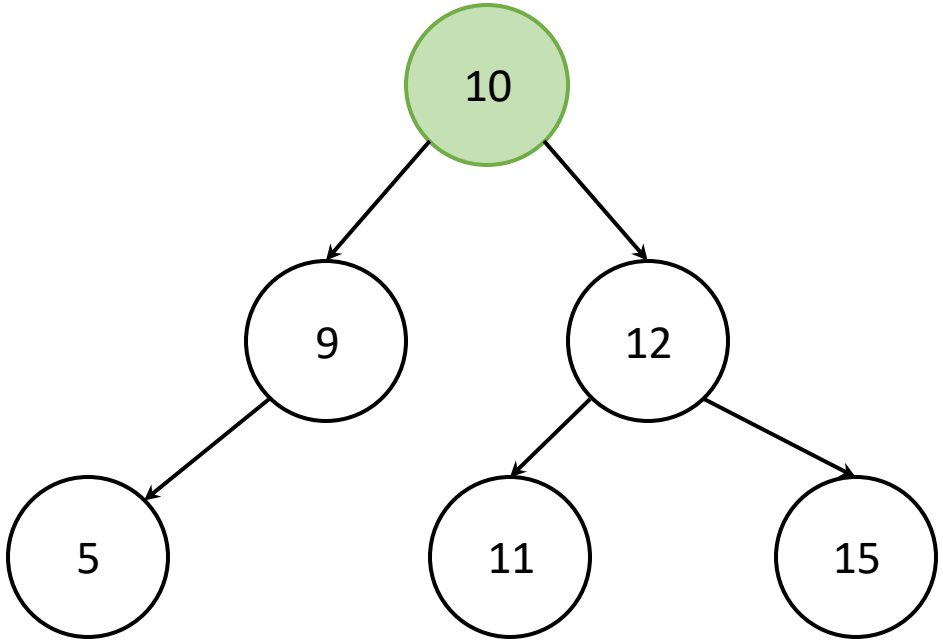
Find 10?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

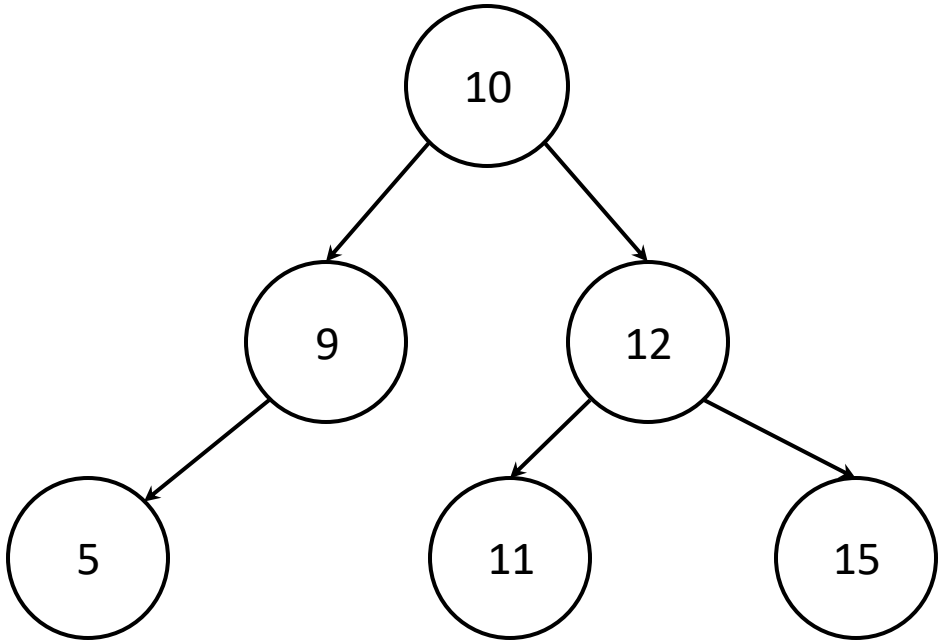
Find 10?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

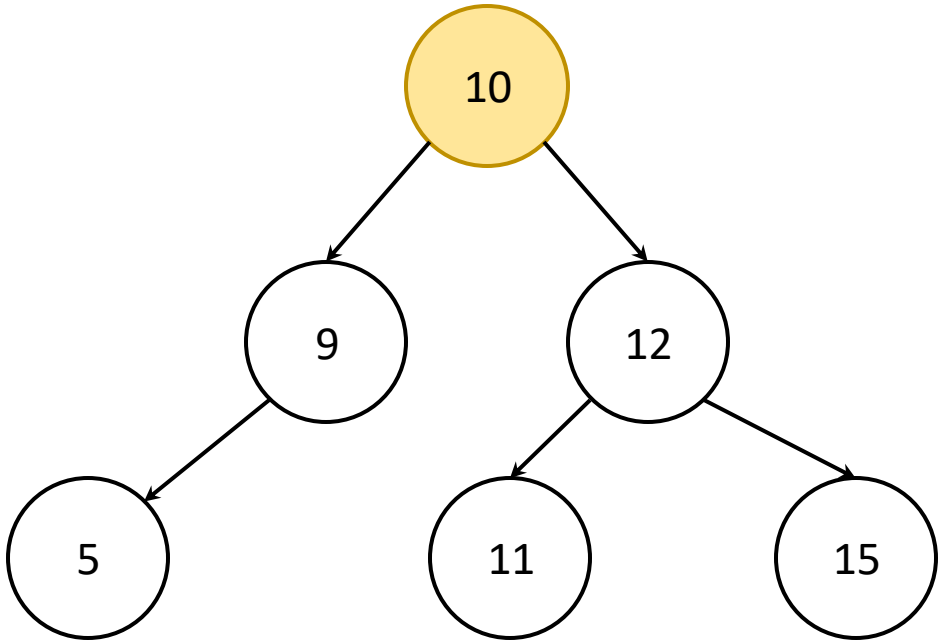
Find 12?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

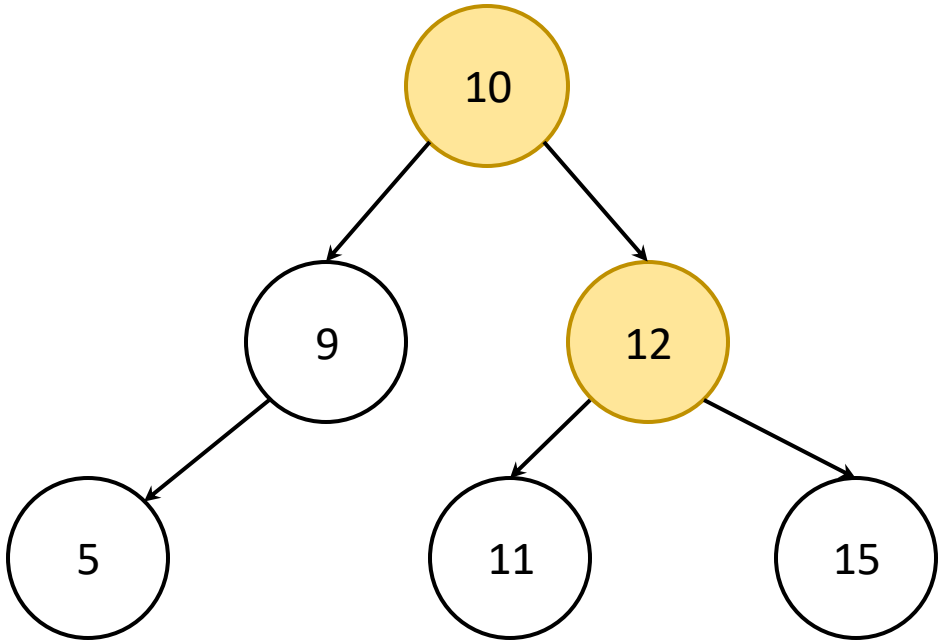
Find 12?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

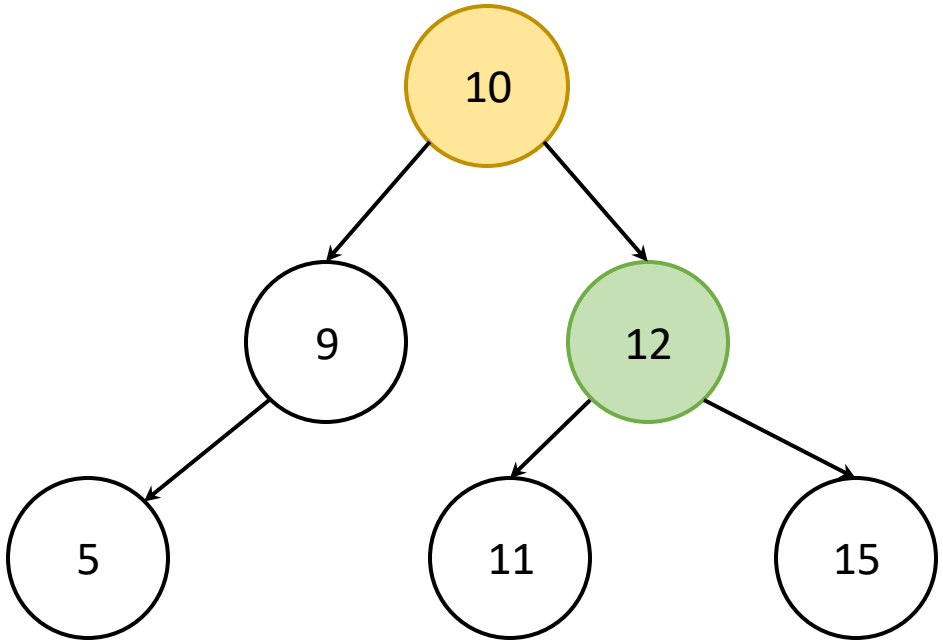
Find 12?



Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST searching

Find 12?



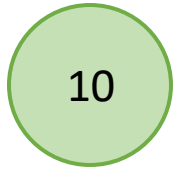
Start at the top of the tree
If root node is empty, the tree is empty
If the value matches our target value, we have the correct node
If our value is smaller repeat the process in the left subtree
If our value is greater repeat the process in the right subtree

BST insertion

- If the root is empty, use new node as root.
- Otherwise we need to compare and determine which subtree the node belongs in.
- If less than current root, put in left subtree.
- If greater than current root, put in right subtree.

BST insertion

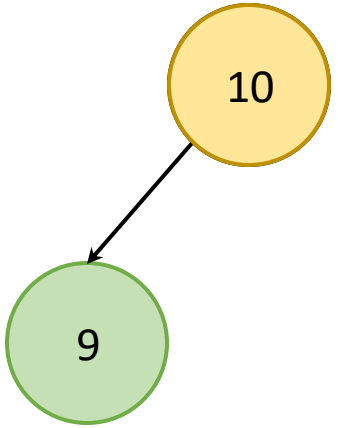
Insert
10



If the root is empty, use new node as root
If less than current root, put in left subtree
If greater than current root, put in right subtree

BST insertion

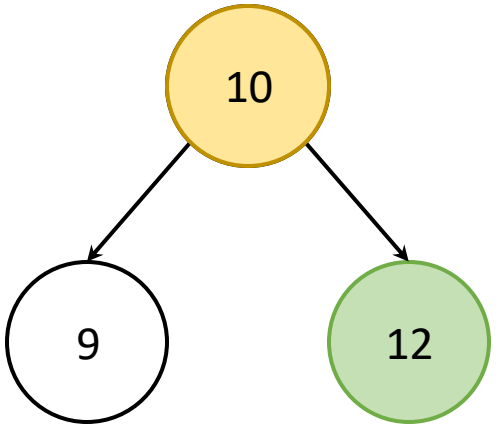
Insert 9



If the root is empty, use new node as root
If less than current root, put in left subtree
If greater than current root, put in right subtree

BST insertion

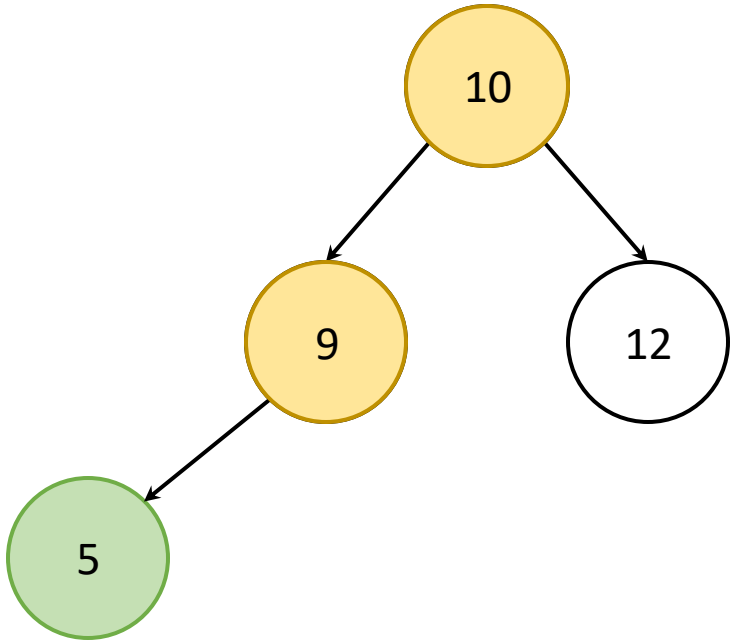
Insert
12



If the root is empty, use new node as root
If less than current root, put in left subtree
If greater than current root, put in right subtree

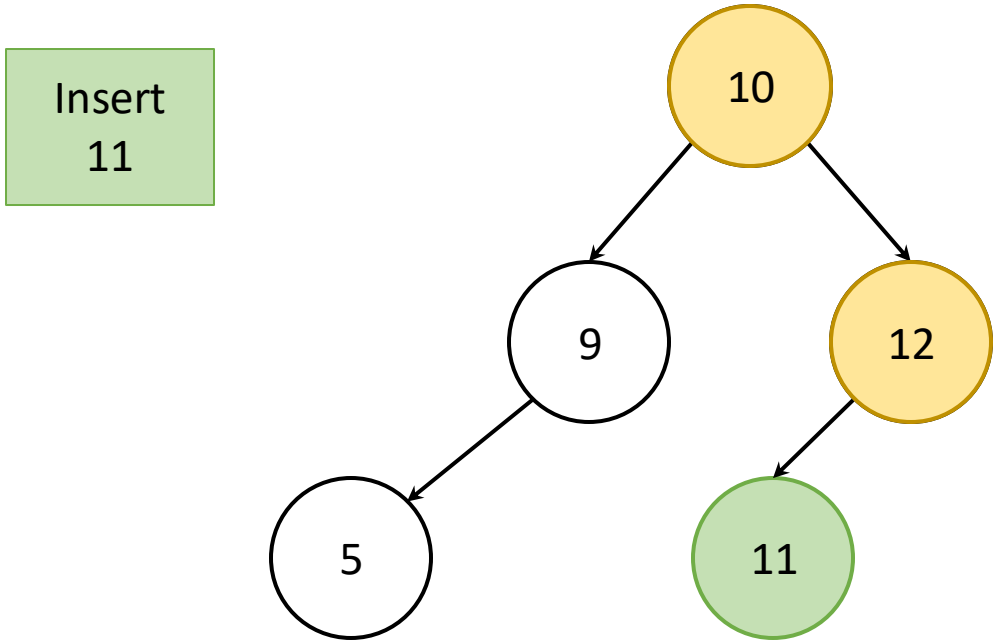
BST insertion

Insert 5



If the root is empty, use new node as root
If less than current root, put in left subtree
If greater than current root, put in right subtree

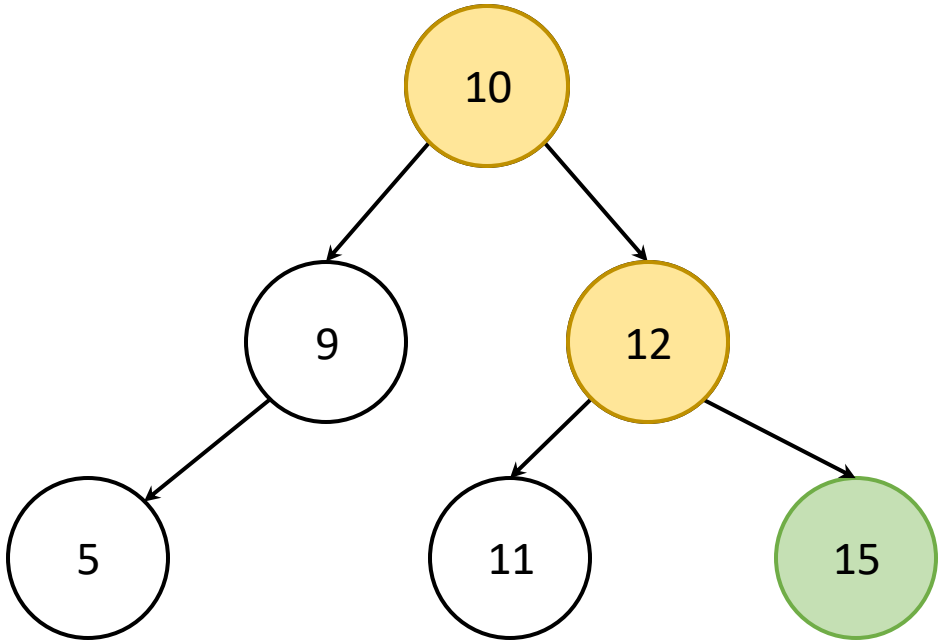
BST insertion



If the root is empty, use new node as root
If less than current root, put in left subtree
If greater than current root, put in right subtree

BST insertion

Insert
15



If the root is empty, use new node as root
If less than current root, put in left subtree
If greater than current root, put in right subtree

BST deletion

Three cases for deletion:

- Leaf node: simply remove node from tree.
- Single Parent: node has only one child, replace the node with its child.
- Two Children: node has two subtrees, but which node to replace with?

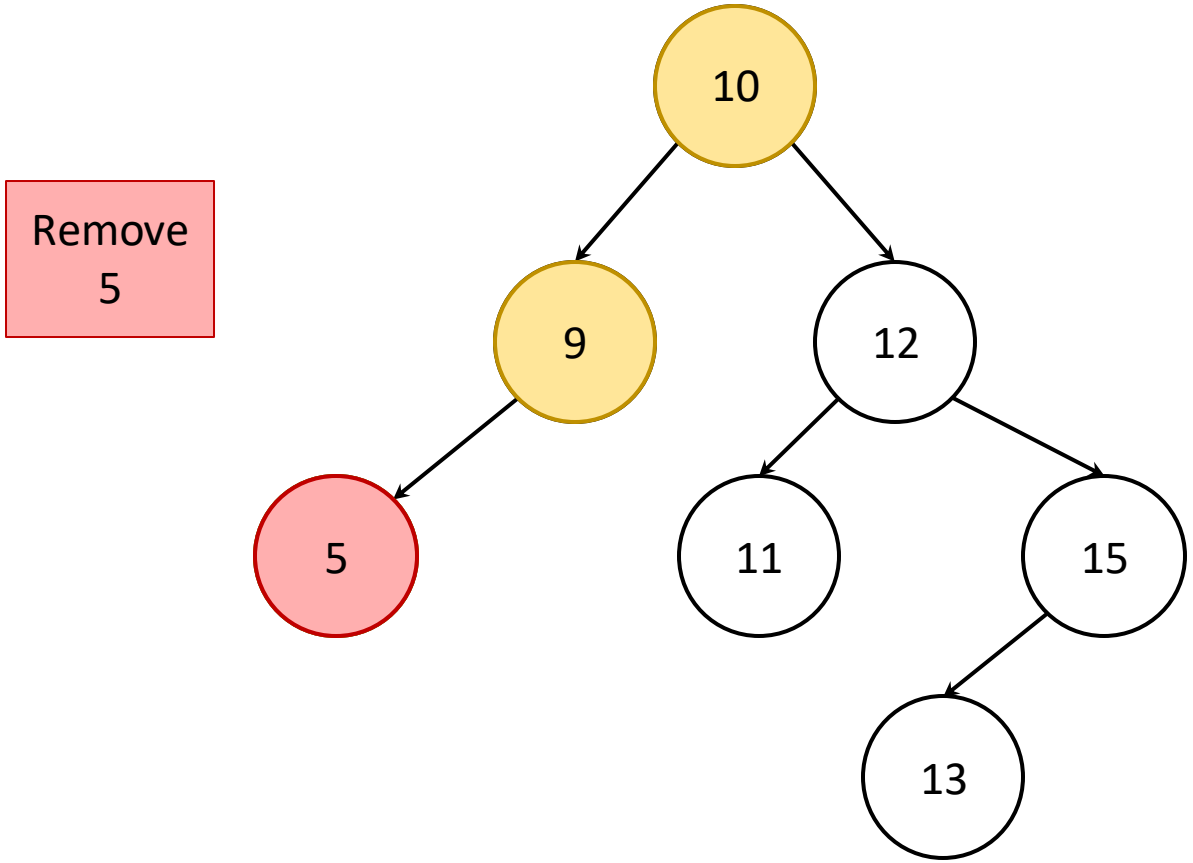
BST deletion

Three cases for deletion:

- Leaf node: simply remove node from tree.
- Single Parent: node has only one child, replace the node with its child.
- Two Children: node has two subtrees, but which node to replace with?

    Typically the left-most node of the right subtree

BST deletion

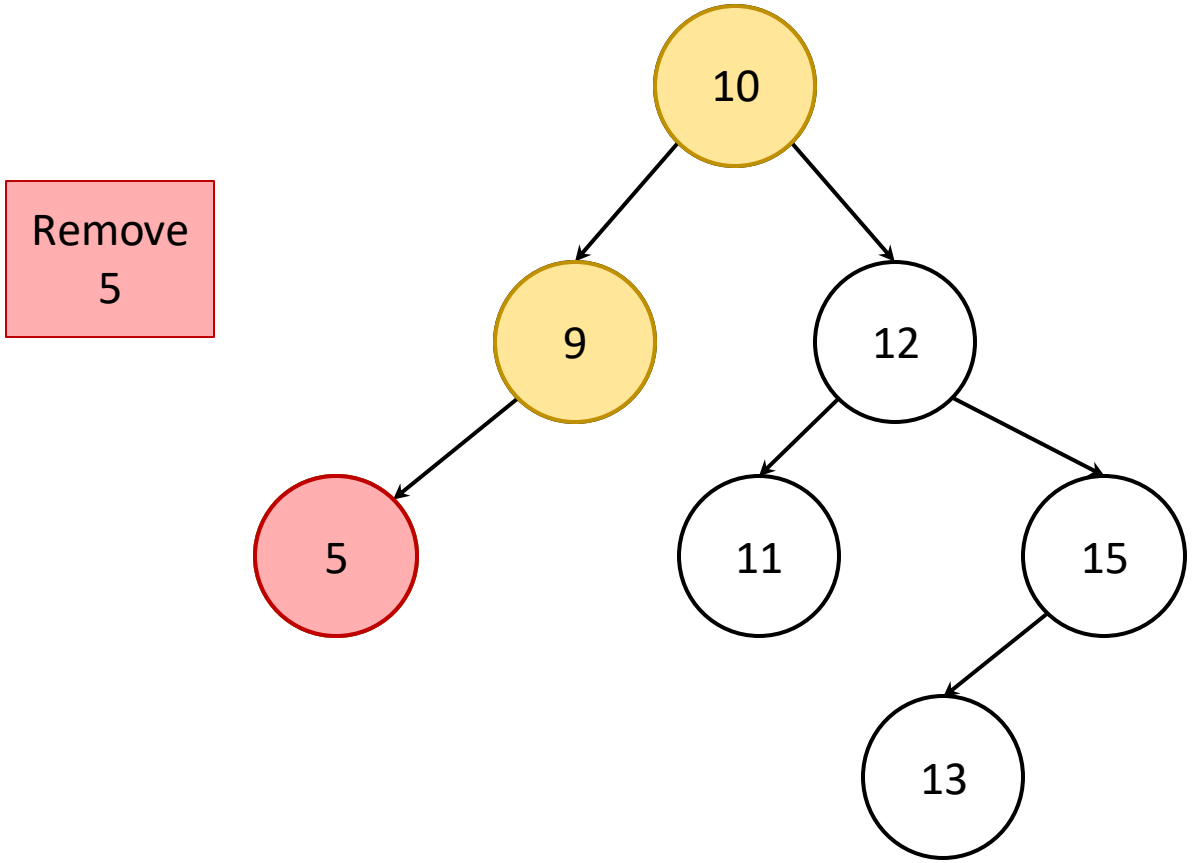


Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion



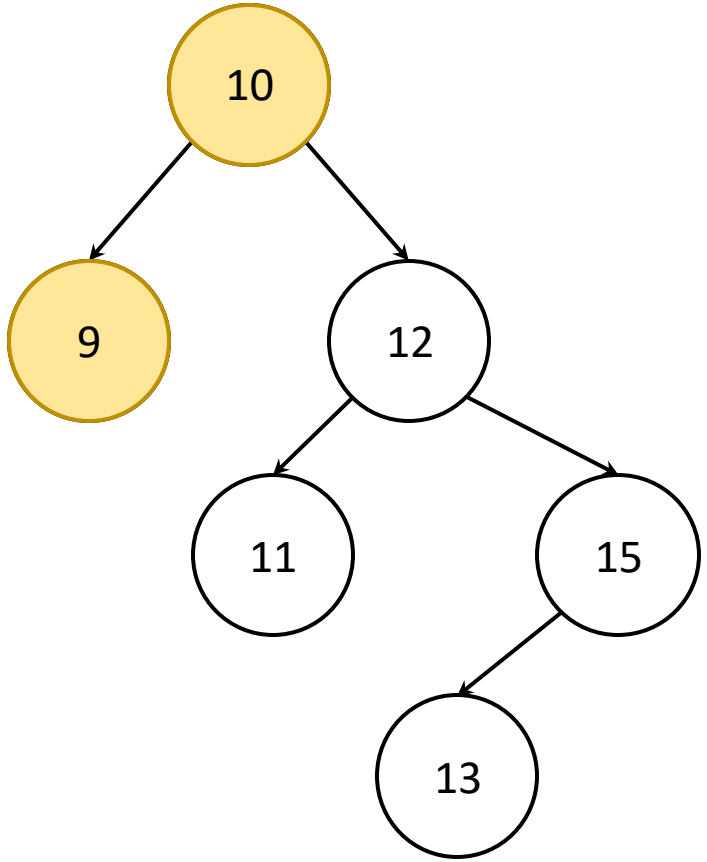
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 5

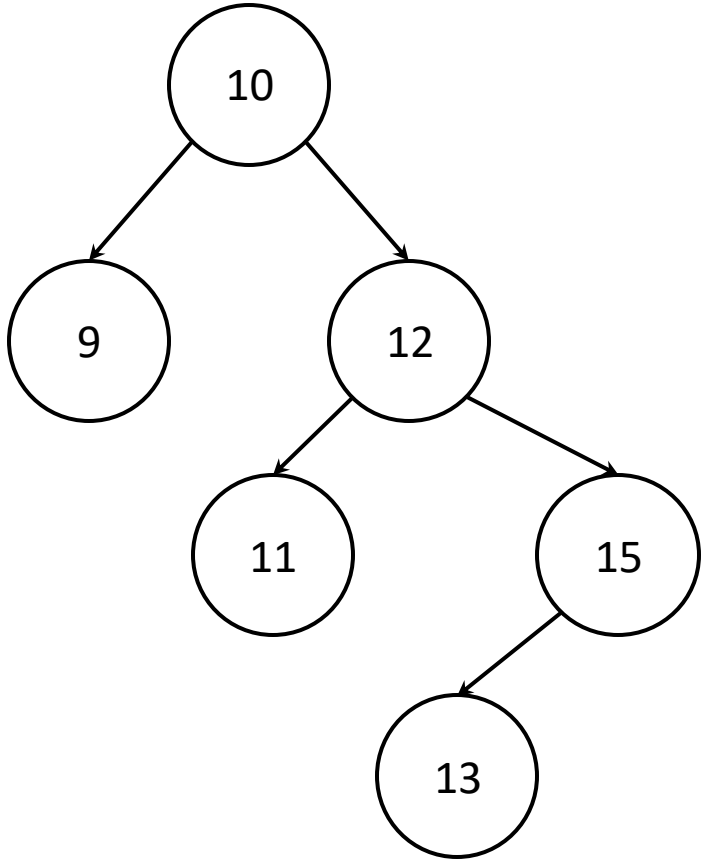


Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion



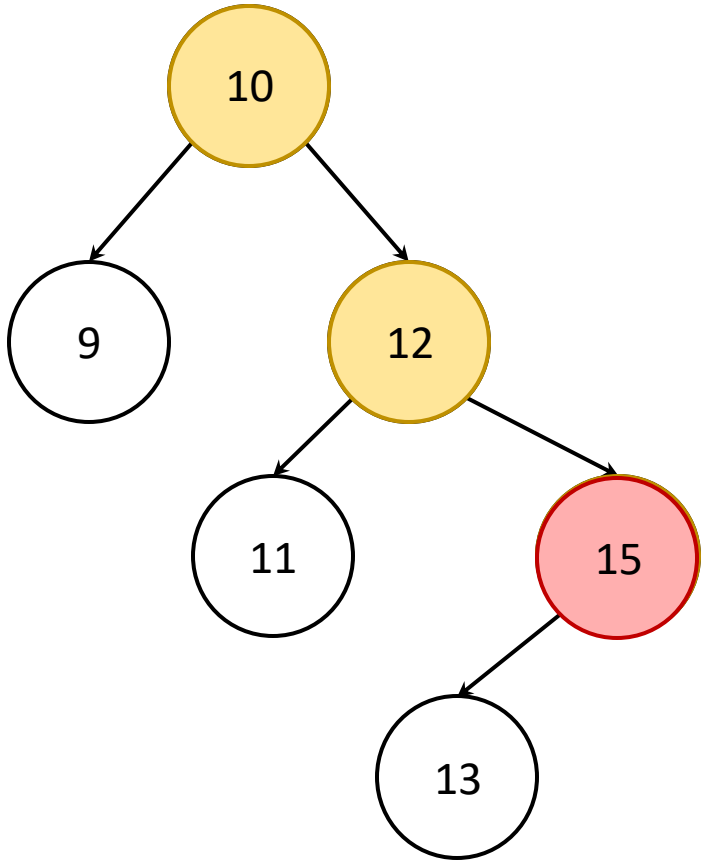
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 15



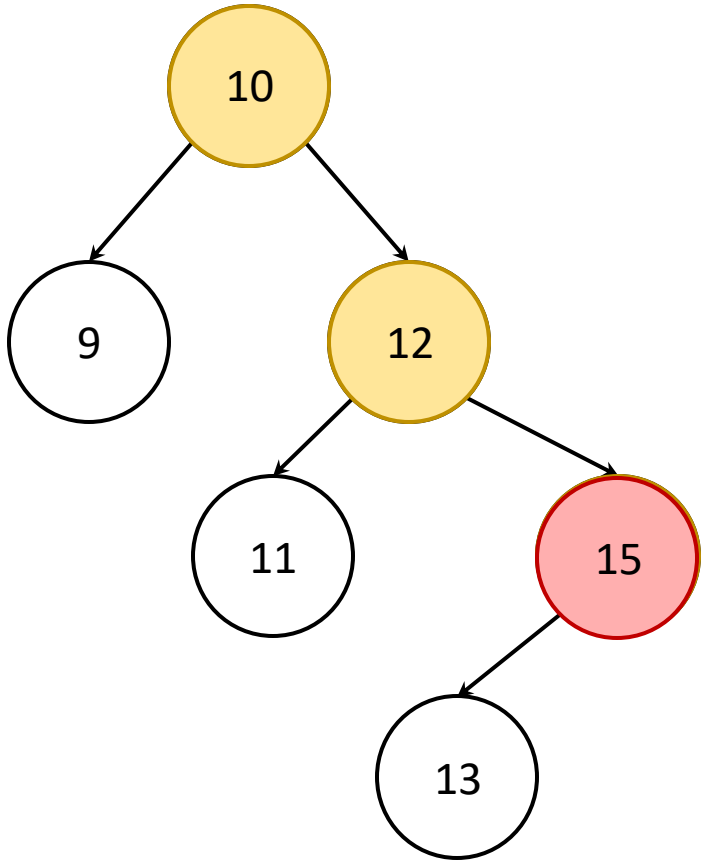
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 15



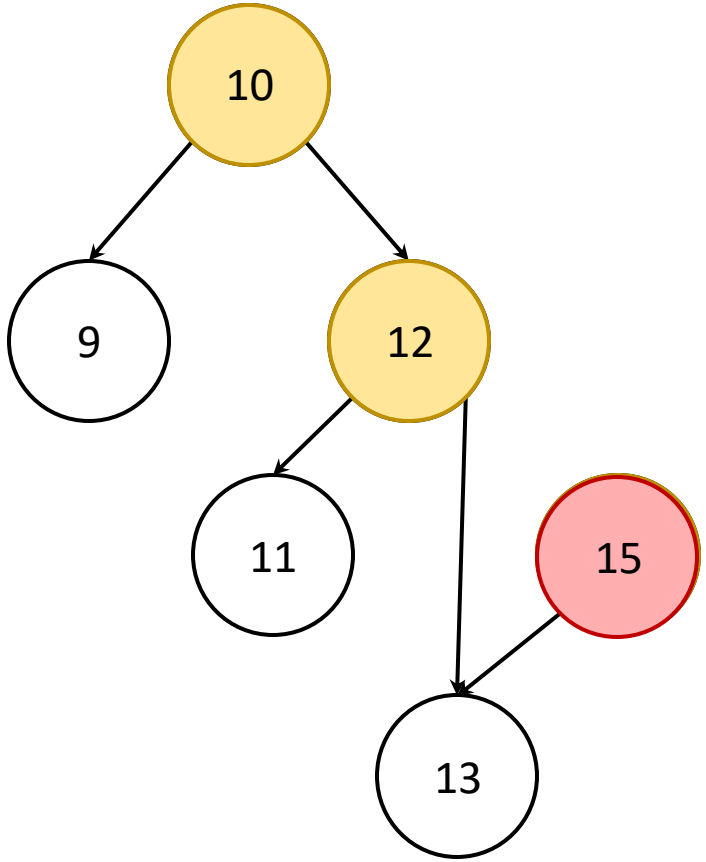
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 15



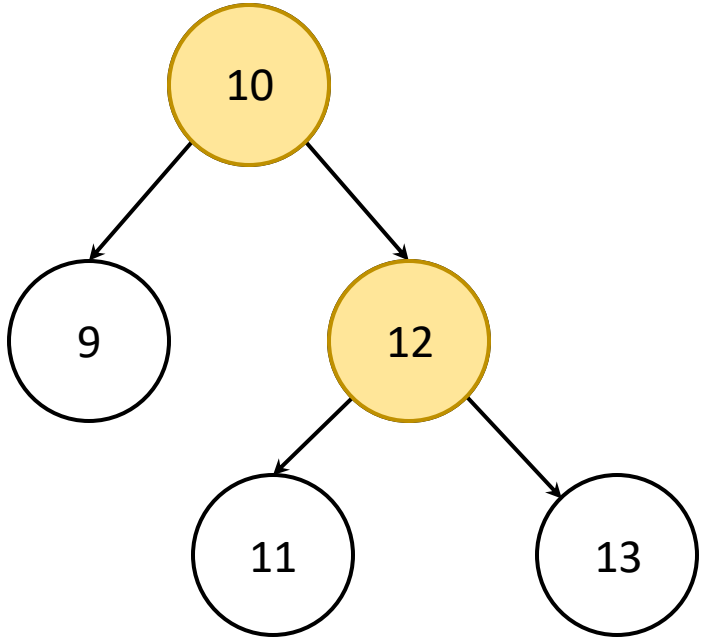
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 15

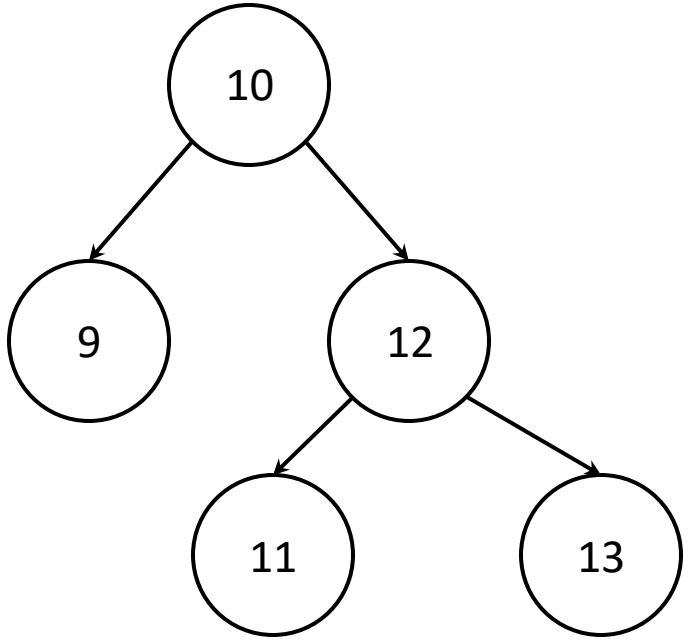


Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion



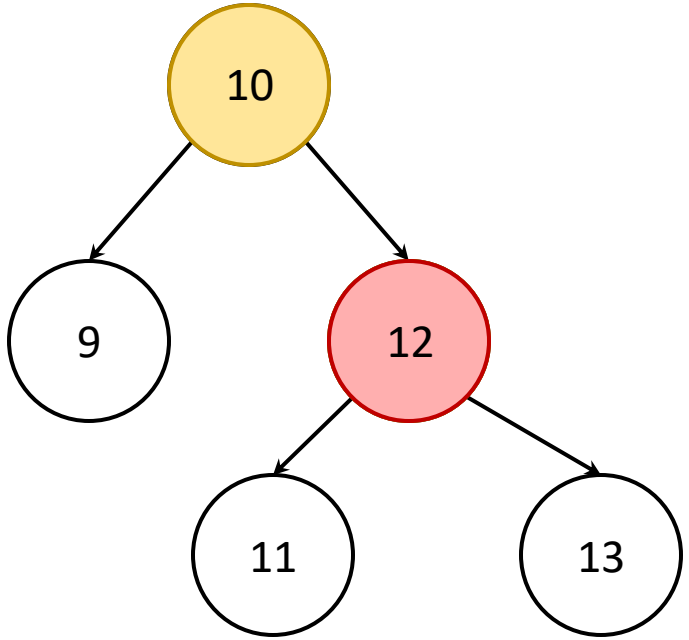
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12



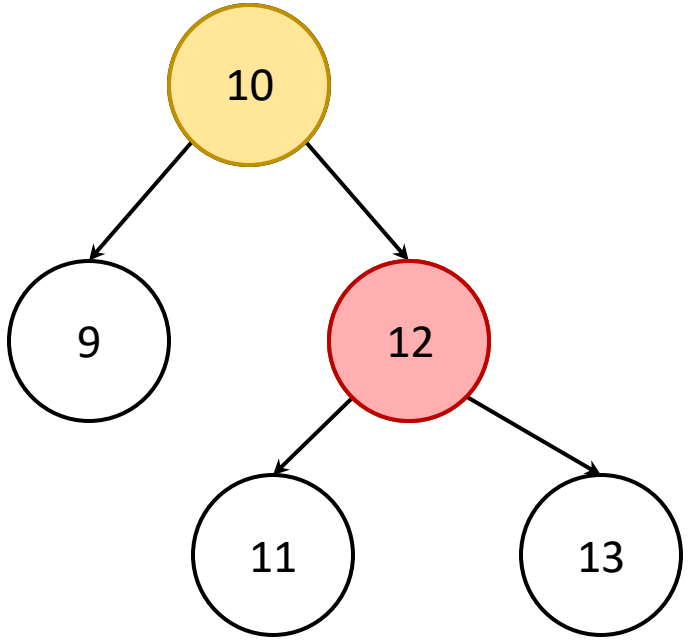
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12



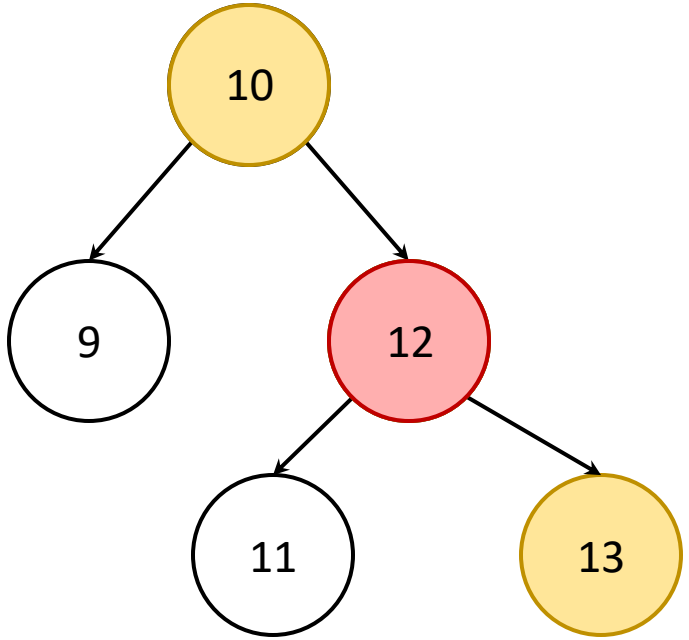
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12



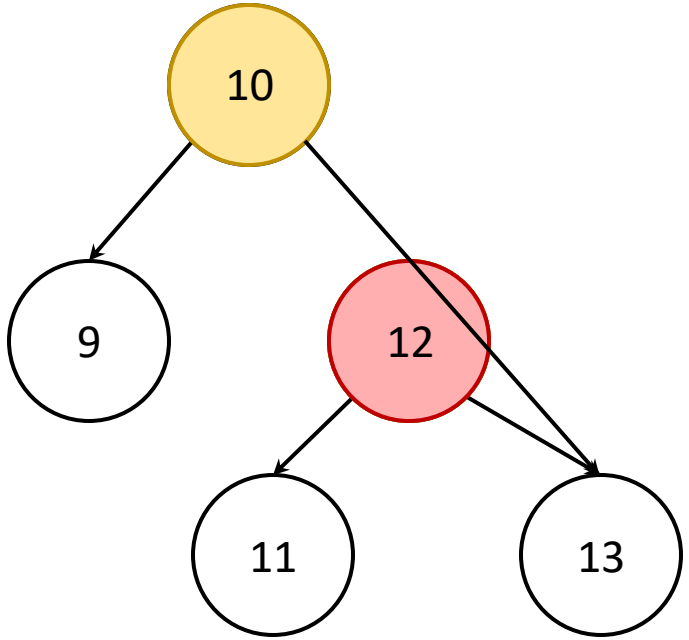
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12



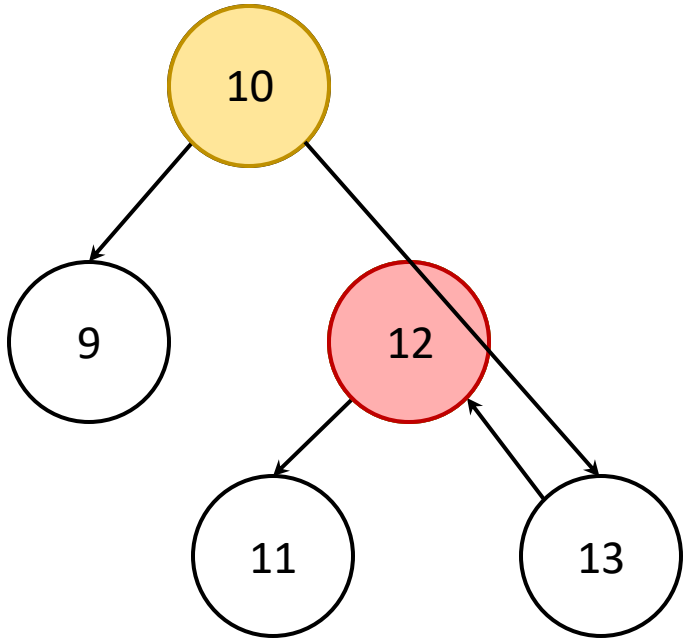
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12



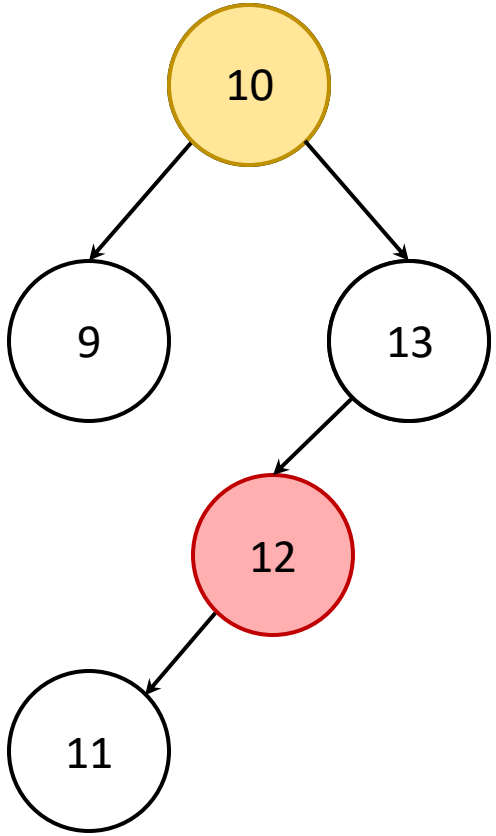
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12



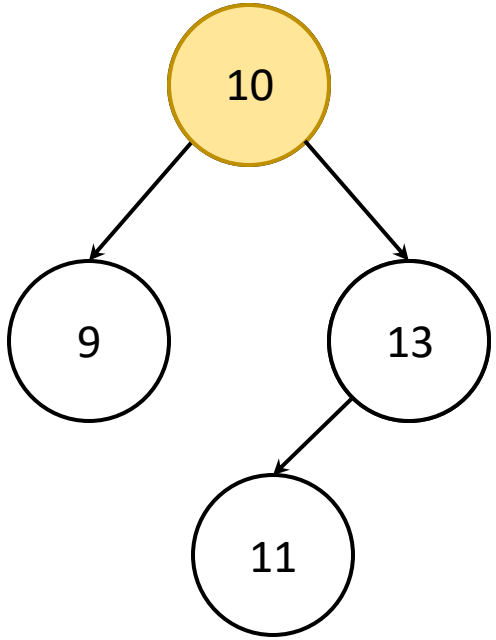
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion

Remove 12

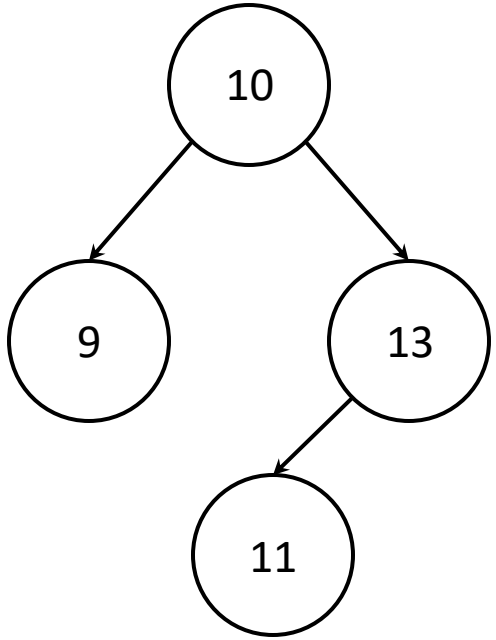


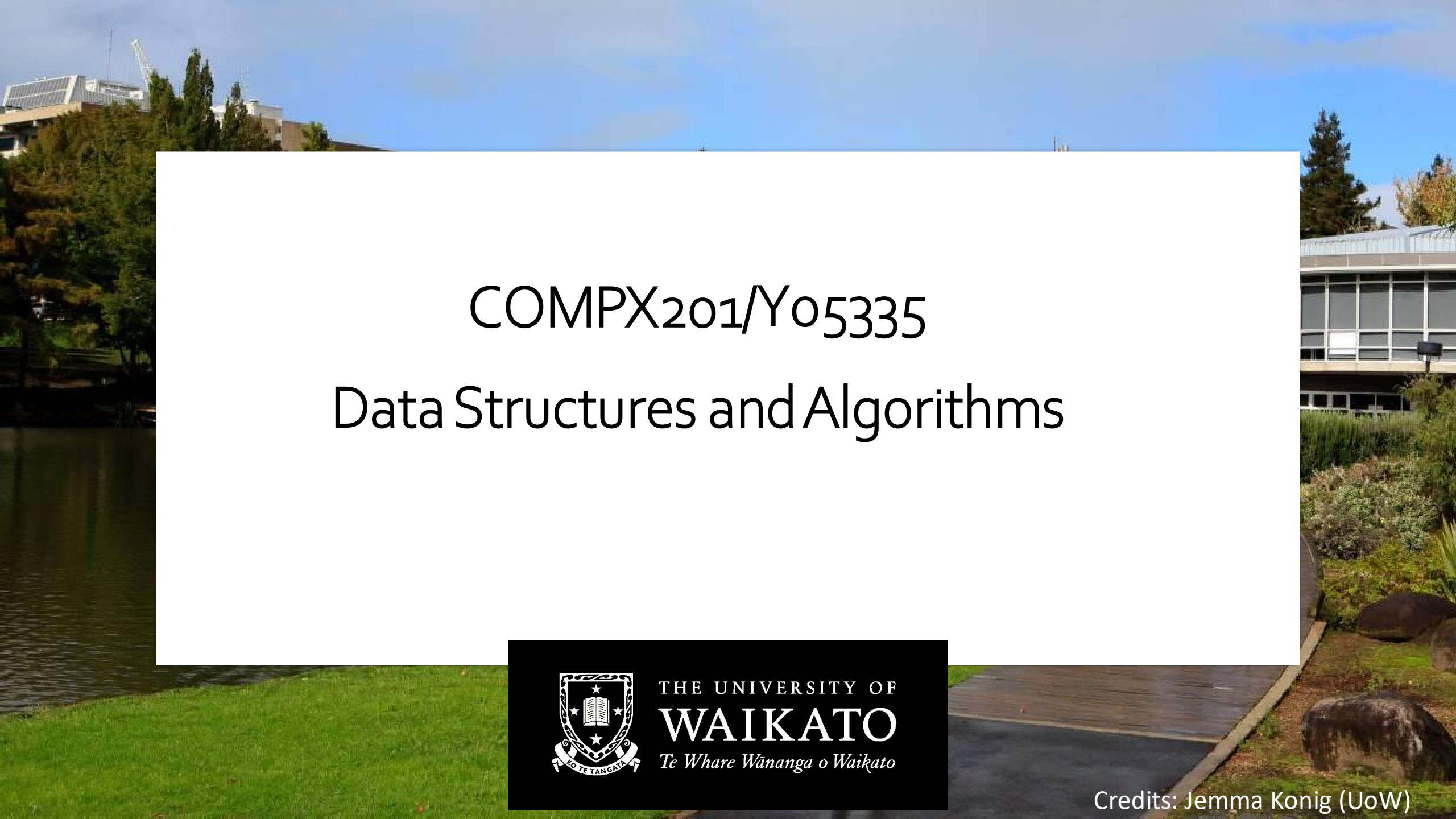
Leaf node: simply remove node from tree.

Single Parent: node has only one child, replace the node with its child.

Two Children: node has two subtrees, but which node to replace with? *Typically the left-most node of the right subtree*

BST deletion





COMPX201/Y05335

Data Structures and Algorithms

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Credits: Jemma Konig (UoW)

Binary Search Trees
Height, Traversal and, Balancing

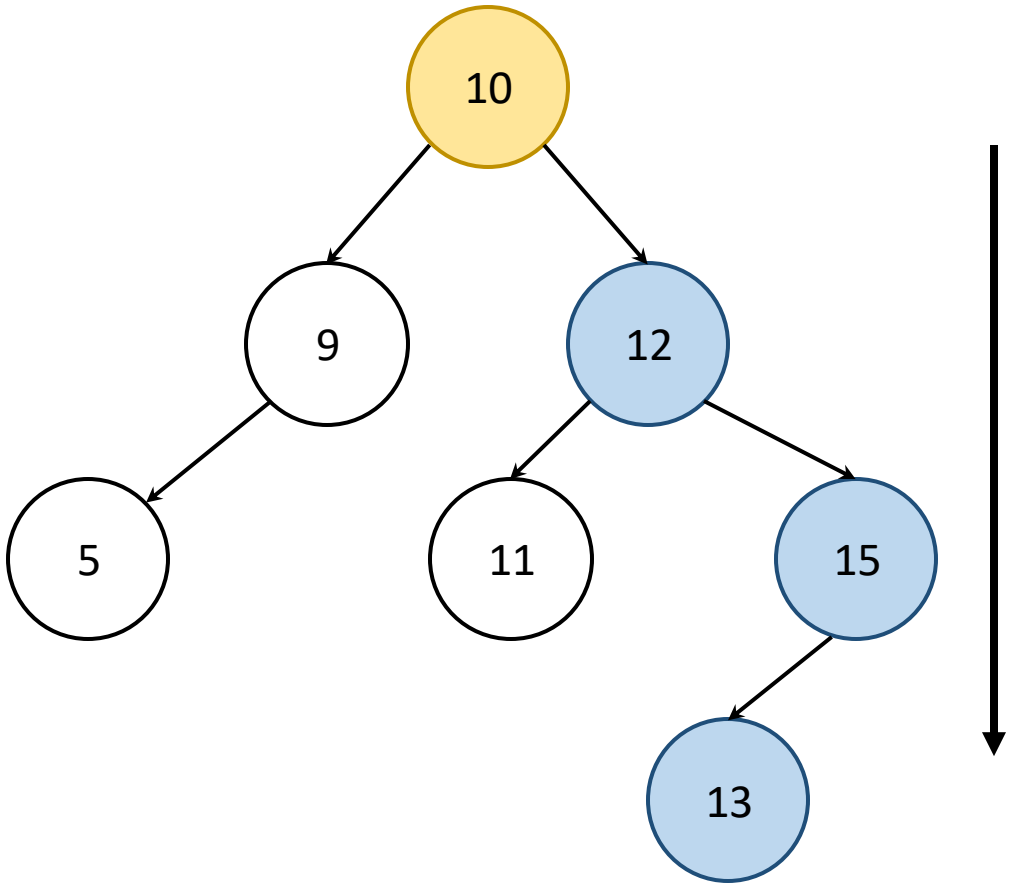
COMPX201/Y05335

Overview

- Height
- Depth
- Traversal
- Balancing

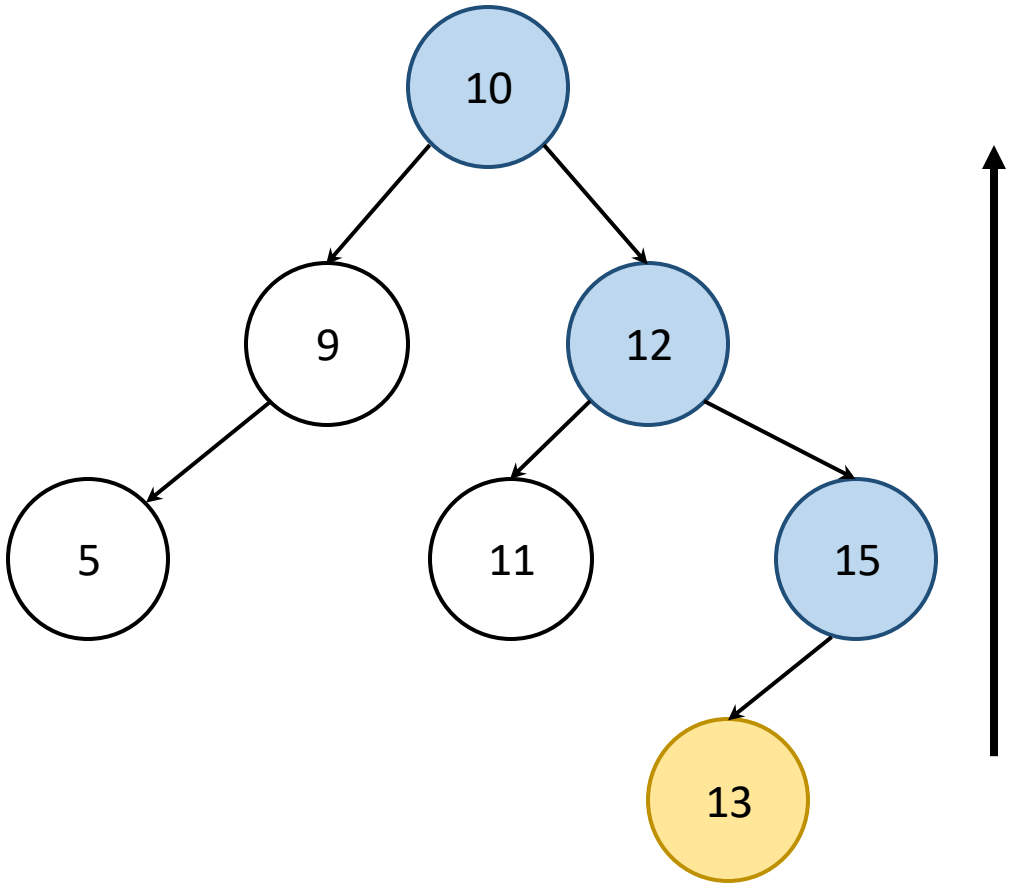
BST Height

- The longest path from some node to a leaf is said to be the "height" of the tree.
- Height is useful for "balancing" the tree.

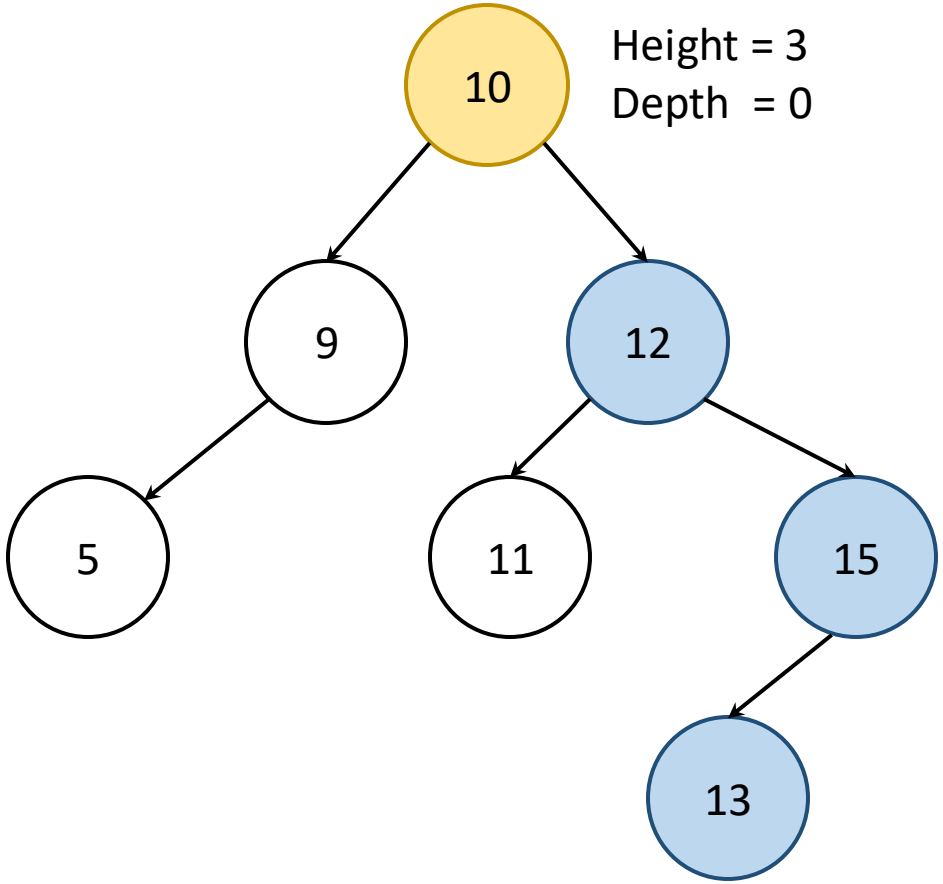


BST Depth

- The longest path from some node to a root is said to be the "depth" of the tree.

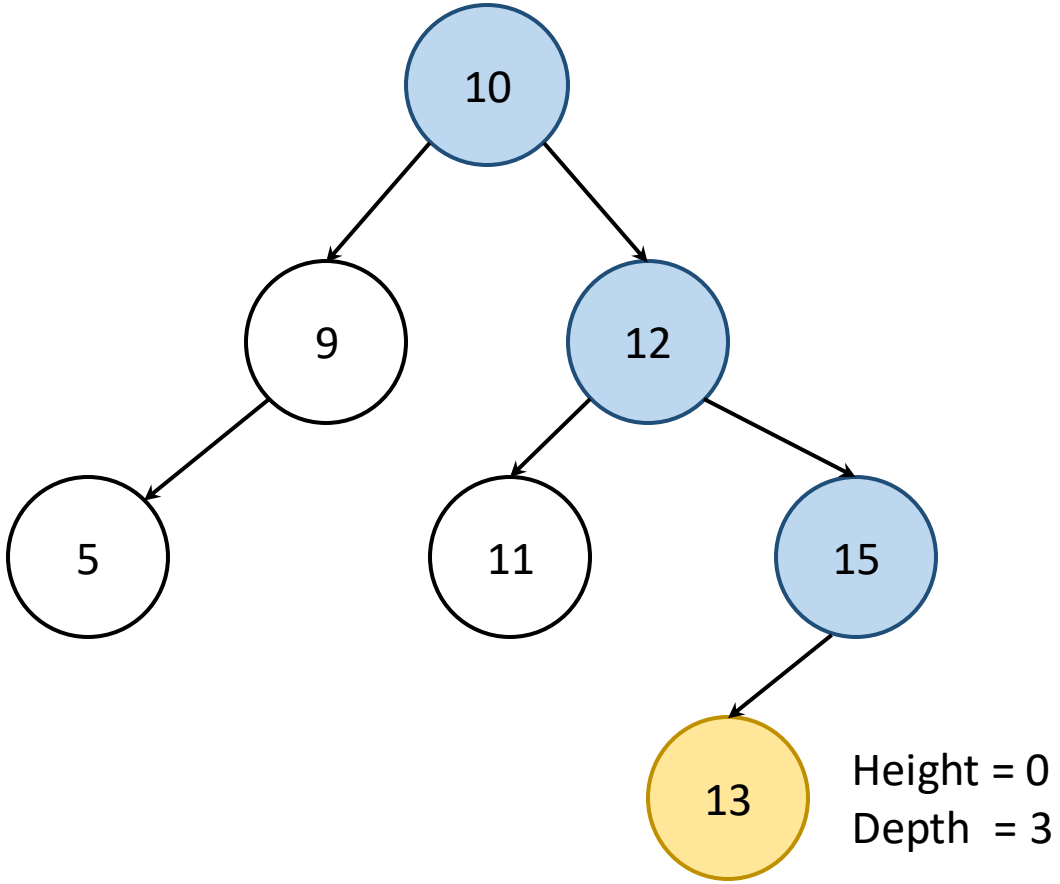


BST height and depth



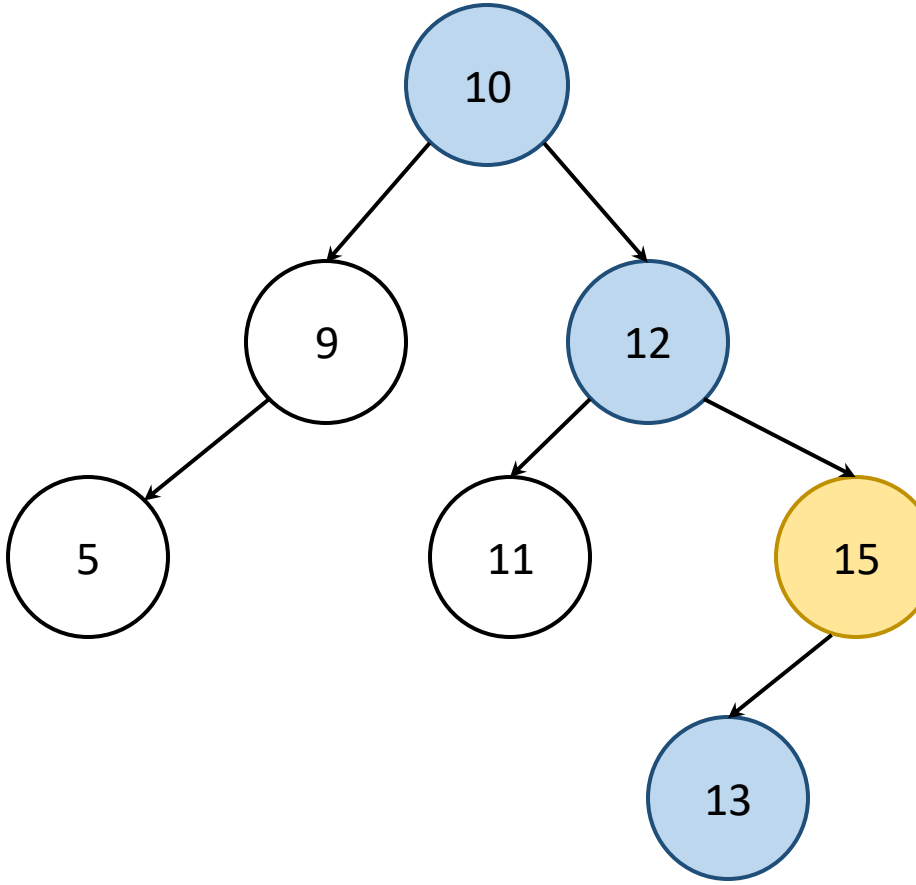
Height: the longest path from some node to a leaf
Depth: the longest path from some node to a root

BST height and depth



Height: the longest path from some node to a leaf
Depth: the longest path from some node to a root

BST height and depth



Height: the longest path from some node to a leaf
Depth: the longest path from some node to a root

BST traversal

- Visiting all nodes in a tree is called a “traversal”.
- Involves moving through all nodes in linear time (just like a list)

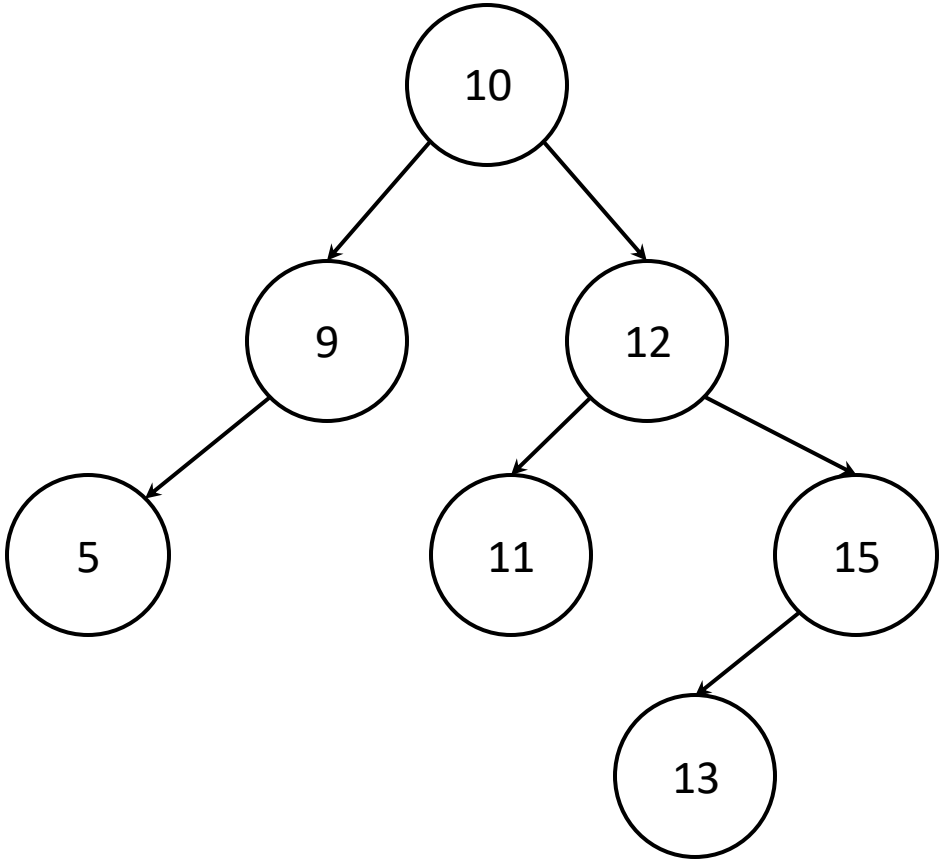
```
public void traverse () {
          CALL traverse on left subtree
          PROCESS the root node
          CALL traverse on right subtree
}
```

- Everything in the left sub-tree is processed first before moving to the right.
- So nodes get processed in order.
- Note algorithm doesn’t show how to stop recursion or what to do when the root is null.

BST traversal

- There are several types of traversal:
- Depth-first:
  - In-order: traverse the tree as discussed on previous slide.
  - Pre-order: process root, then left and right subtrees.
  - Post-order: traverse left and right subtrees, then the root.
  - Iterative deepening: only search to a certain depth and increase until an item is found.
- Breadth-first: traverses the tree in a breadth-ward motion (uses a queue to track nodes).
- … and others …

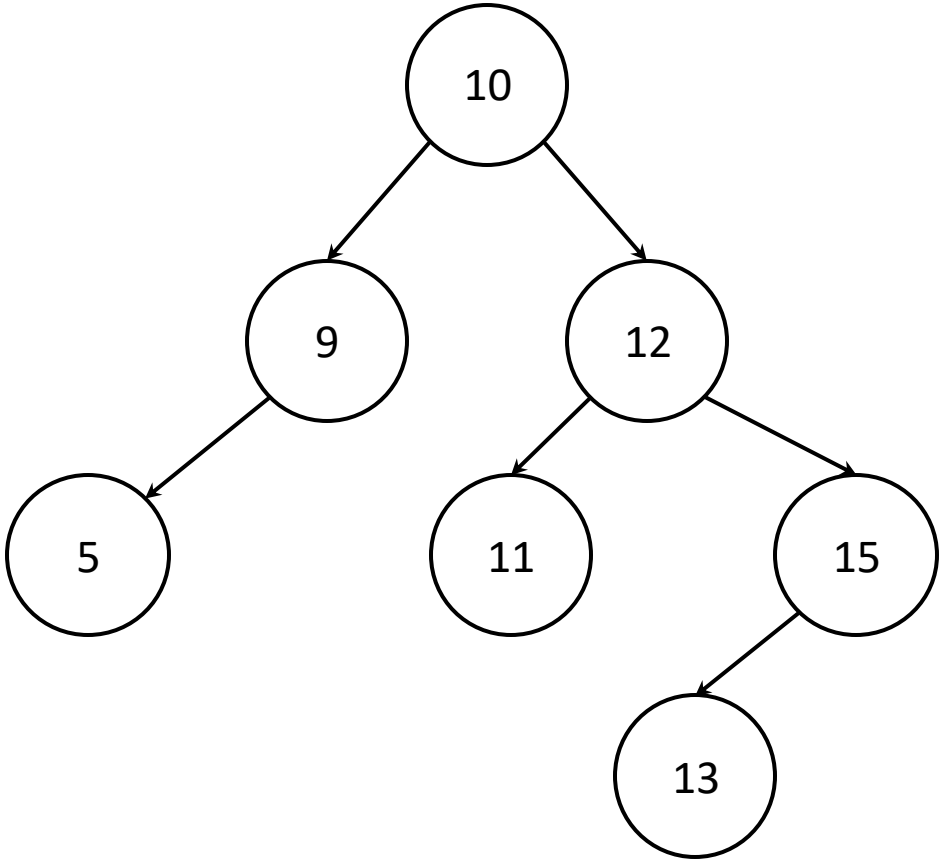
BST traversal



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

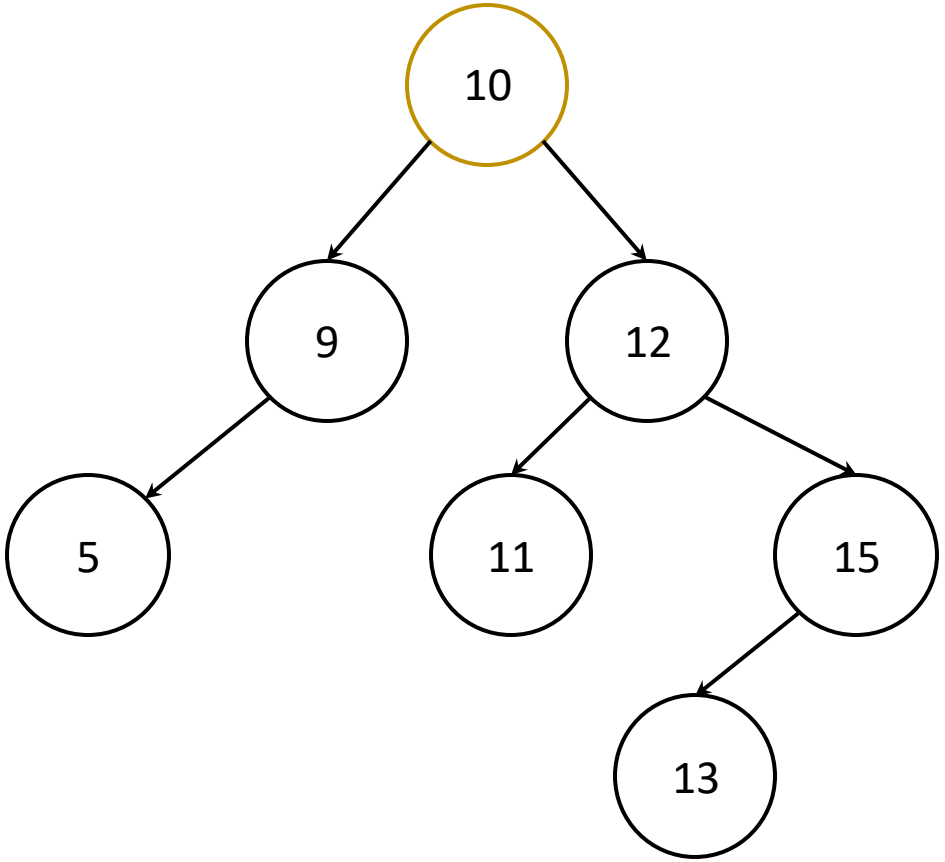
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

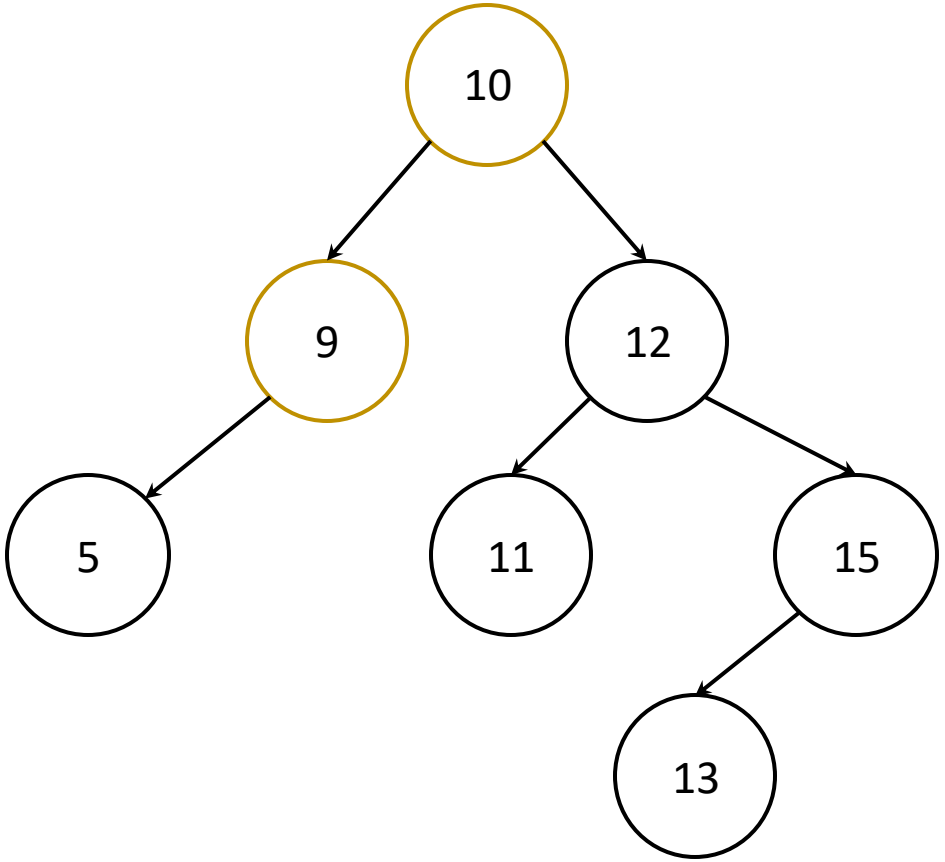
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

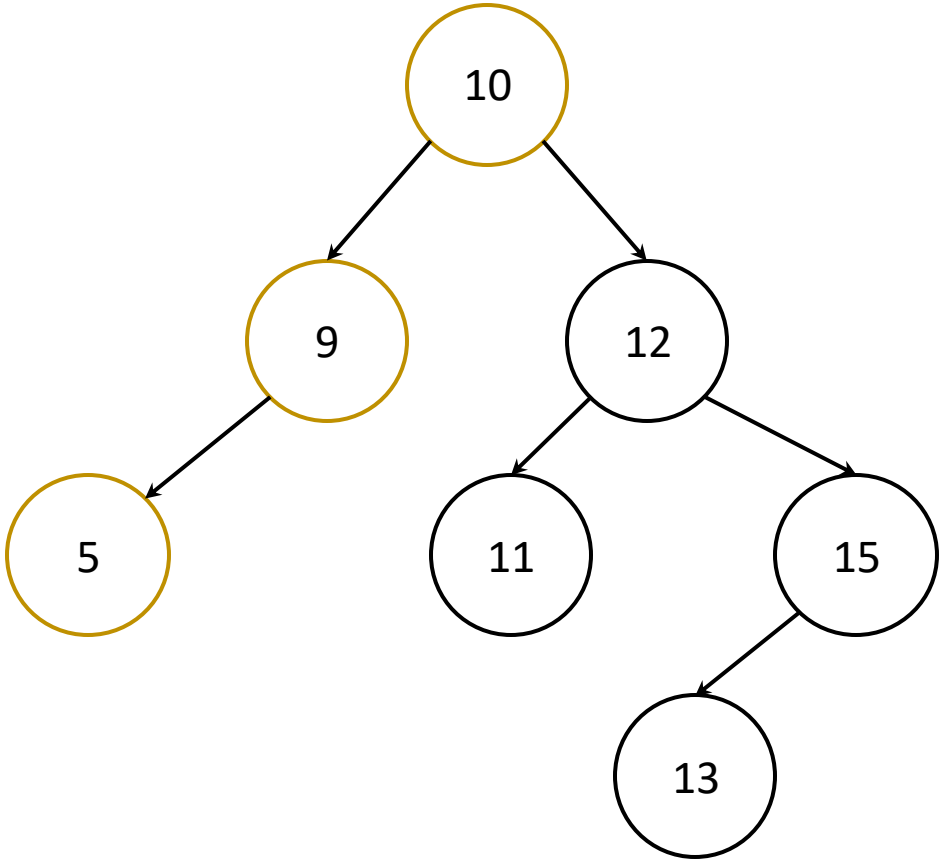
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

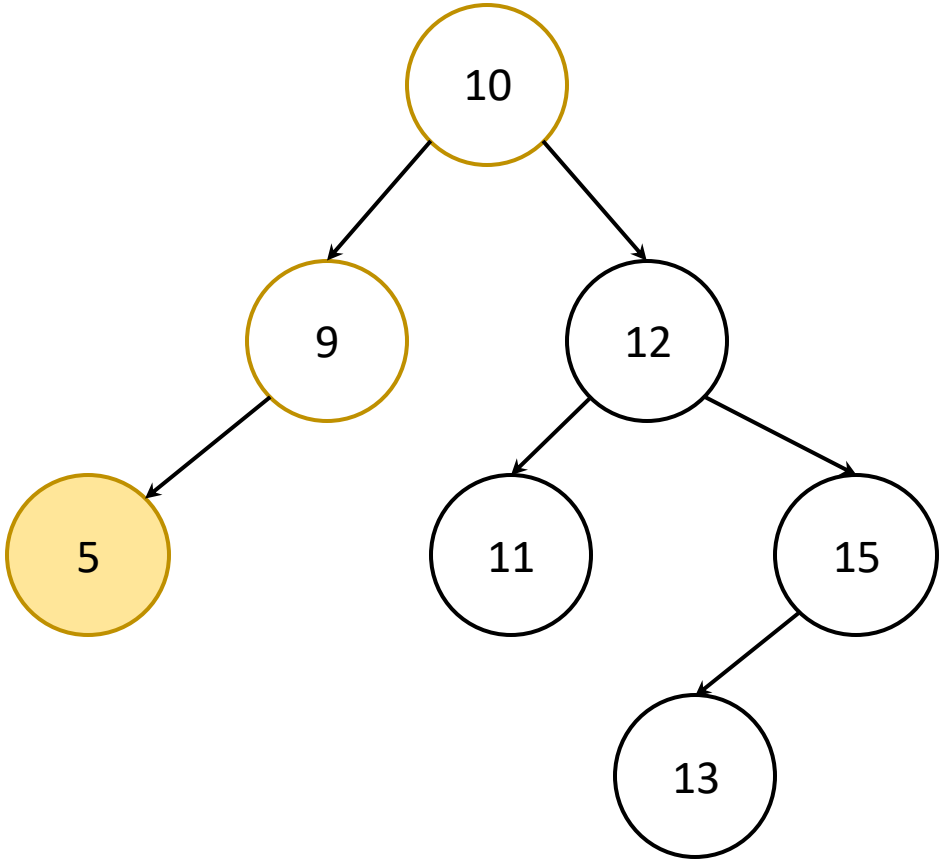
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

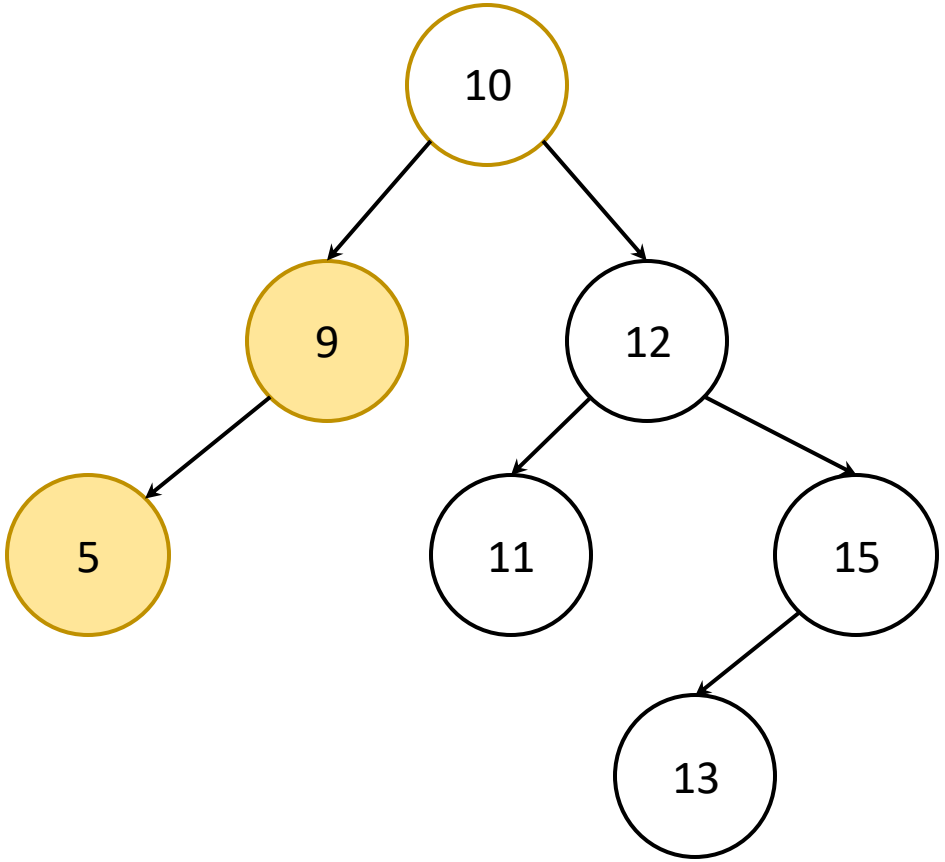
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

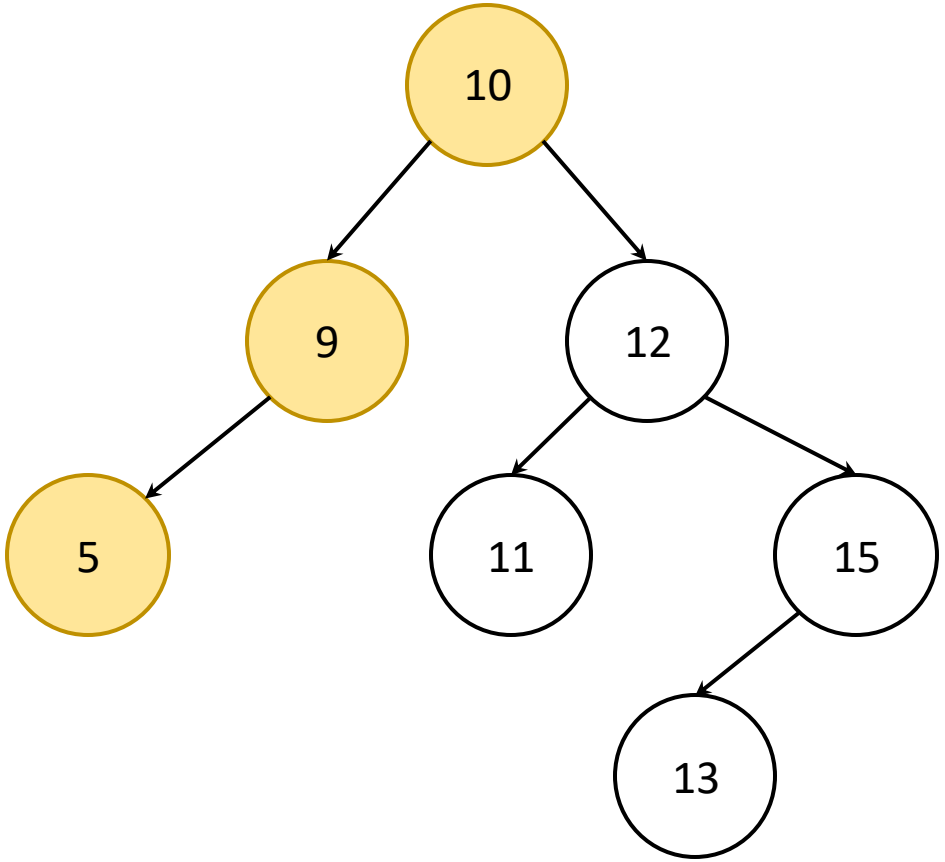
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

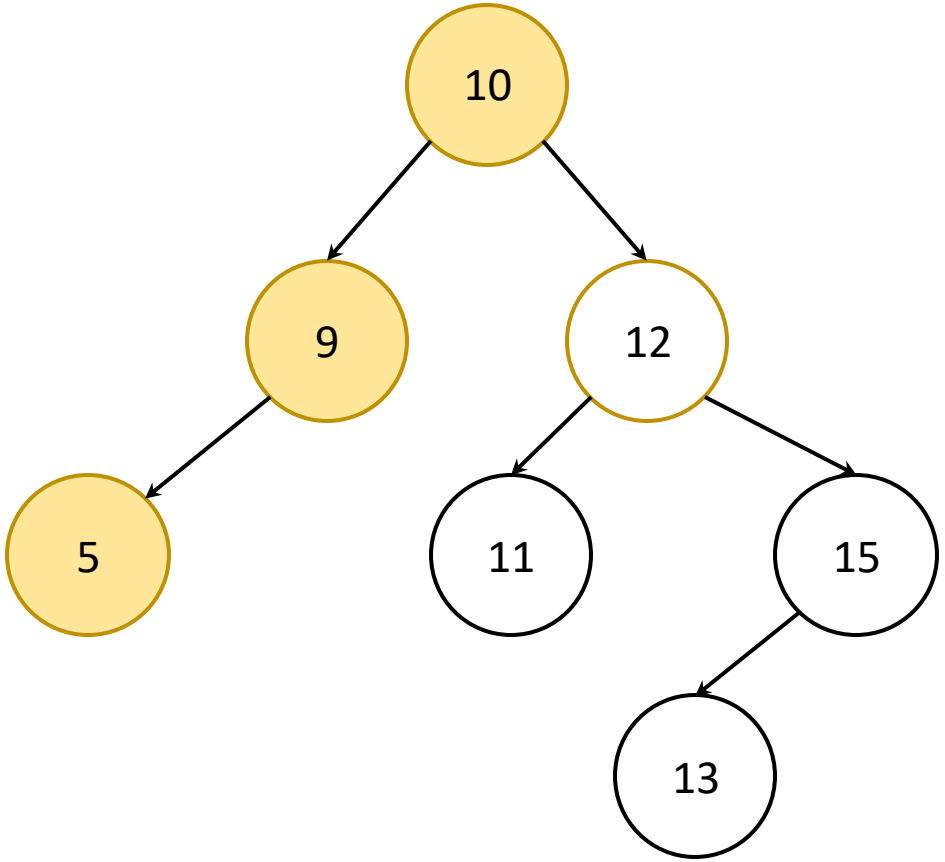
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

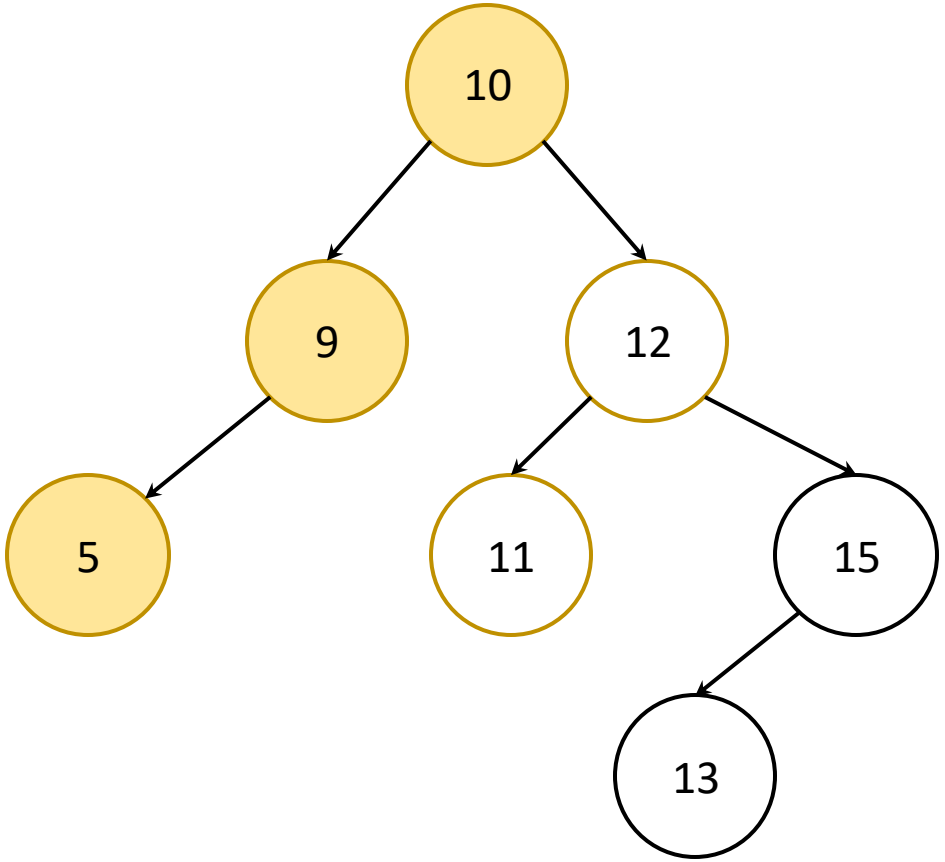
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

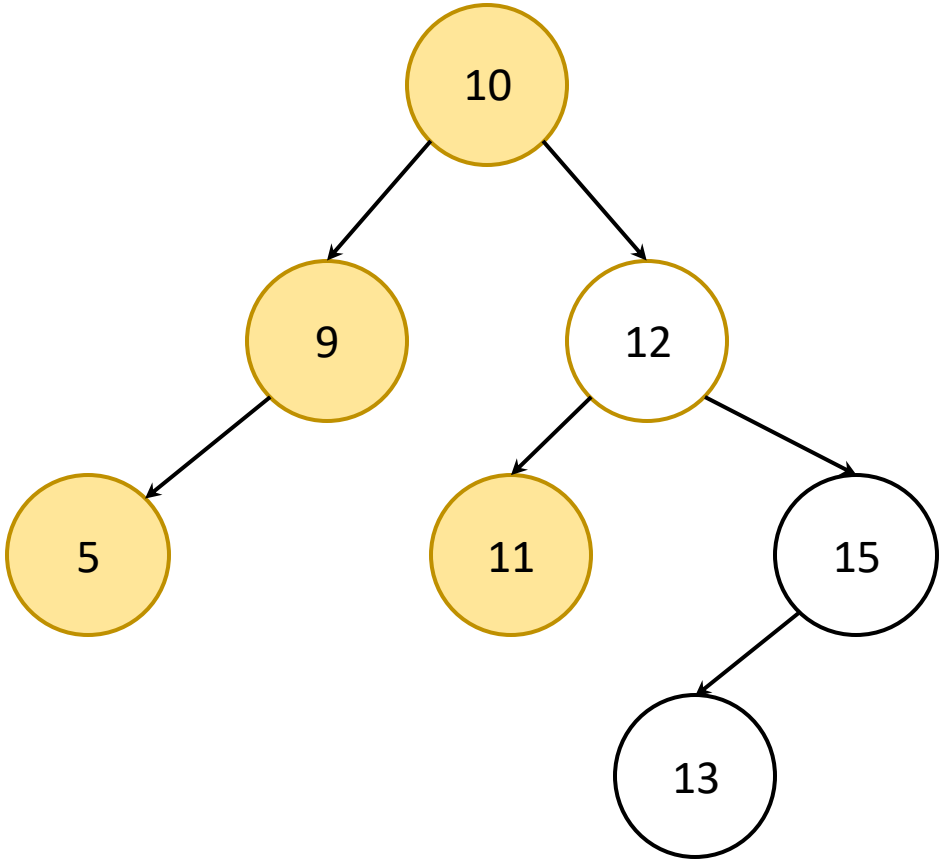
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

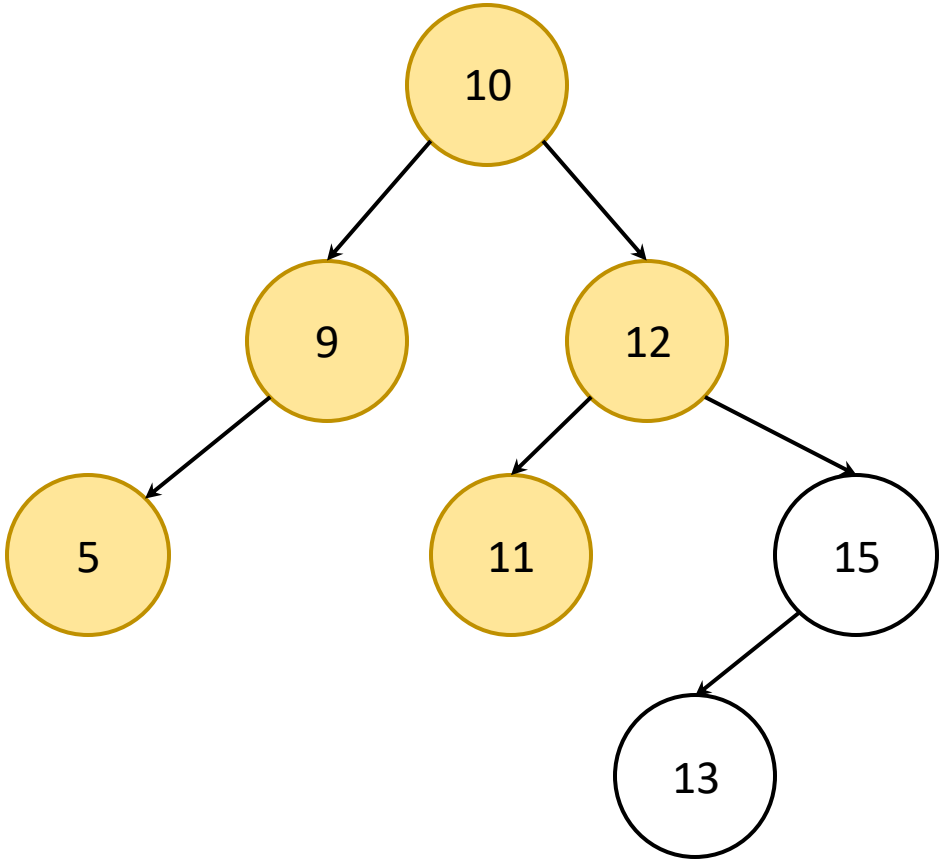
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

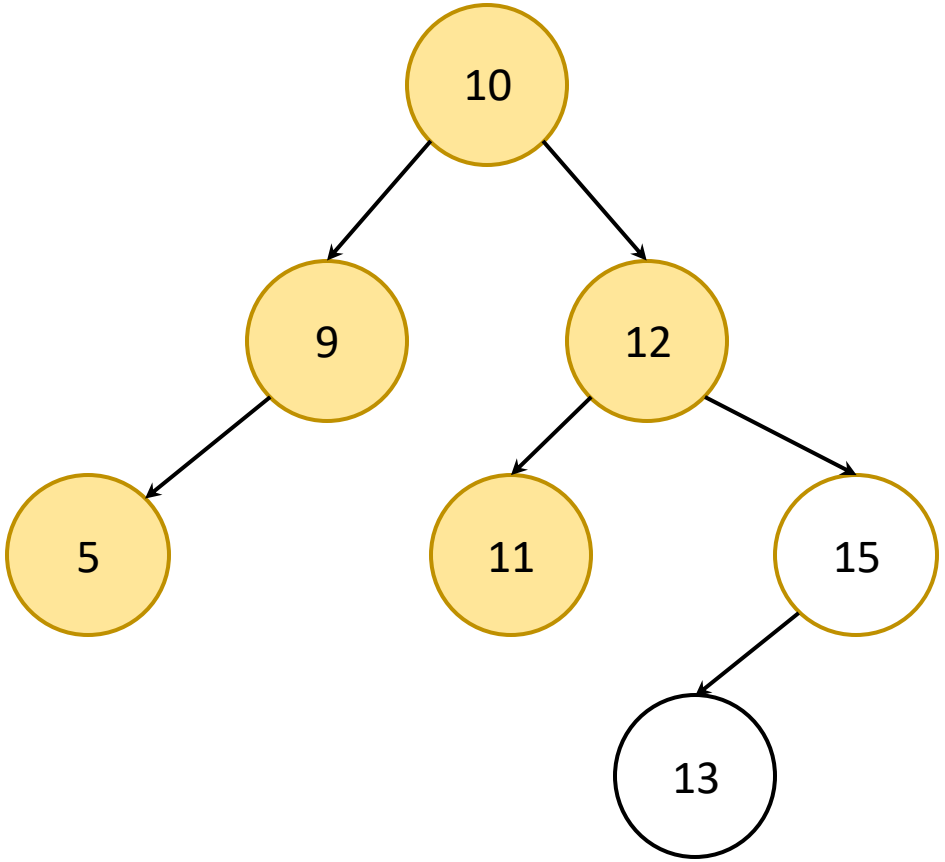
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

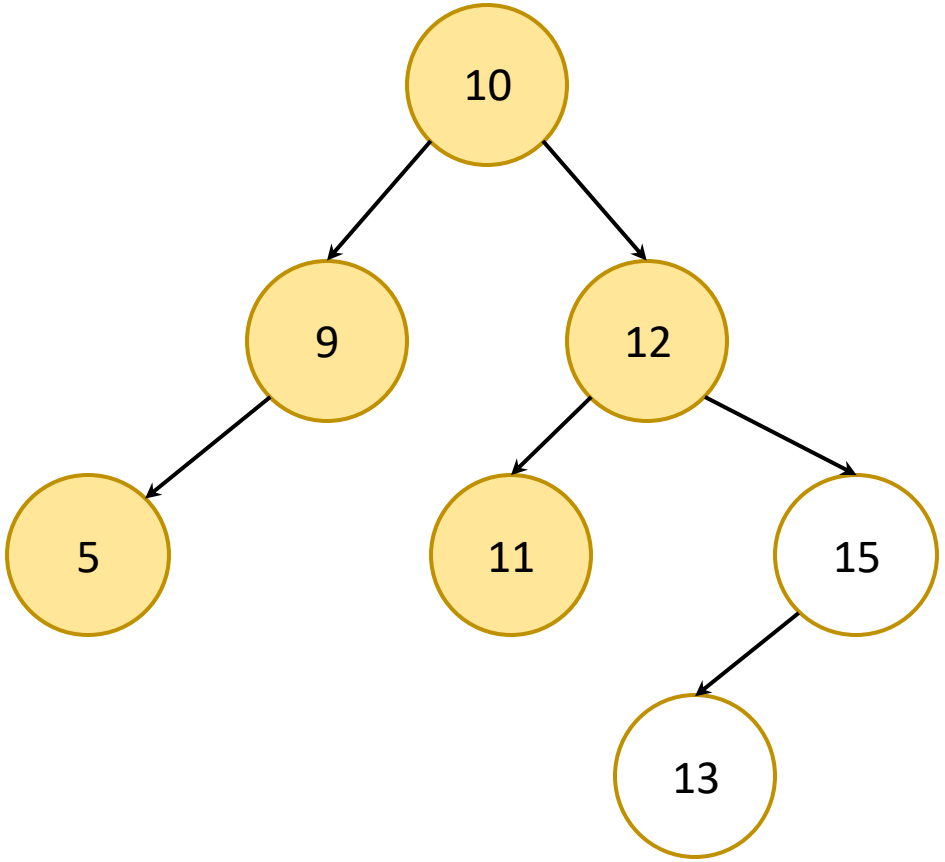
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

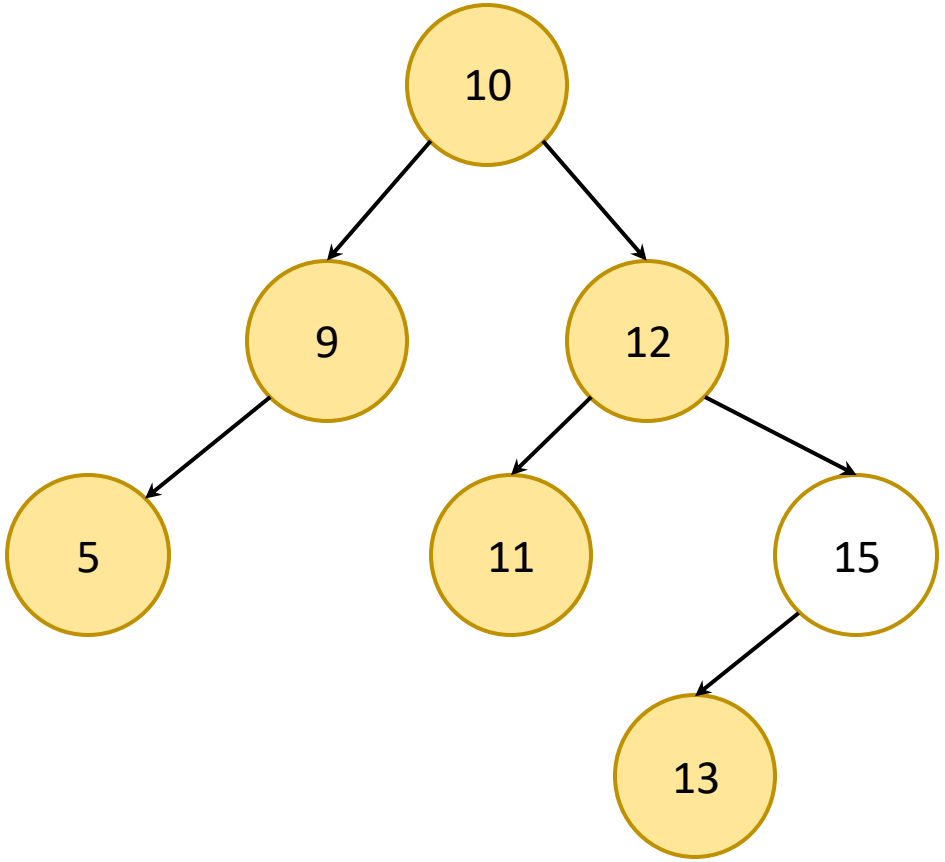
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

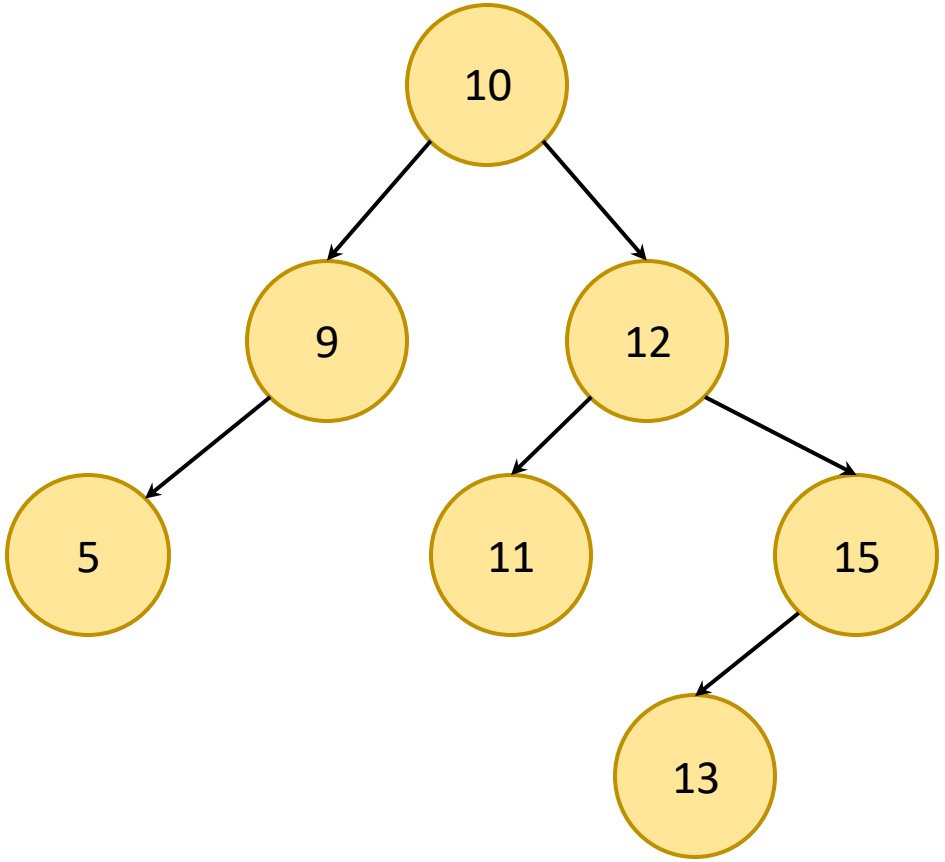
Traverse
In-Order



Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST traversal

Traverse
In-Order



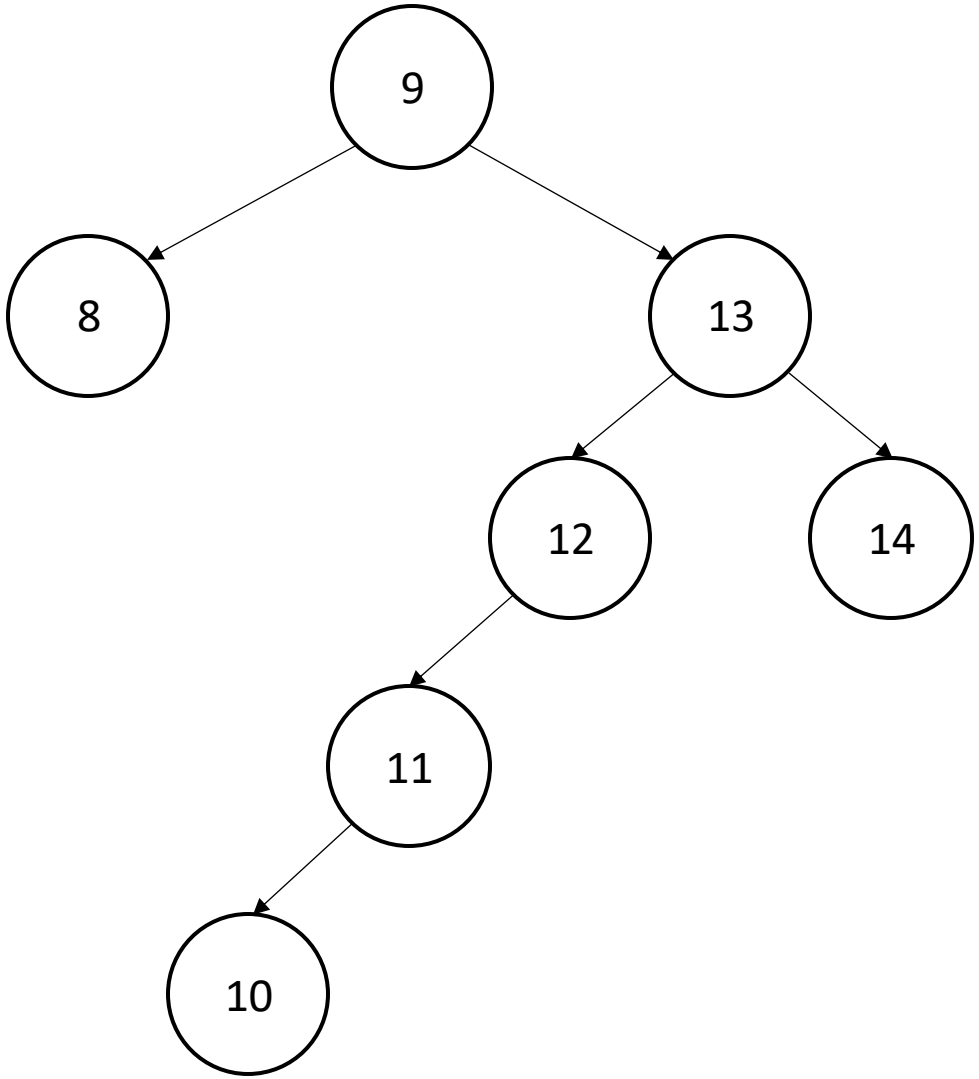
Call traverse on left subtree
Process the root node
Call traverse on right subtree

BST Balancing

- If the height of a left subtree is as near equal as possible to the height of the right subtree of a node.
- This must be true for all descendants.
- If both of these conditions are true the tree is “balanced”.

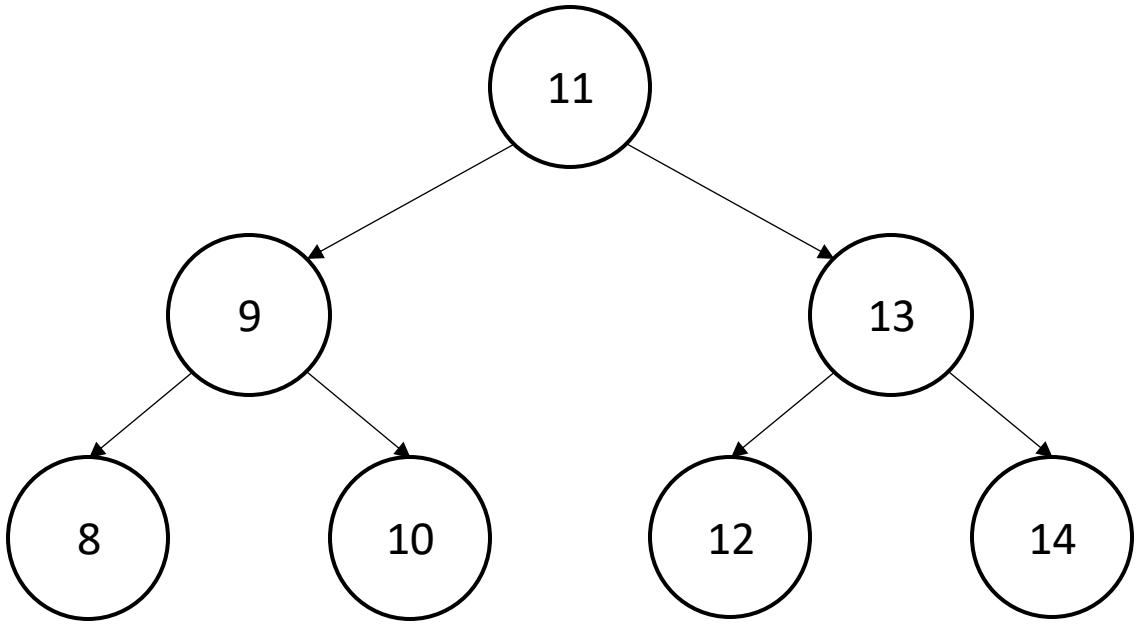
- A balanced search tree has, on average, logarithmic search times…
- … because when searching for a node, results in either finding what you’re looking for or halving the remaining search space.
- Balanced BST with n nodes are proportionate to $log_2 n$.

Unbalanced BST

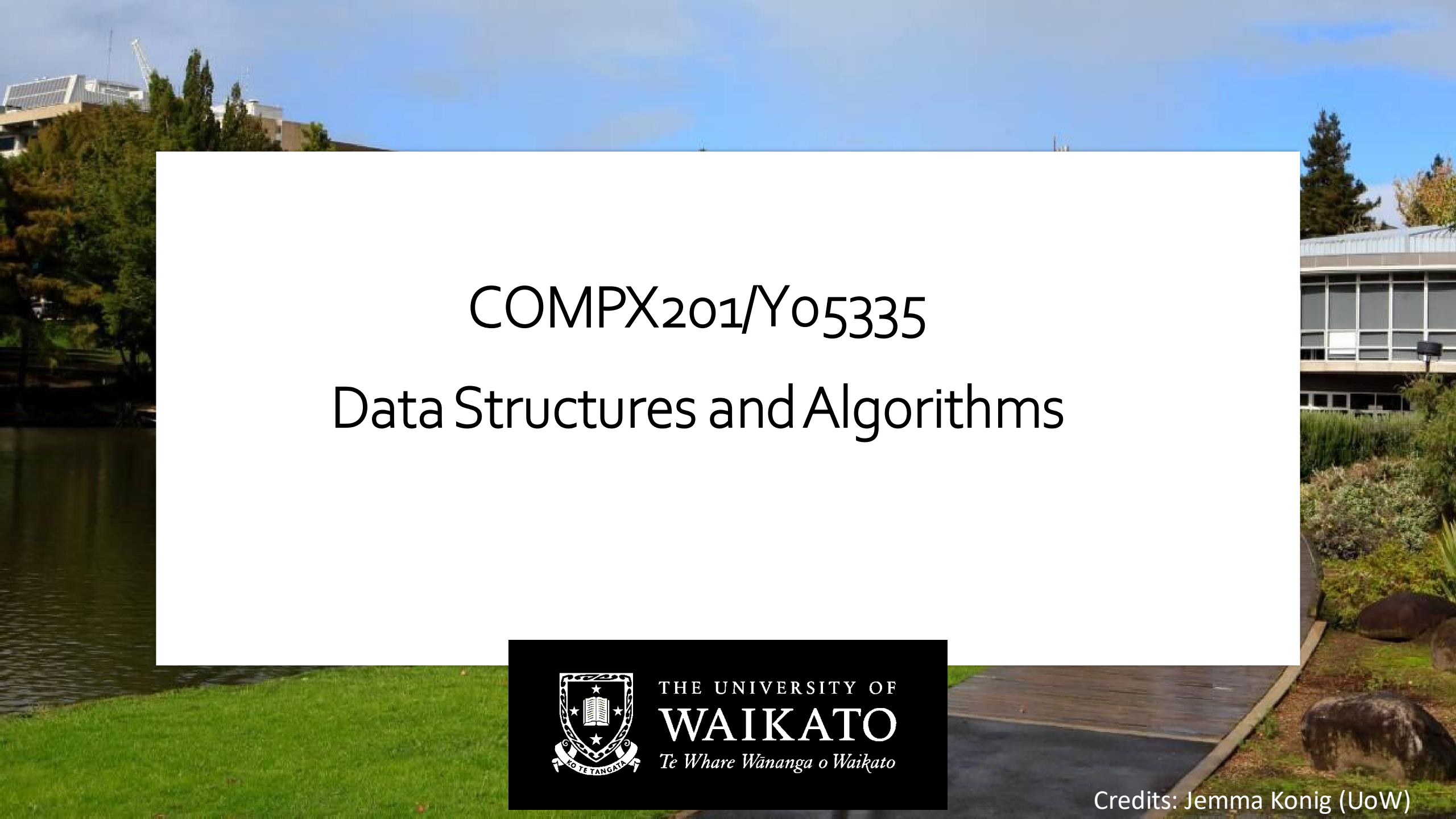


If the height of a left subtree is as near equal as possible to the height of the right subtree of a node.
This must be true for all descendants.

Balanced BST



If the height of a left subtree is as near equal as possible to the height of the right subtree of a node.
This must be true for all descendants.



COMPX201/Y05335

Data Structures and Algorithms

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Credits: Jemma Konig (UoW)