LECTURE 21

BST AND TREE TRAVERSALS

MCS 275 Spring 2021

Emily Dumas

LECTURE 21: BST AND TREE TRAVERSALS

Course bulletins:

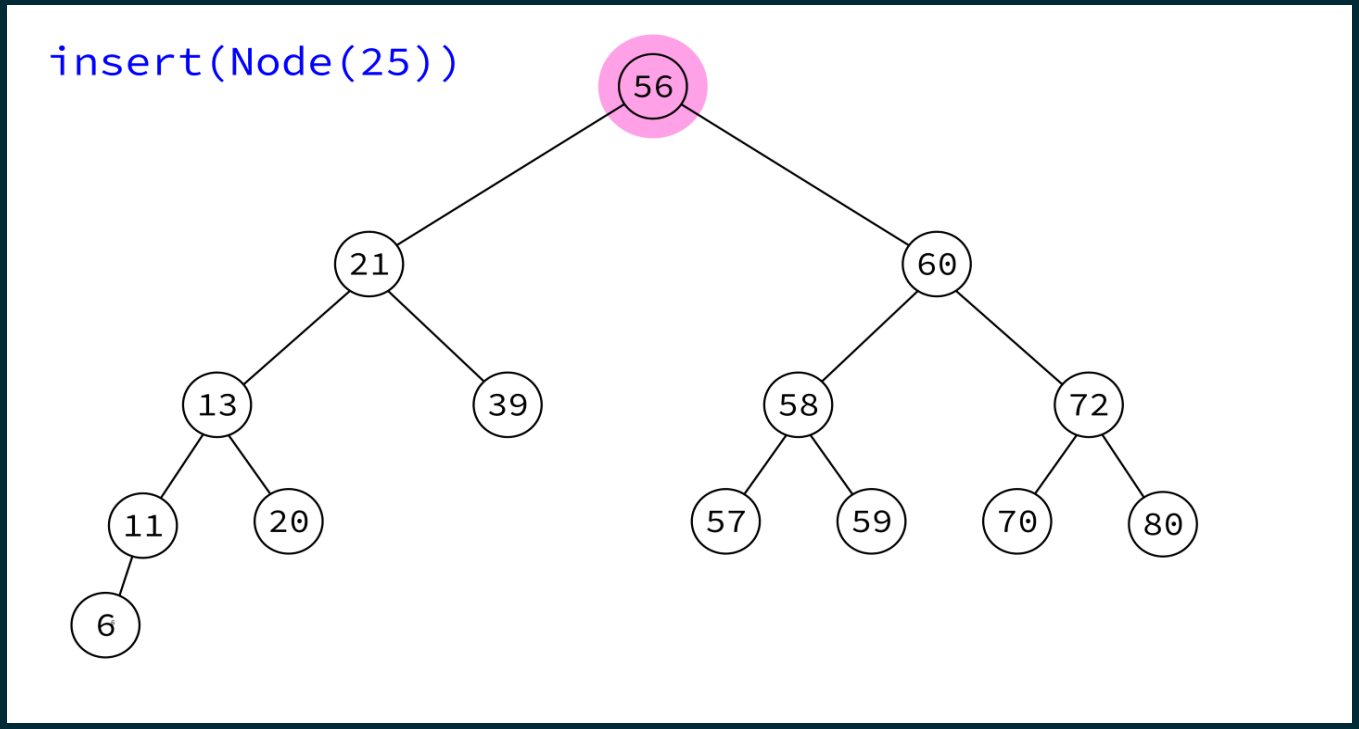
- Worksheet 8 available.
- Thursday discussion students: Please attempt problem 1 before discussion.

SAMPLE CODE

As a reminder, all the tree sample code is available.

INSERT

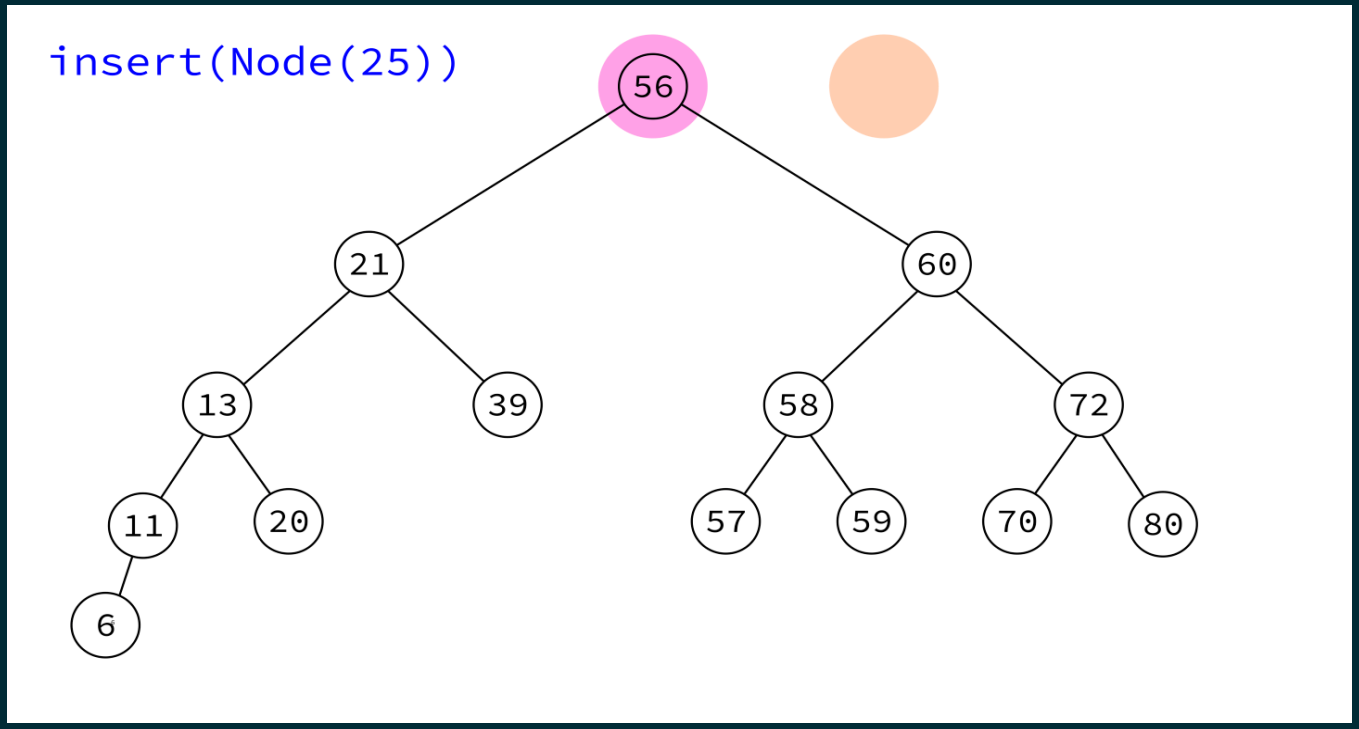
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

INSERT

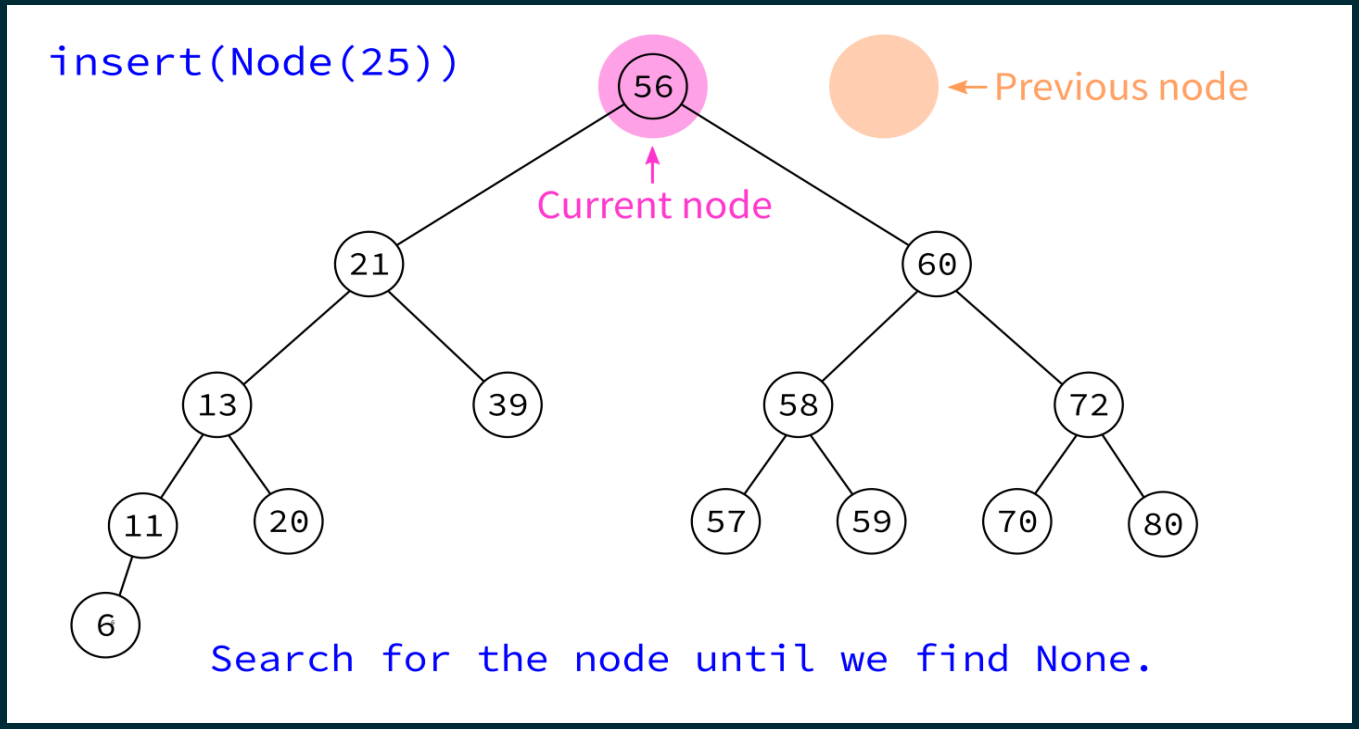
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

INSERT

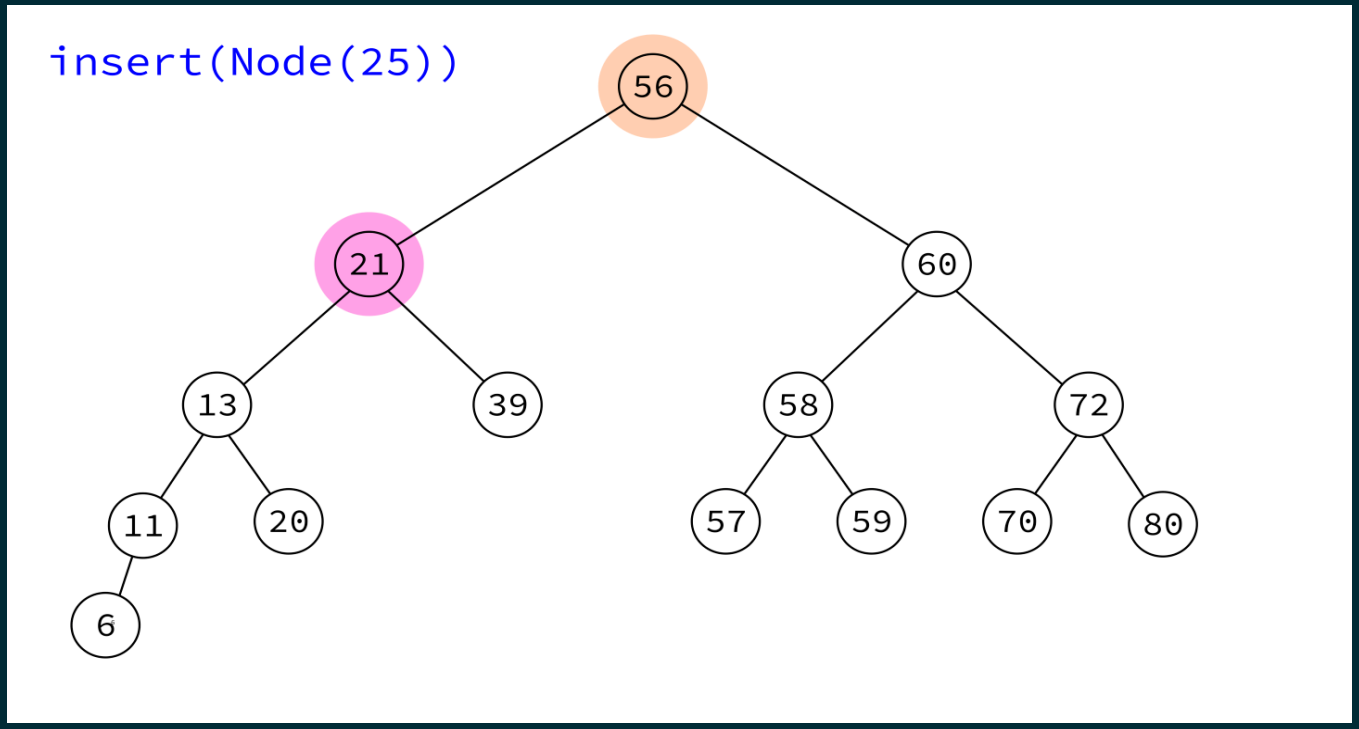
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

INSERT

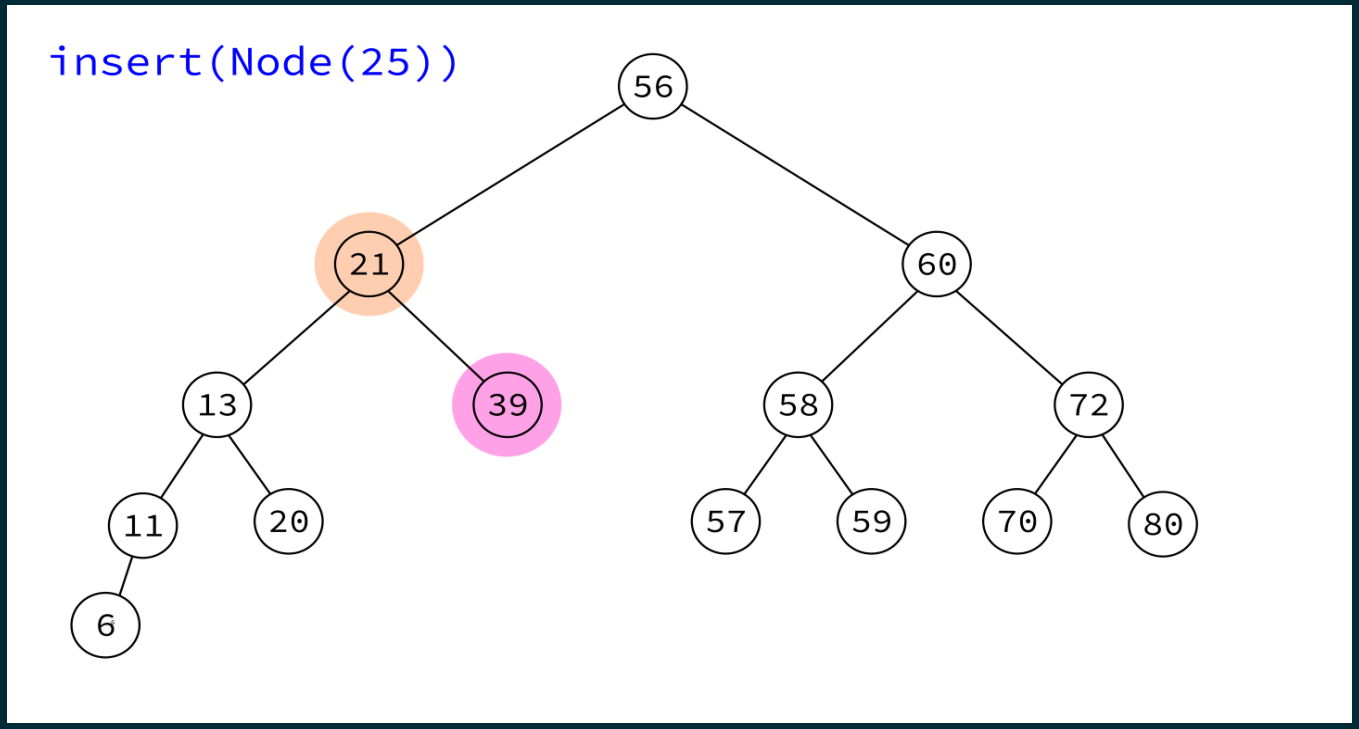
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

INSERT

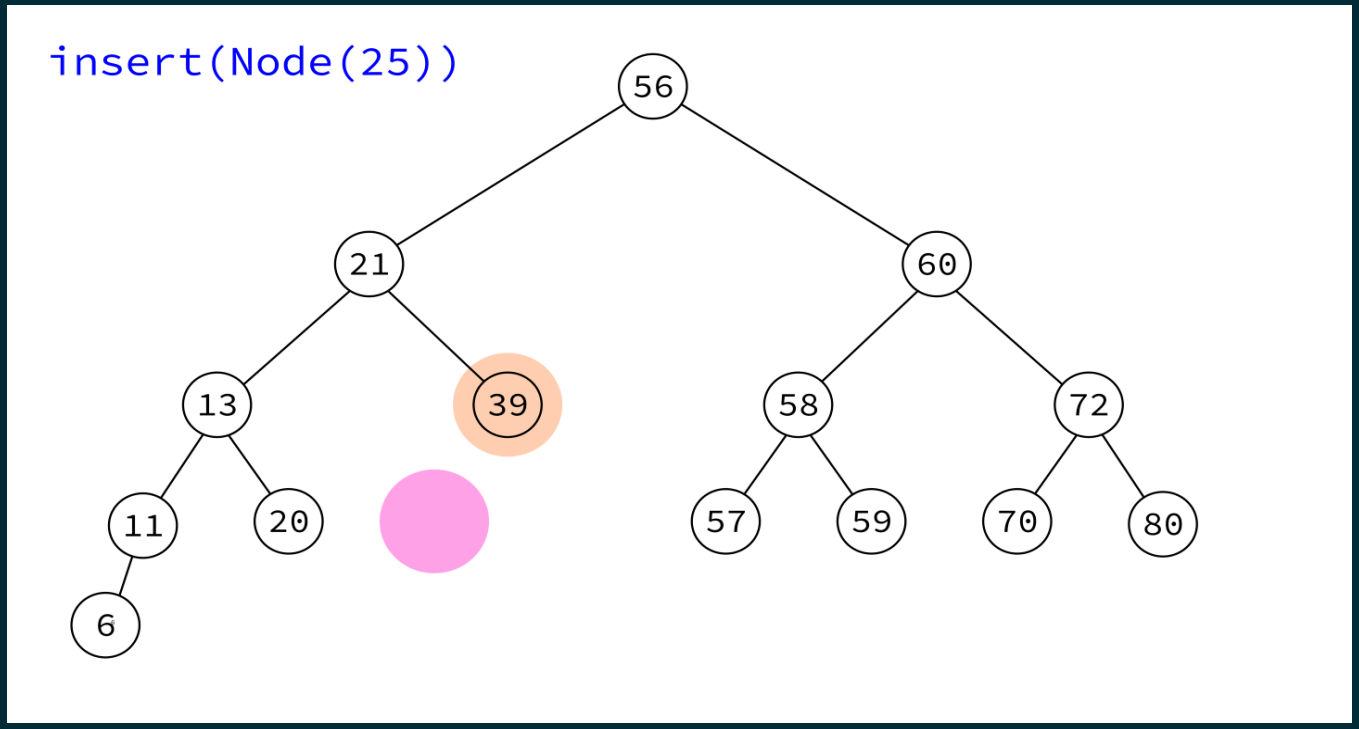
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

INSERT

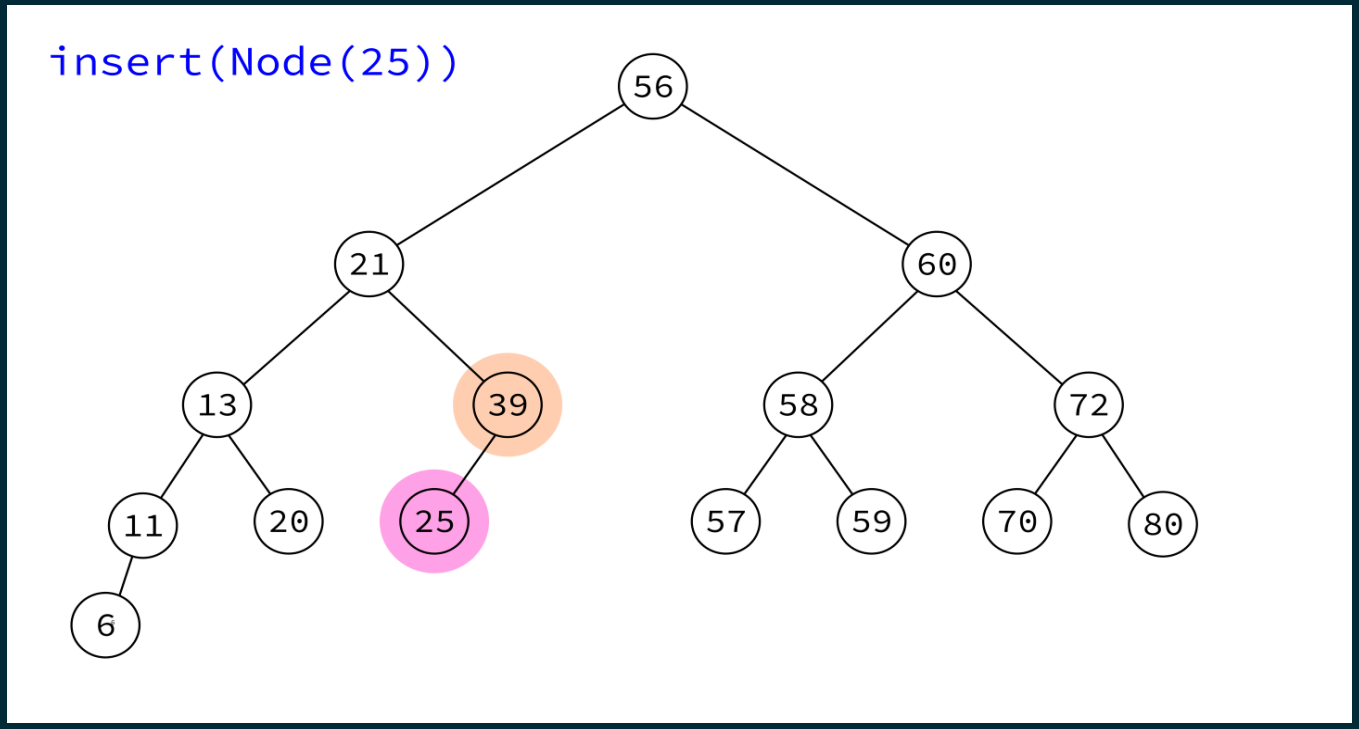
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

INSERT

Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

Algorithm `insert`:

Input: BST `T` and a node `x` whose key is set, but which has no parent or children.

Goal: Add `x` to `T`, maintaining the BST property.

1. If `T.root` is `None`, make `x` the root and return.
2. Otherwise, initialize variables `prev=None` and `cur=T.root`.
3. Descend into the tree as if searching for `x.key`, using `cur` to keep track of the current node and `prev` the last one visited, continuing until `cur` is `None`.
4. Make `x` a child of `prev` (choosing left or right as suits their keys).

INTEGERSET

As a sample application of BST, we can make a class that stores a set of integers, supporting membership testing and adding new elements.

Compare alternatives:

- Unsorted list - fast to insert, but slow membership test
- Sorted list - fast membership test, slow insert

IMPLEMENTATION HIDING

To use `BST`, you need to know about and work with `Node` objects.

In contrast, `IntegerSet` has an interface based directly on the values to be stored. It hides the fact that its implementation uses a BST.

WALKING A TREE

Back to discussing binary trees (not necessarily BST).

For some purposes we need to visit every node in a tree and perform some action on them.

To do this is to **traverse** or **walk** the tree.

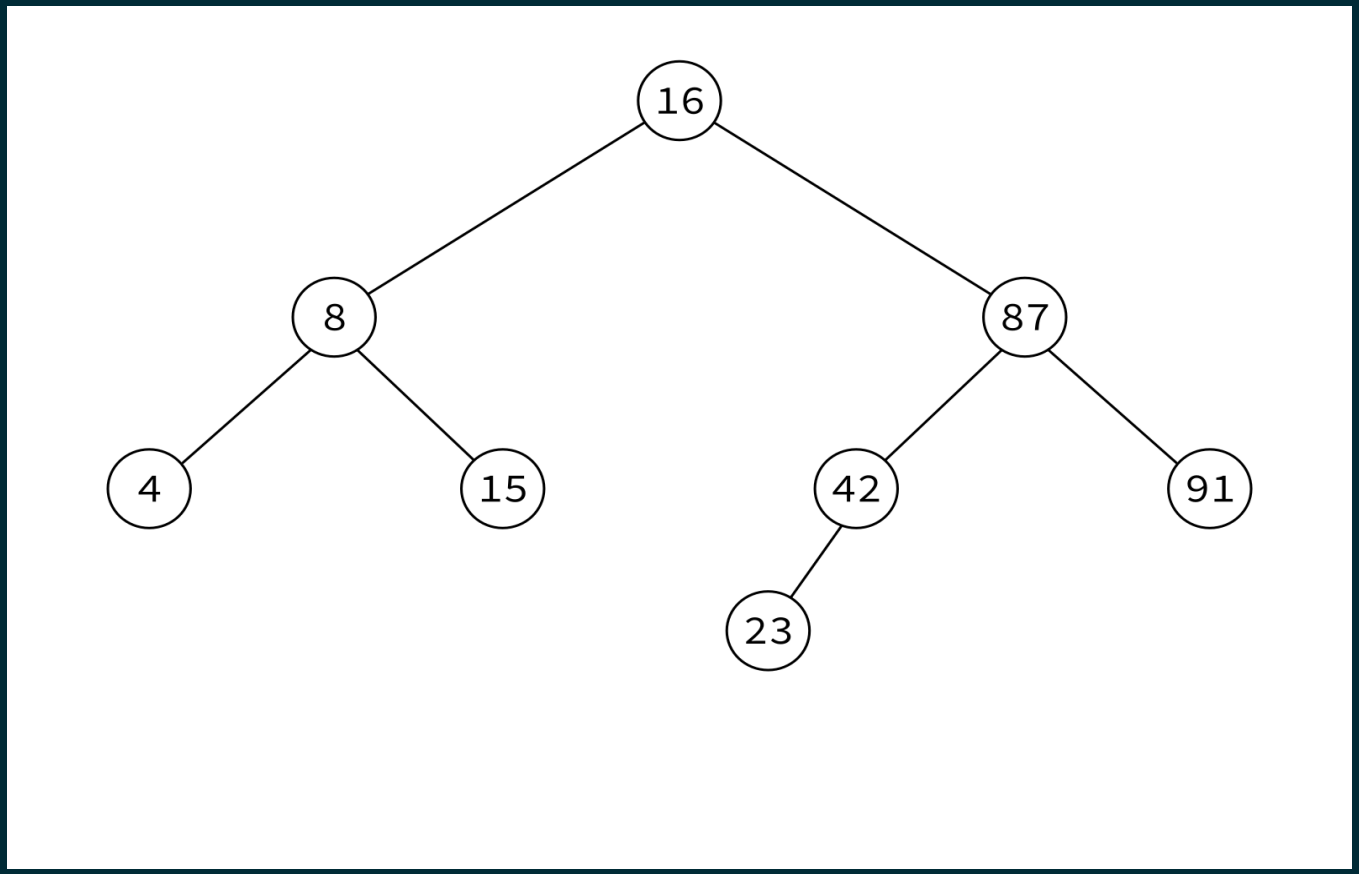
NAMED TRAVERSALS

The three most-often used recursive traversals:

- **preorder** - Node, left subtree, then right subtree.
- **postorder** - Left subtree, right subtree, then node.
- **inorder** - Left subtree, node, then right subtree.

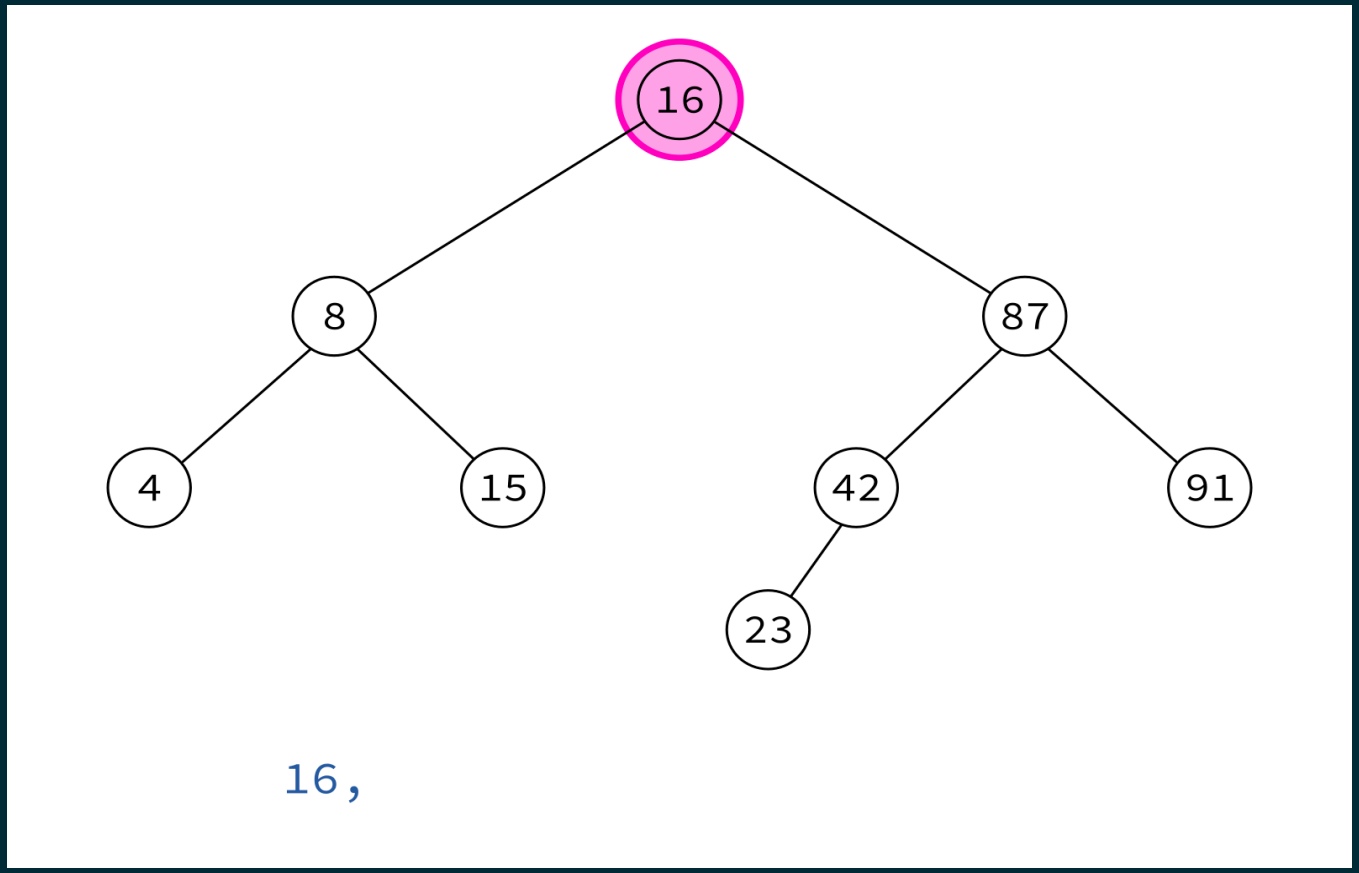
Note: They *all* visit left child before right child.

PREORDER TRAVERSAL



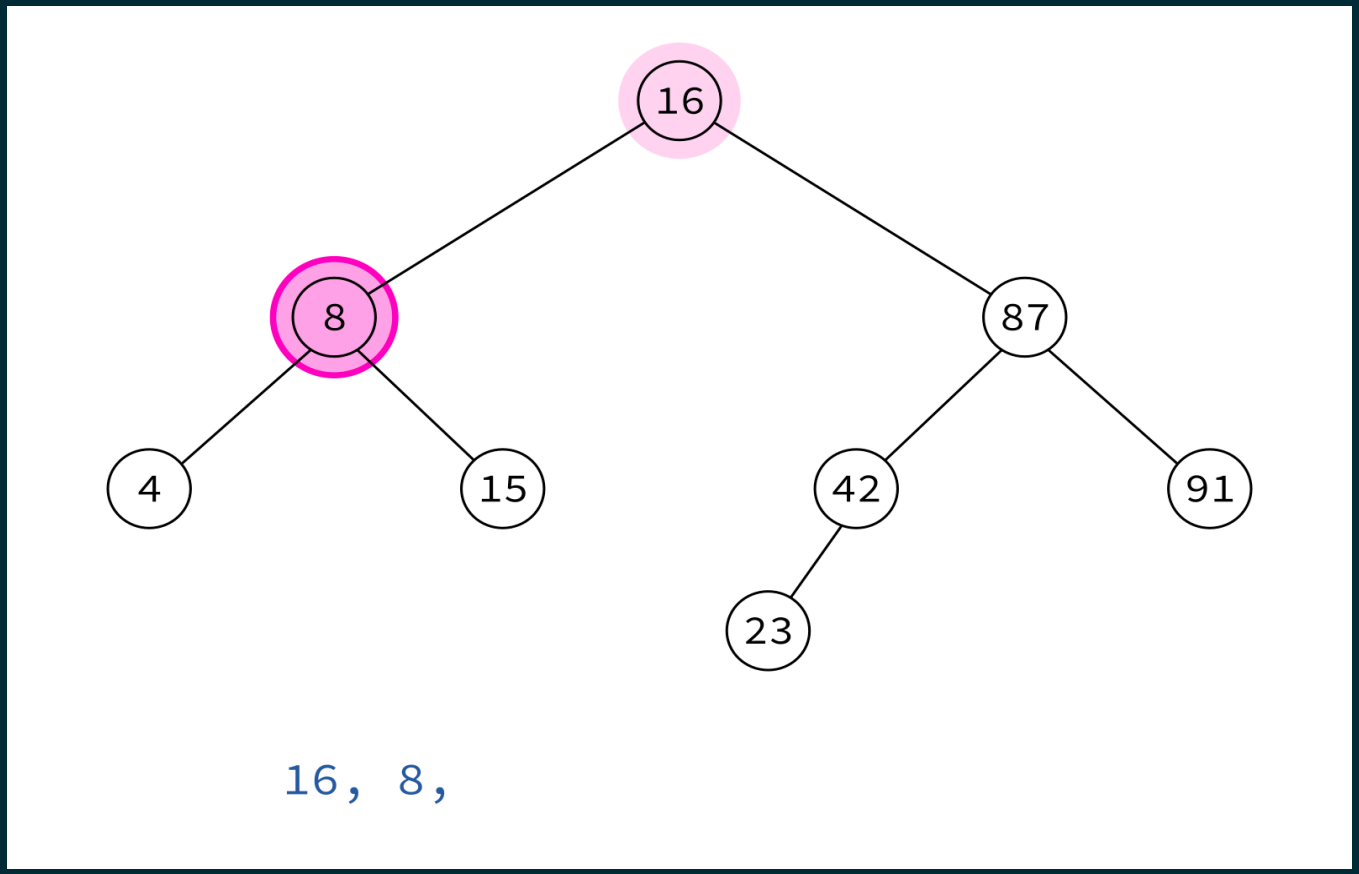
node, left, right

PREORDER TRAVERSAL



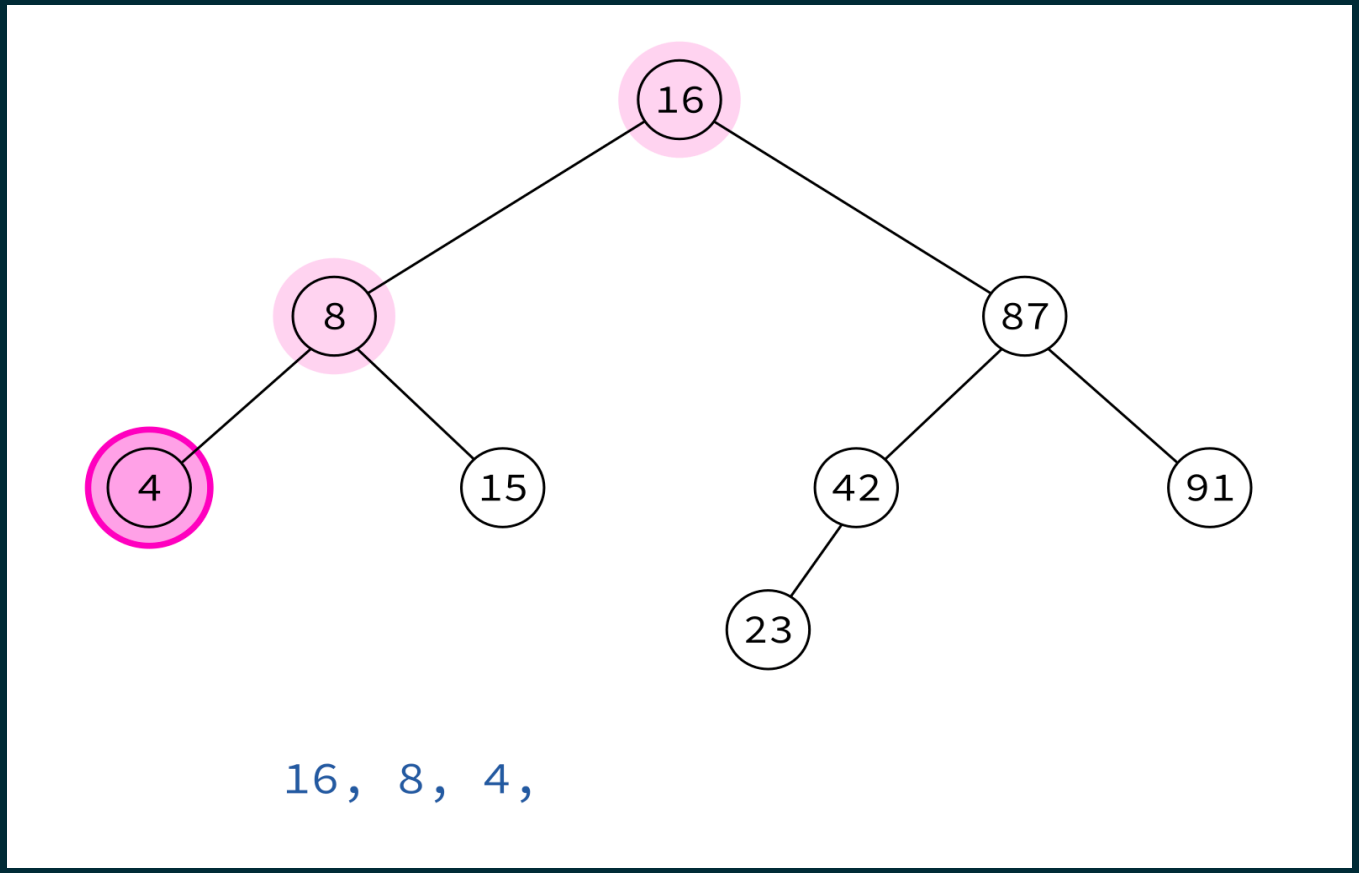
node, left, right

PREORDER TRAVERSAL



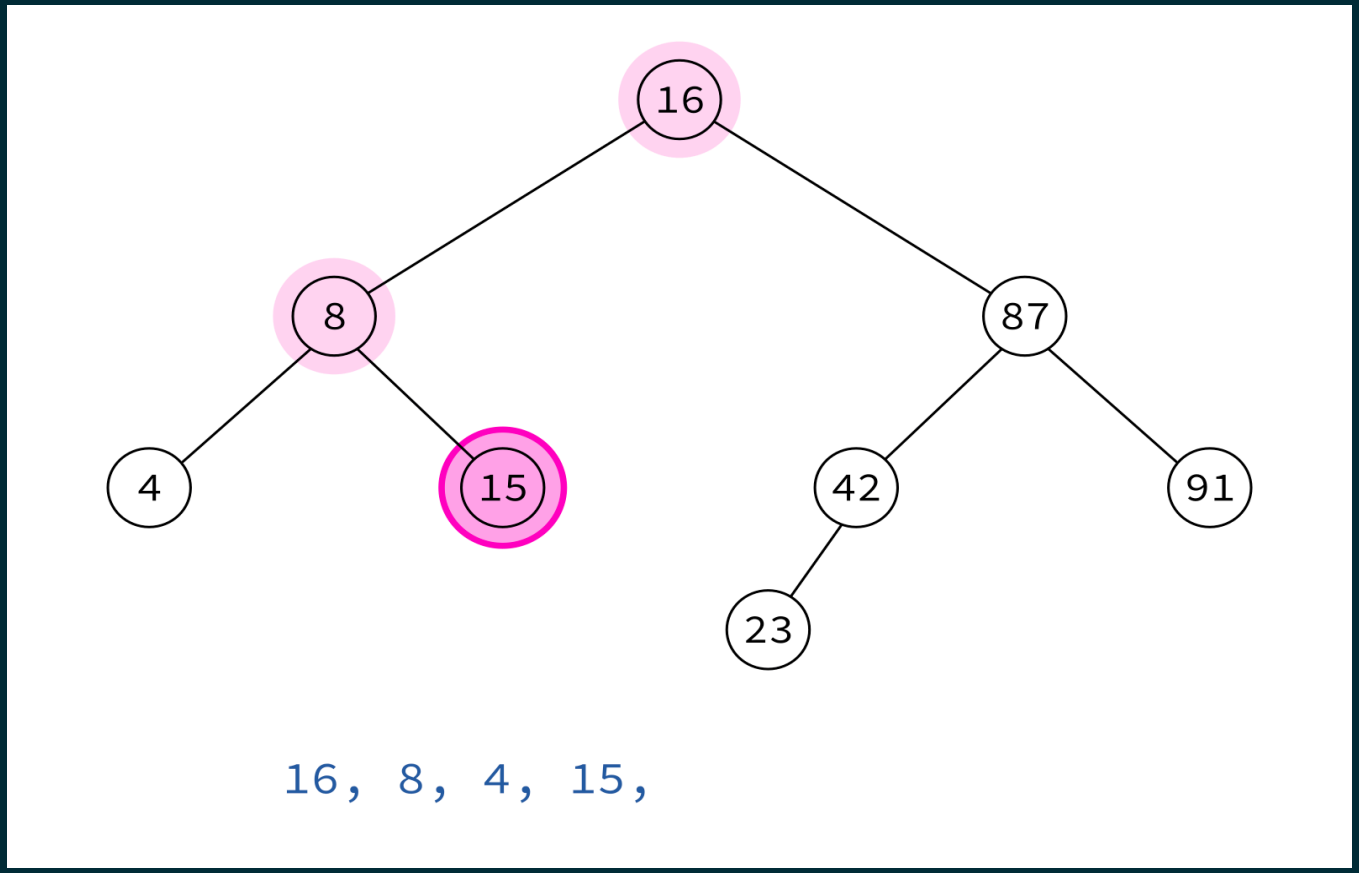
node, left, right

PREORDER TRAVERSAL



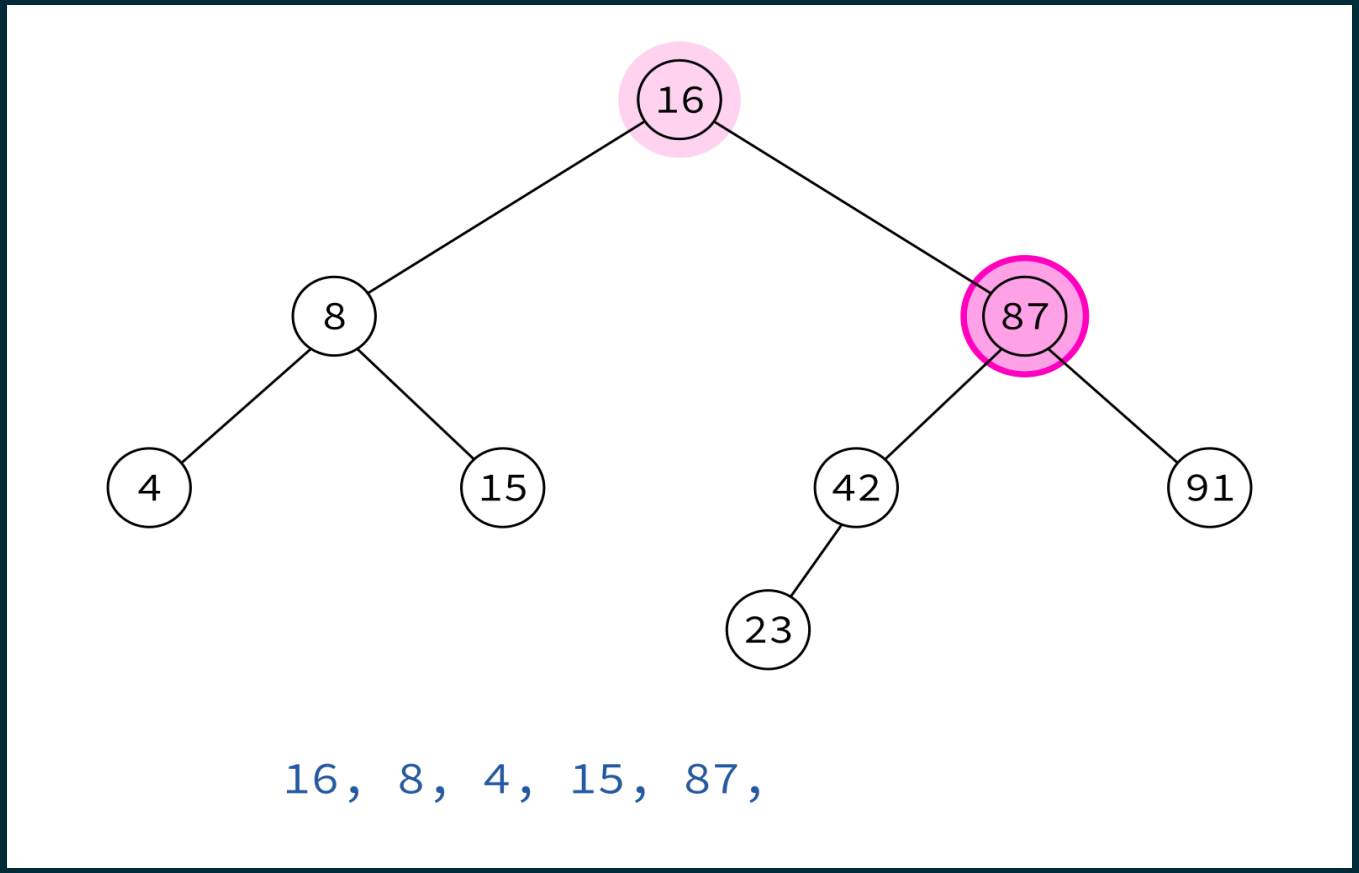
node, left, right

PREORDER TRAVERSAL



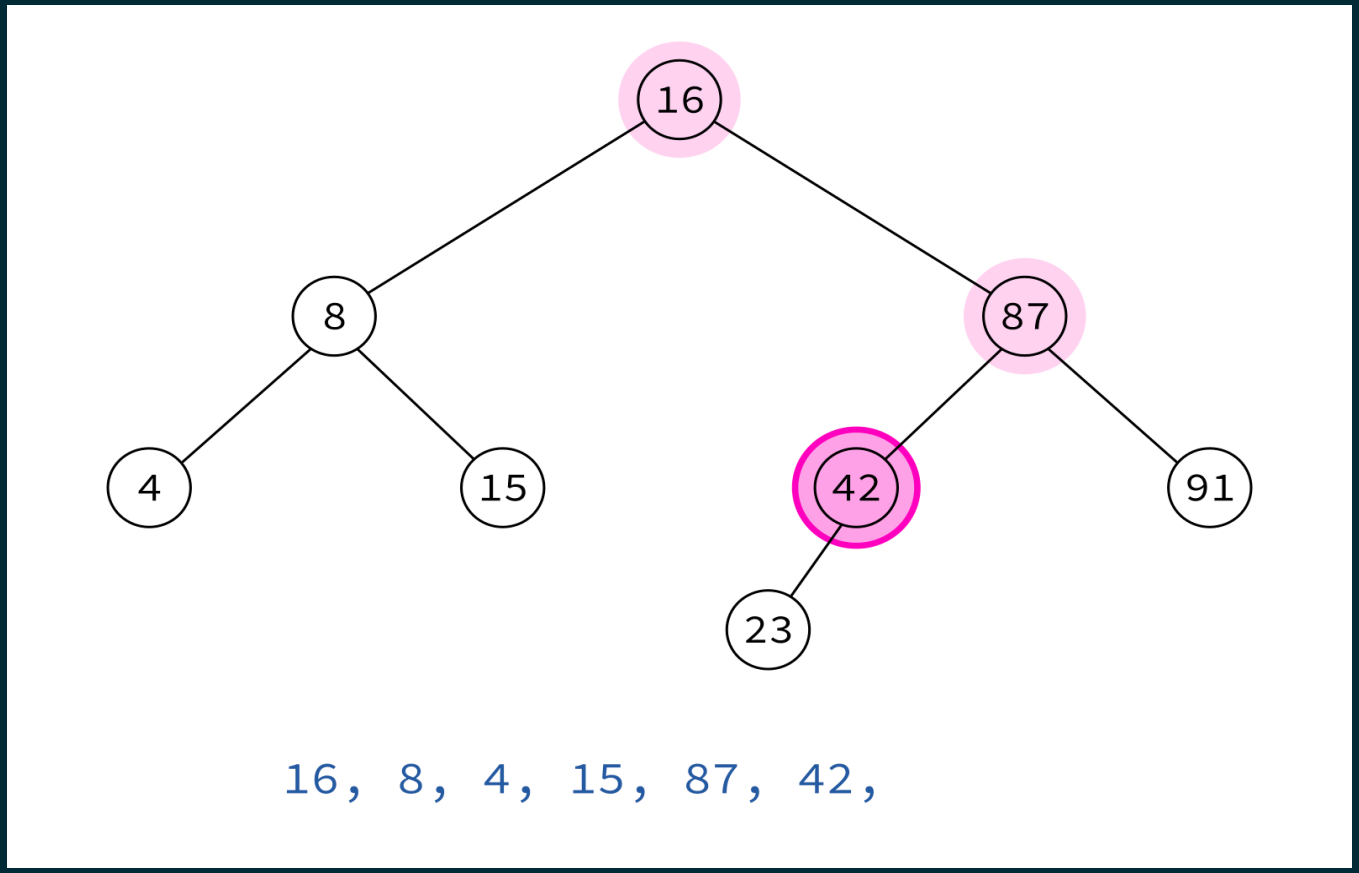
node, left, right

PREORDER TRAVERSAL



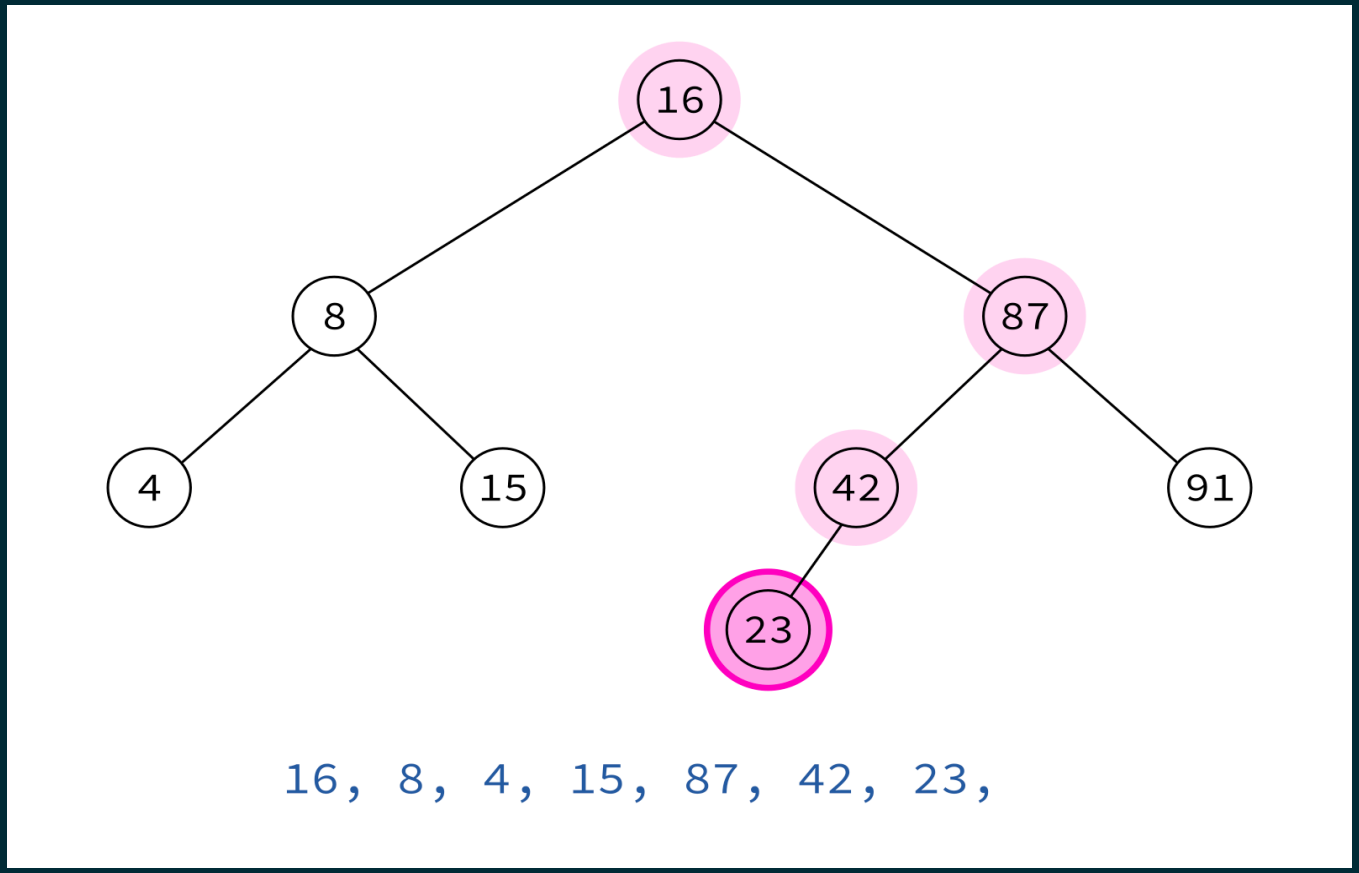
node, left, right

PREORDER TRAVERSAL



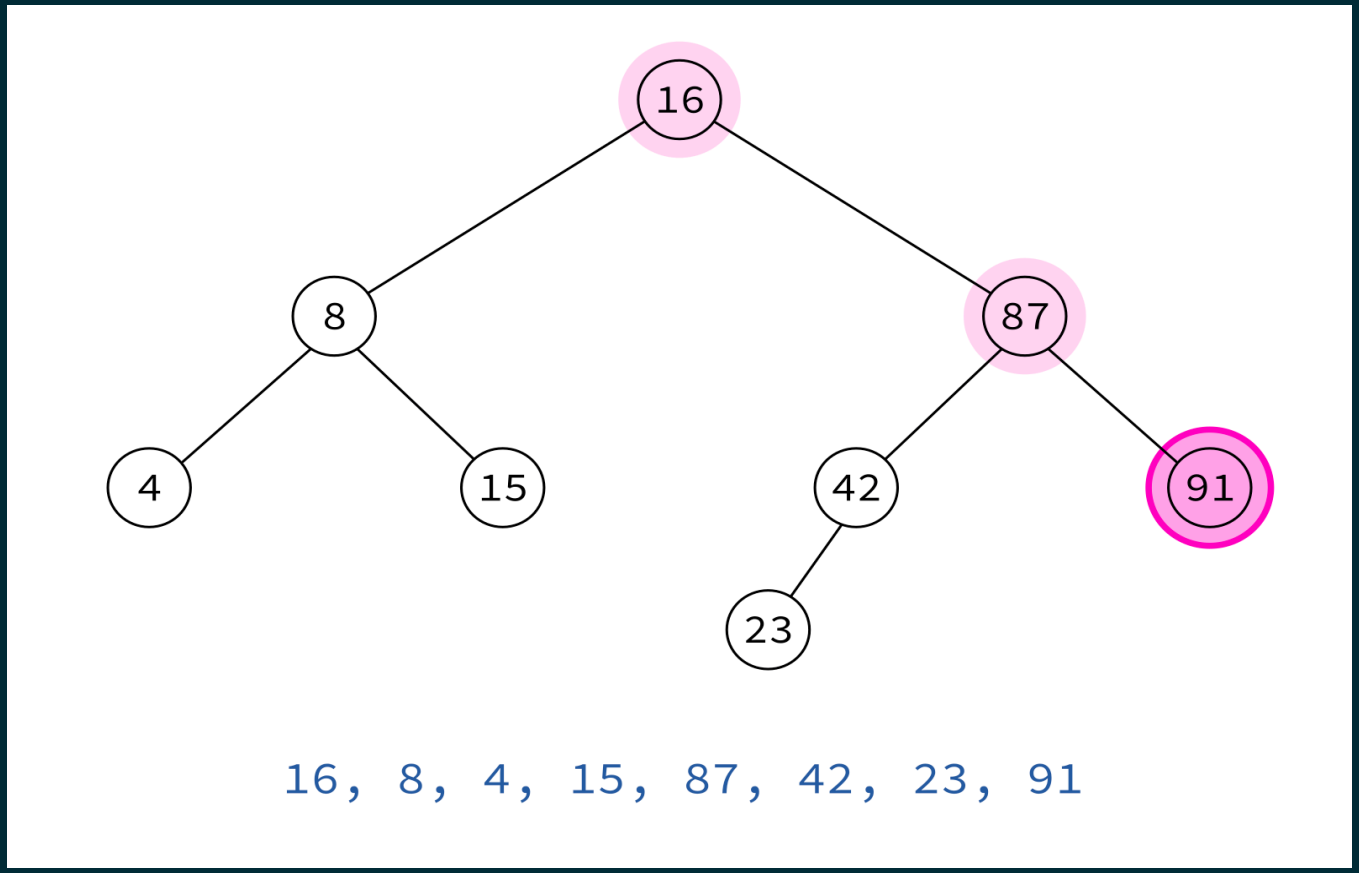
node, left, right

PREORDER TRAVERSAL



node, left, right

PREORDER TRAVERSAL



node, left, right

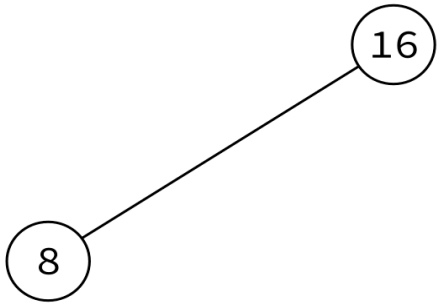
PREORDER TRAVERSAL

Typical use: Make a copy of the tree.

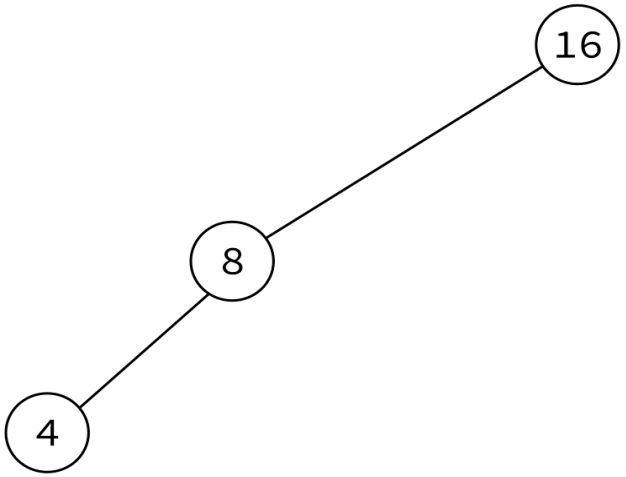
Insert the keys into an empty BST in this order to recreate the original tree.

16, 8, 4, 15, 87, 42, 23, 91

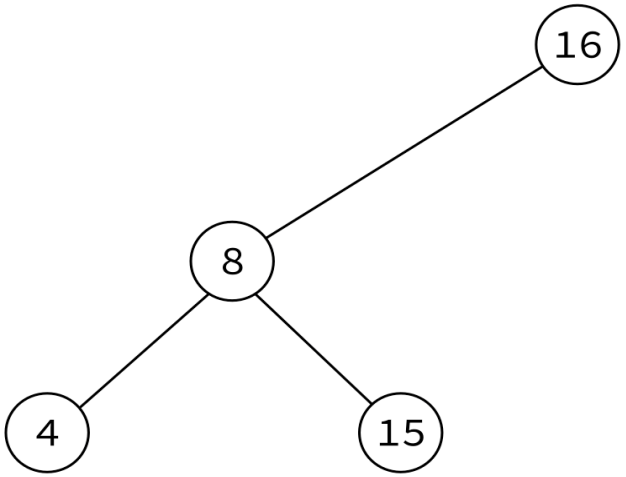
16, 8, 4, 15, 87, 42, 23, 91



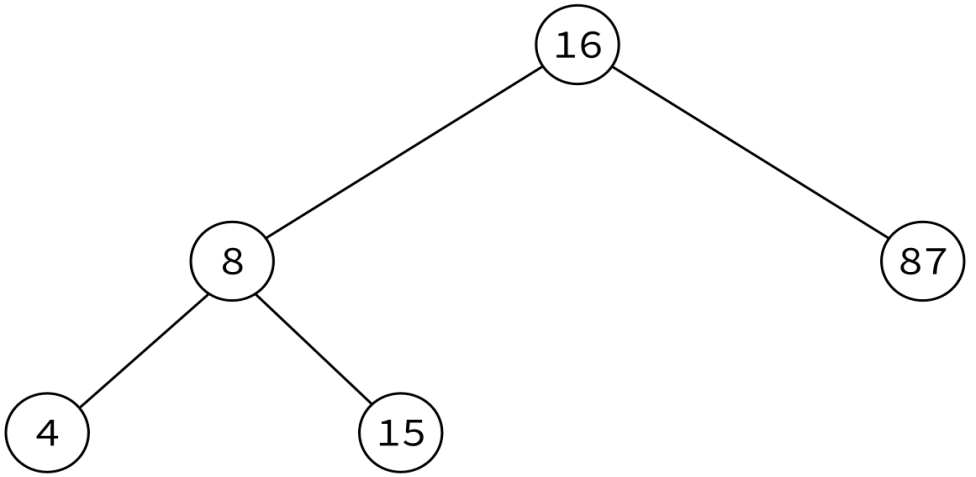
16, 8, 4, 15, 87, 42, 23, 91



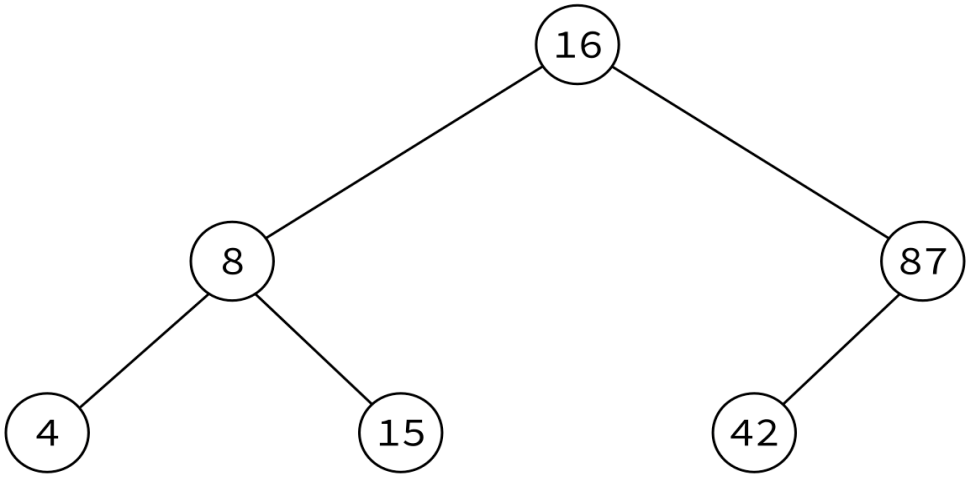
16, 8, 4, 15, 87, 42, 23, 91



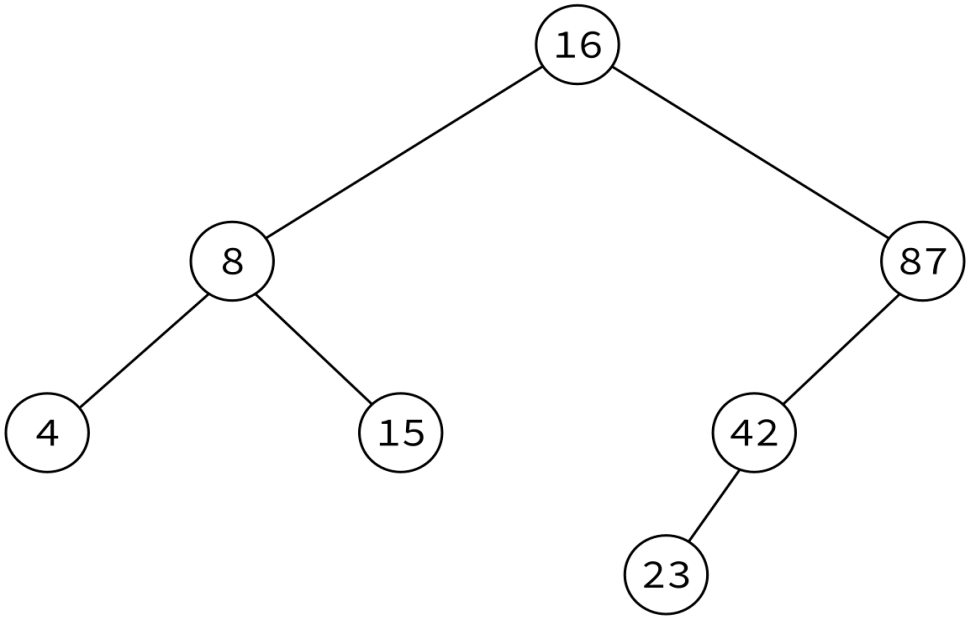
16, 8, 4, 15, 87, 42, 23, 91



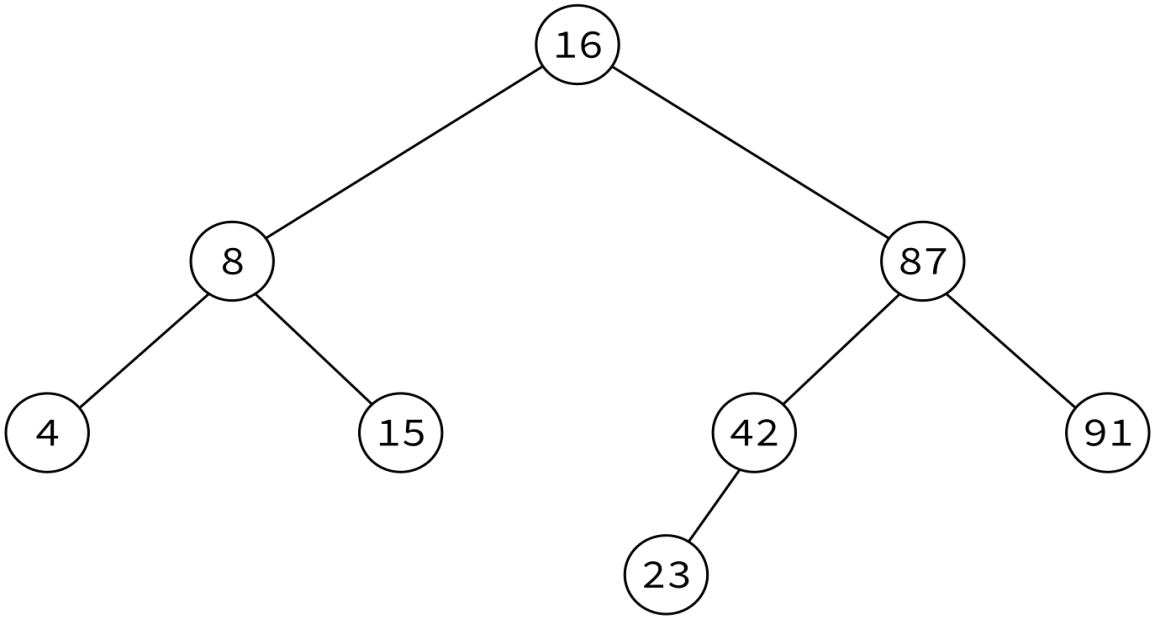
16, 8, 4, 15, 87, 42, 23, 91



16, 8, 4, 15, 87, 42, 23, 91

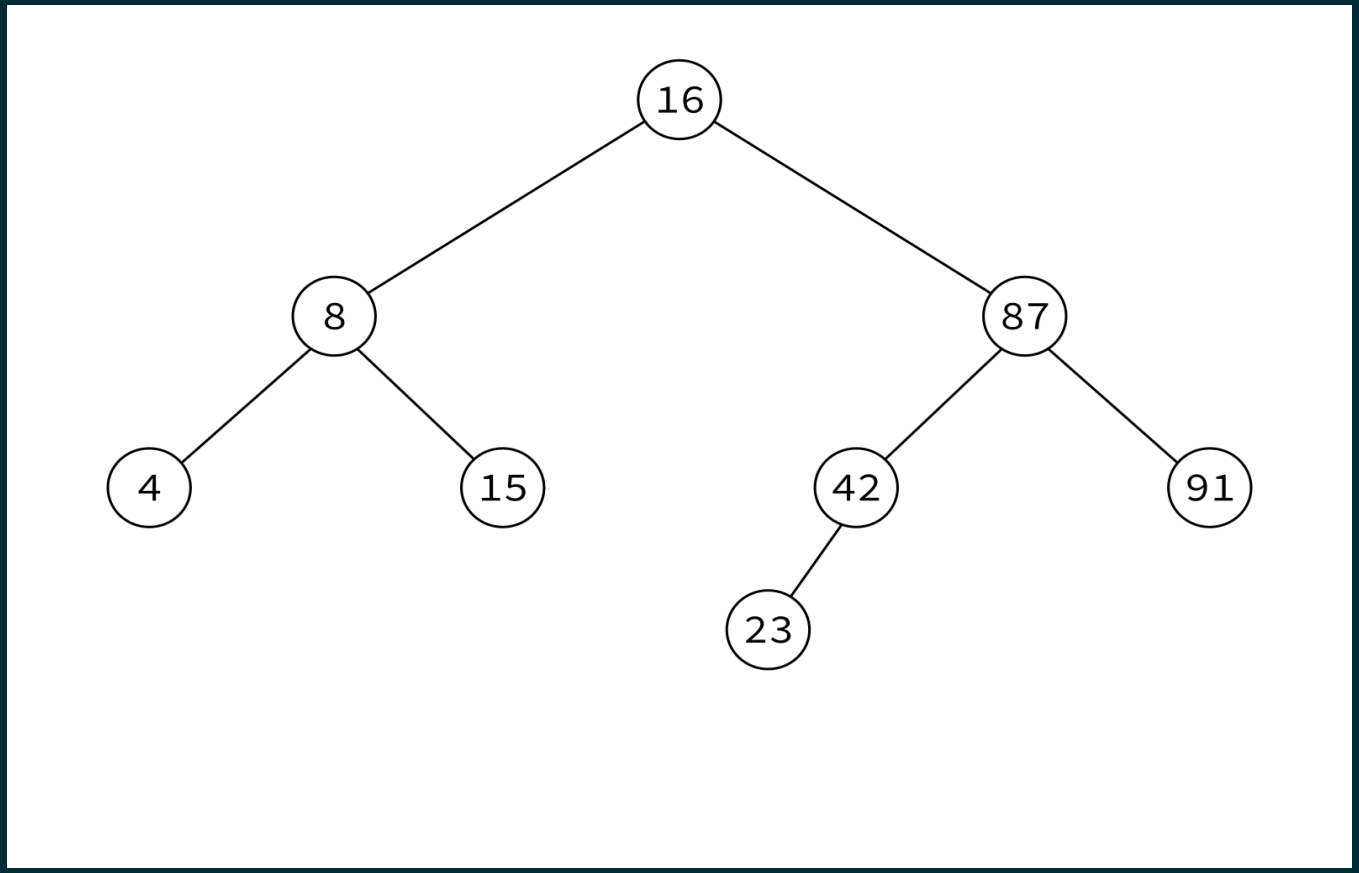


16, 8, 4, 15, 87, 42, 23, 91



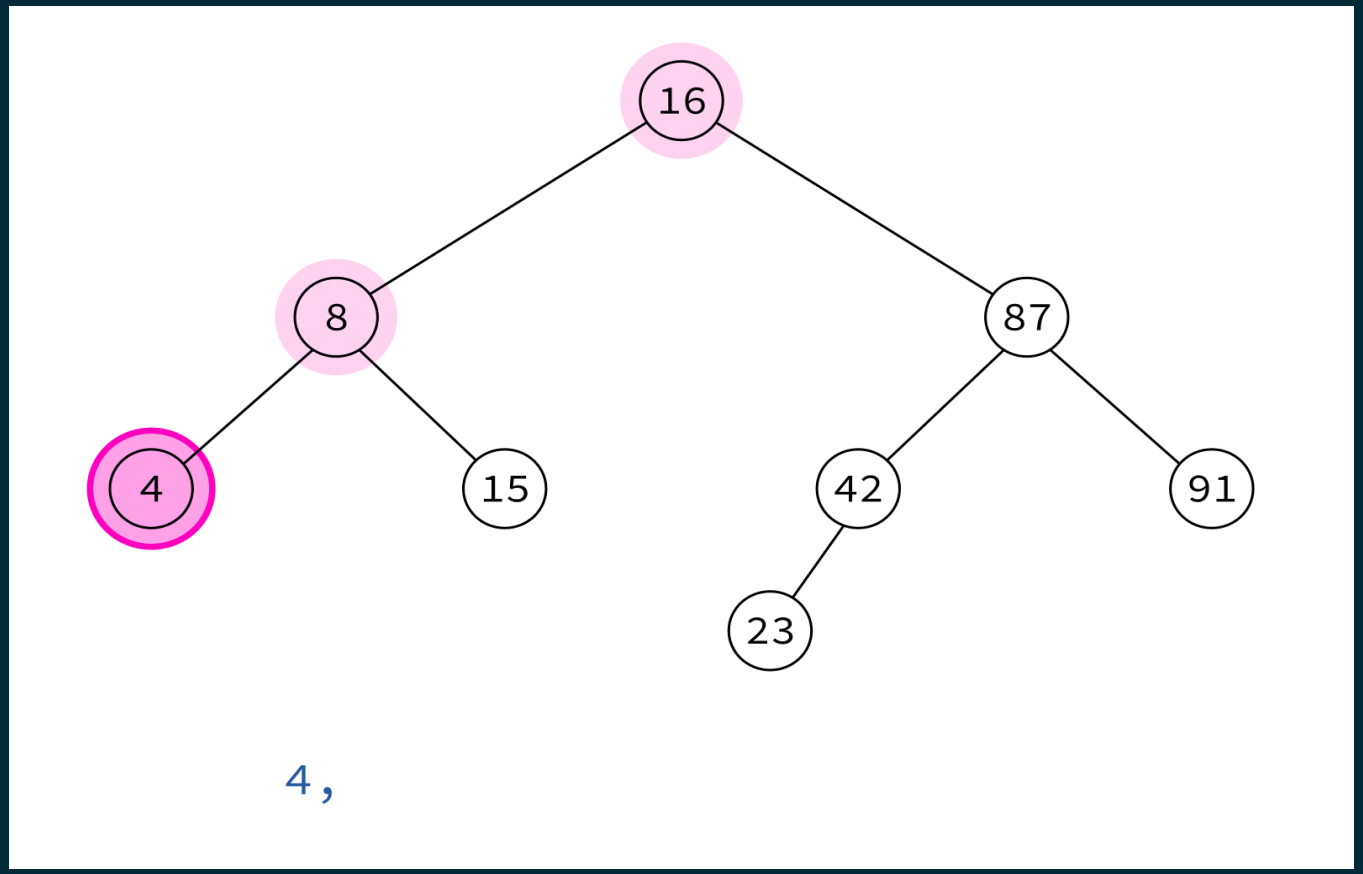
16, 8, 4, 15, 87, 42, 23, 91

POSTORDER TRAVERSAL



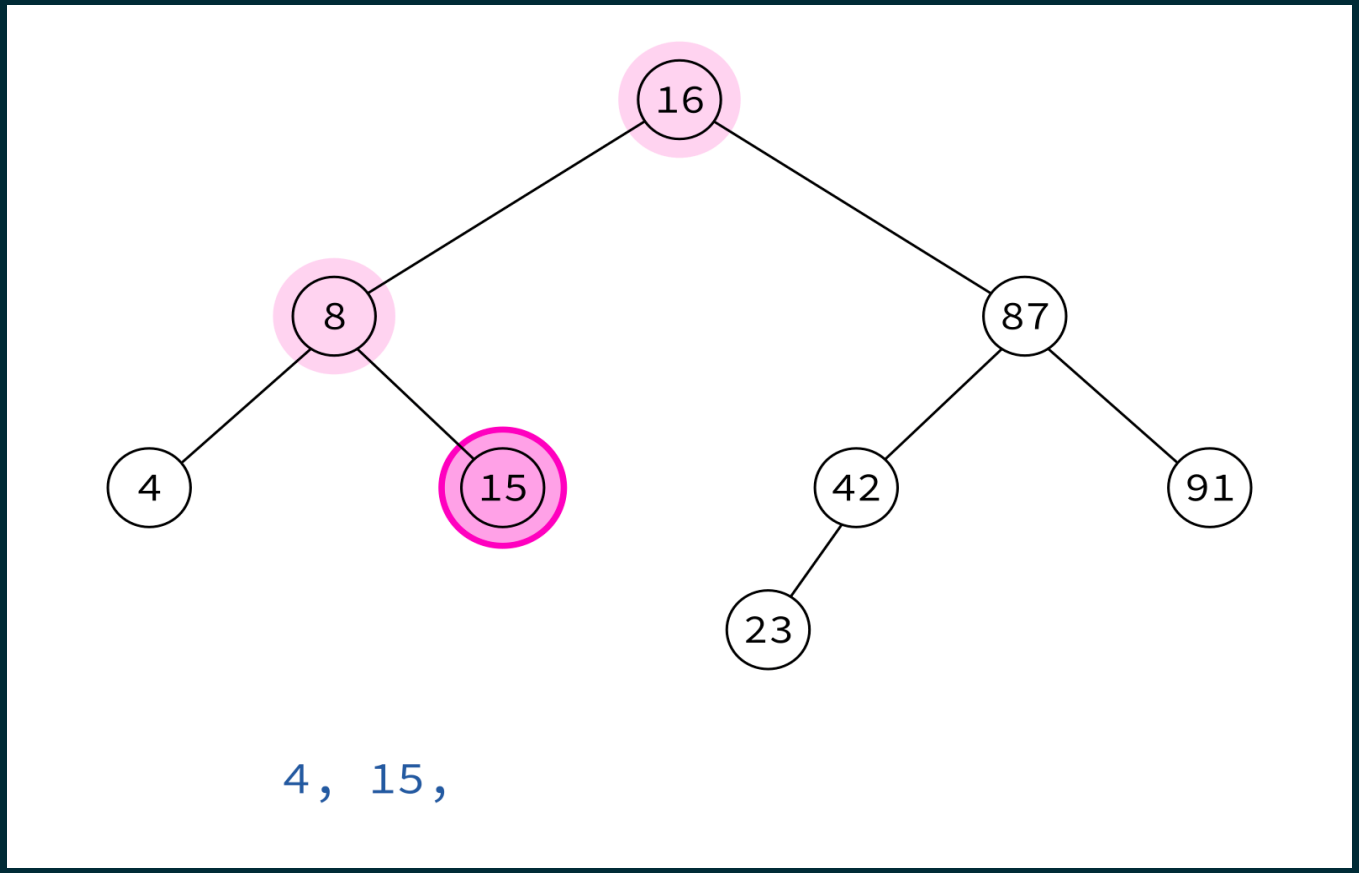
left, right, **node**

POSTORDER TRAVERSAL



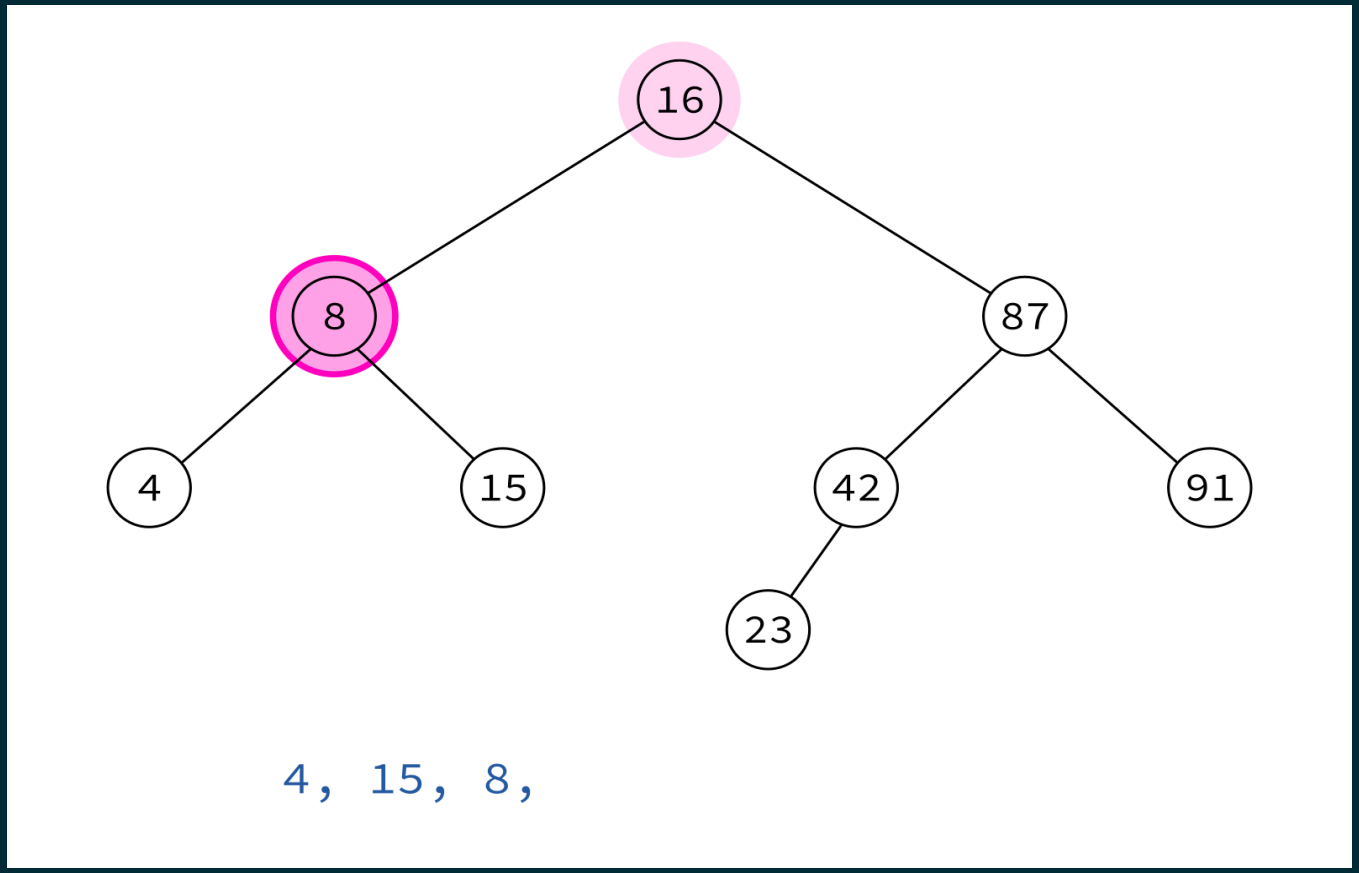
left, right, **node**

POSTORDER TRAVERSAL



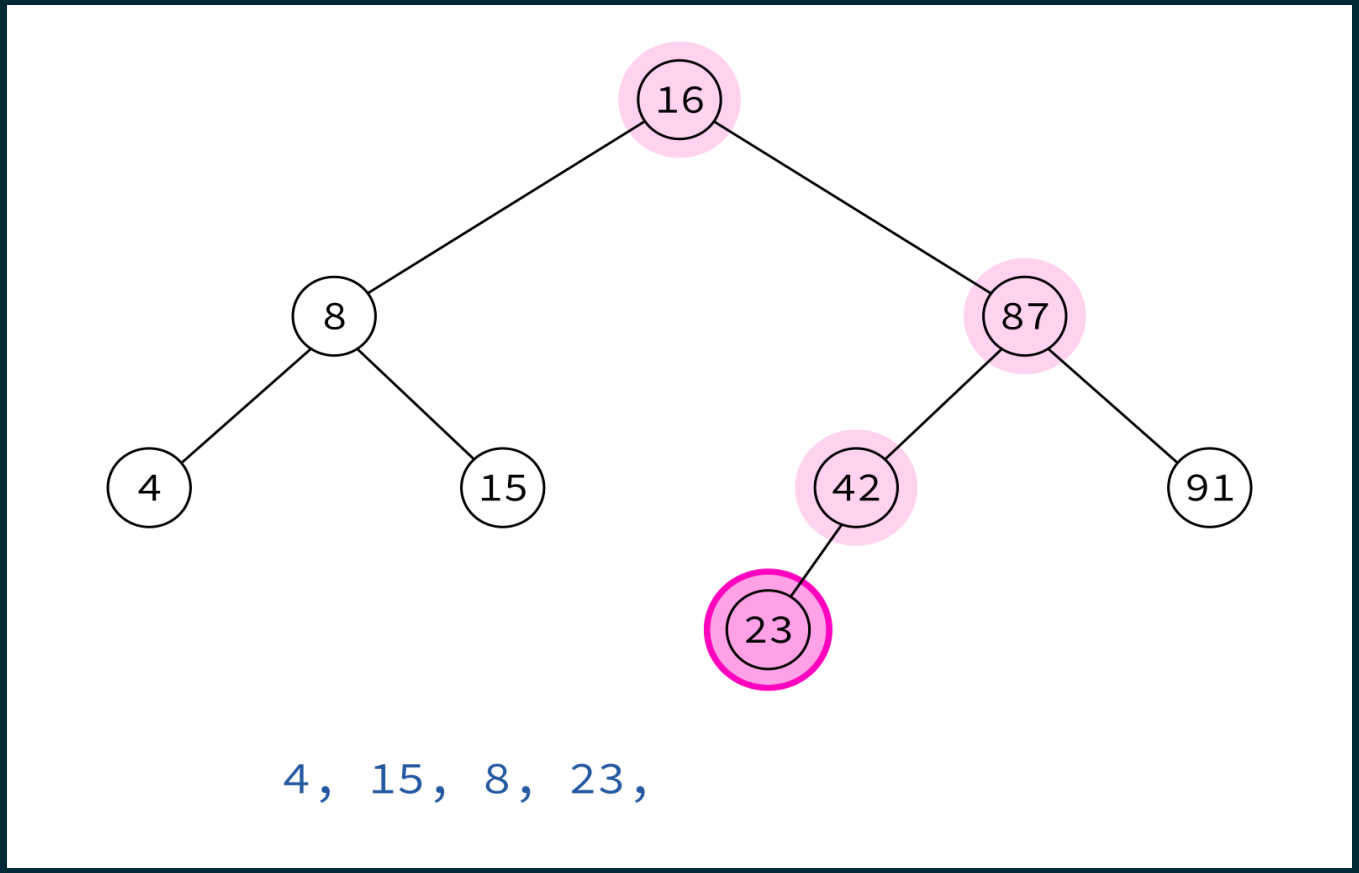
left, right, **node**

POSTORDER TRAVERSAL



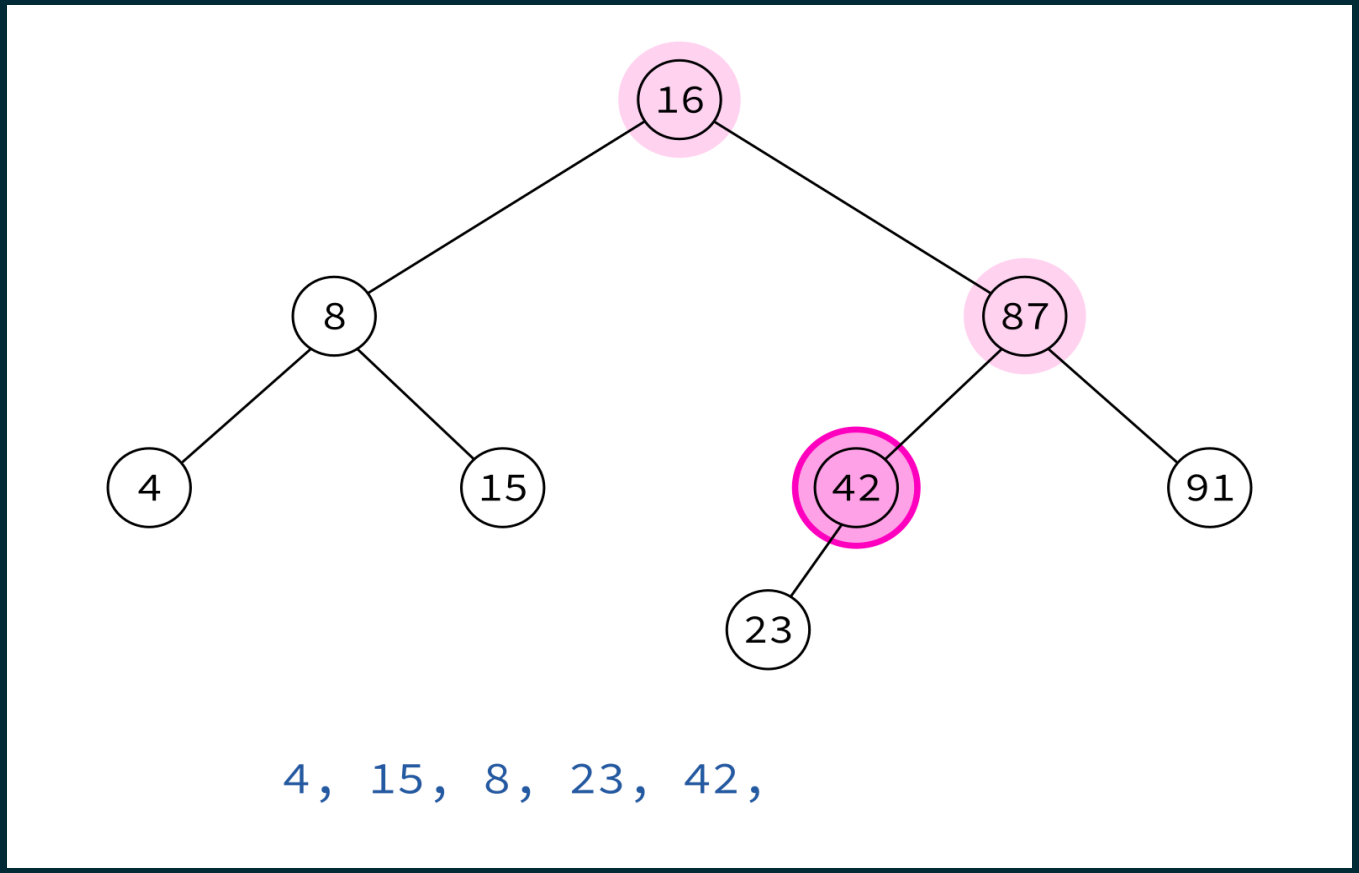
left, right, **node**

POSTORDER TRAVERSAL



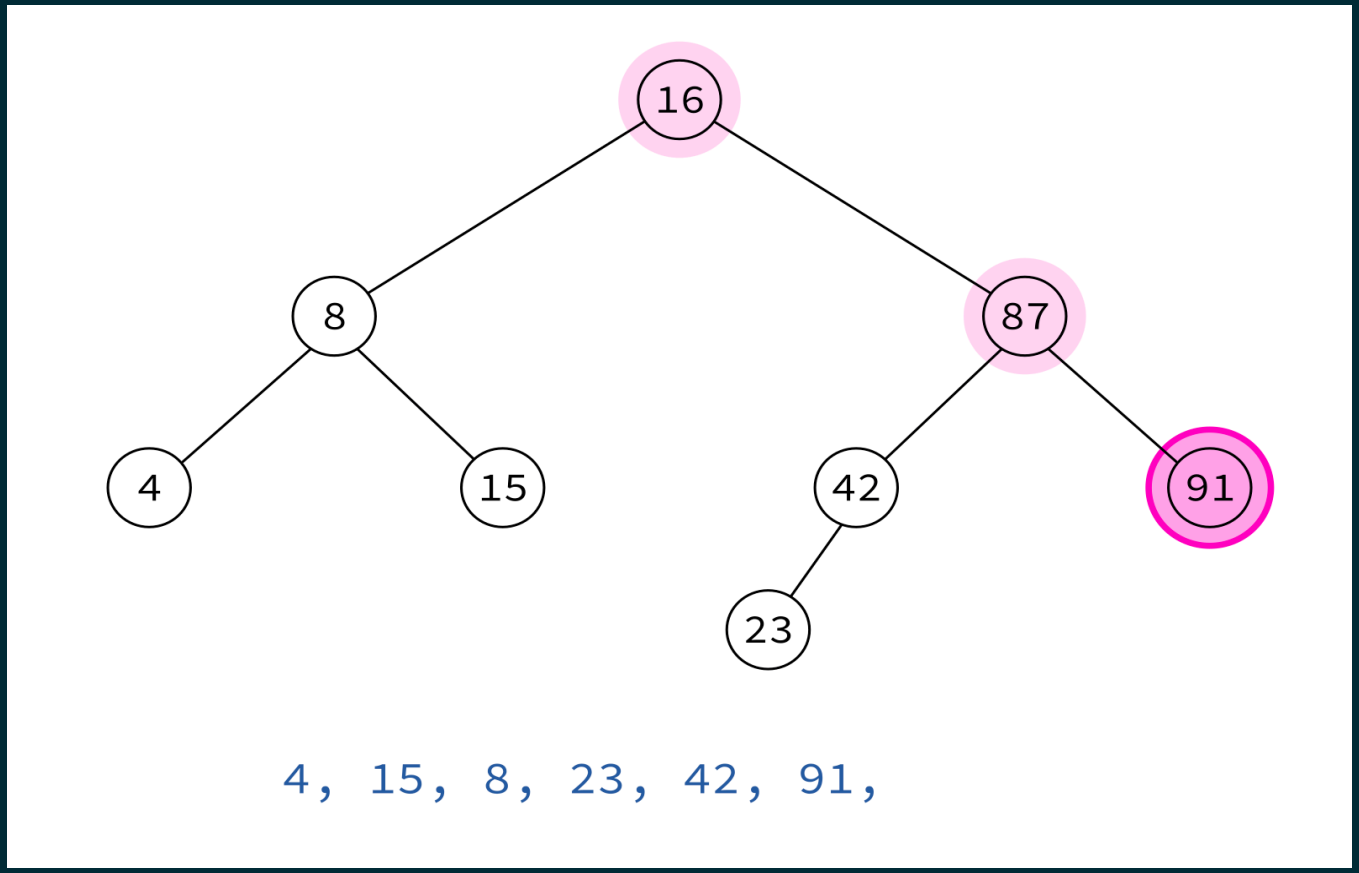
left, right, **node**

POSTORDER TRAVERSAL



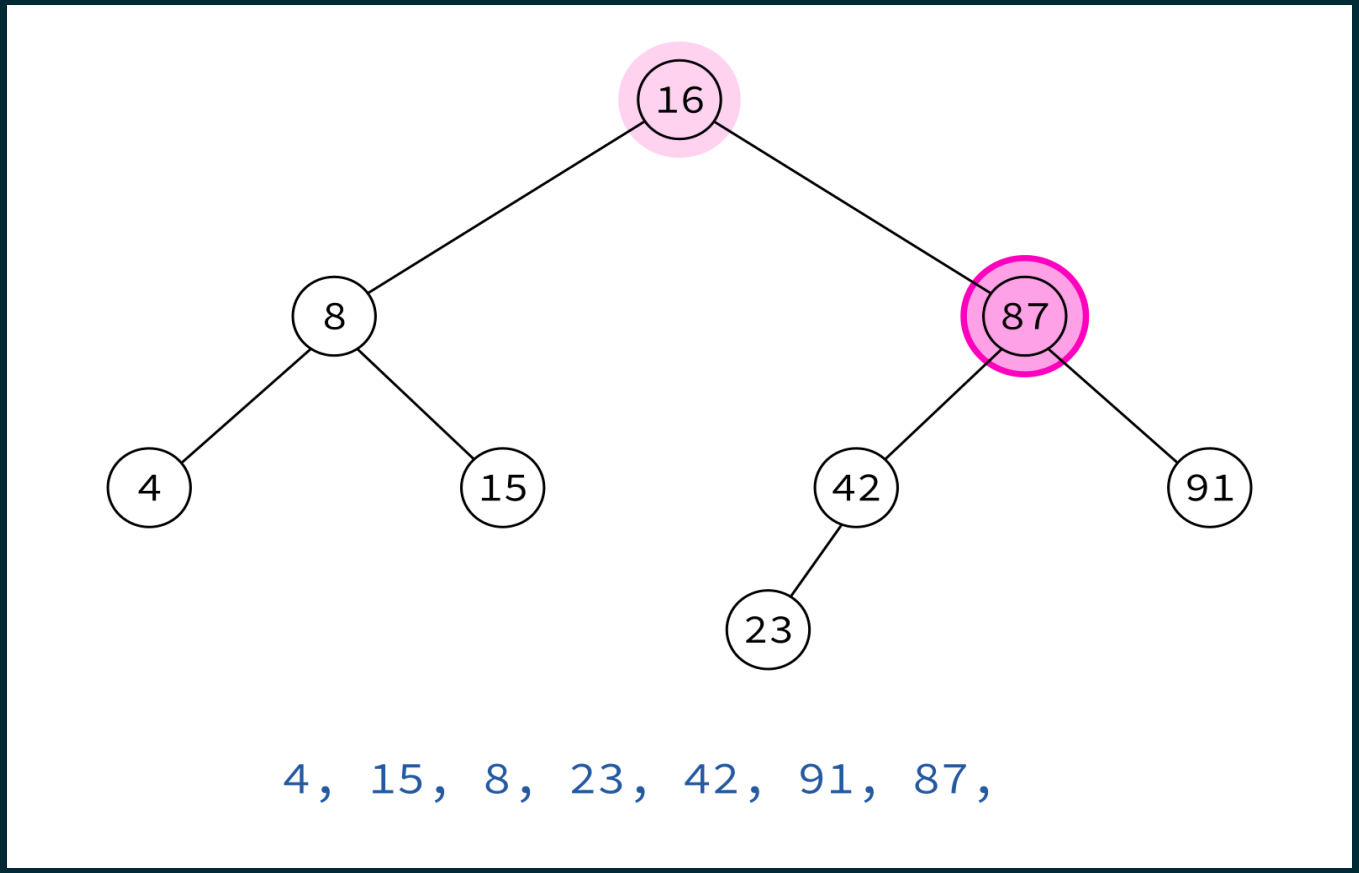
left, right, **node**

POSTORDER TRAVERSAL



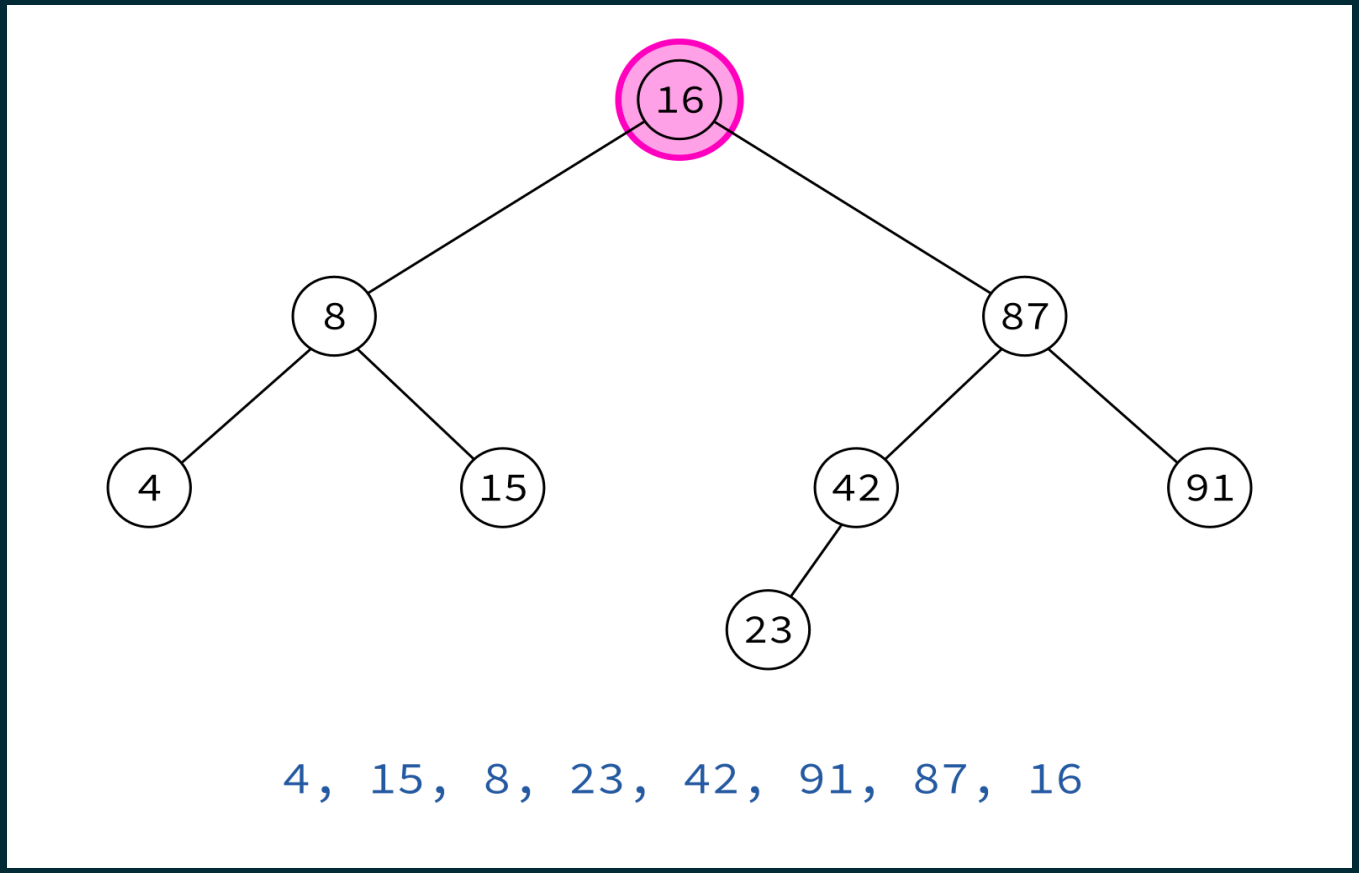
left, right, **node**

POSTORDER TRAVERSAL



left, right, **node**

POSTORDER TRAVERSAL

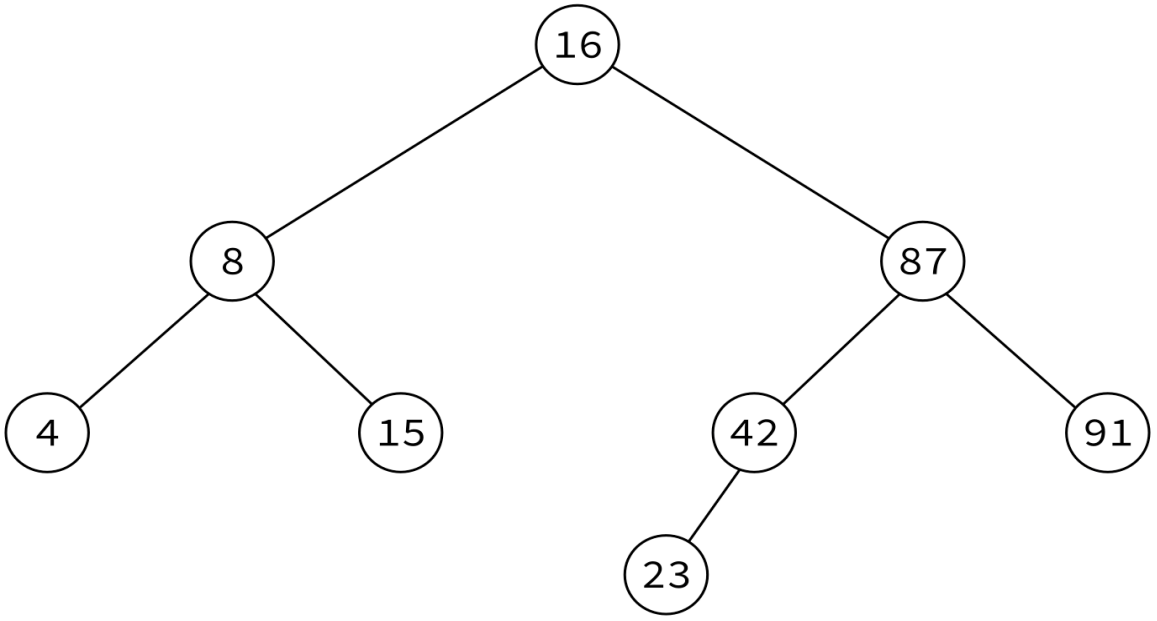


left, right, **node**

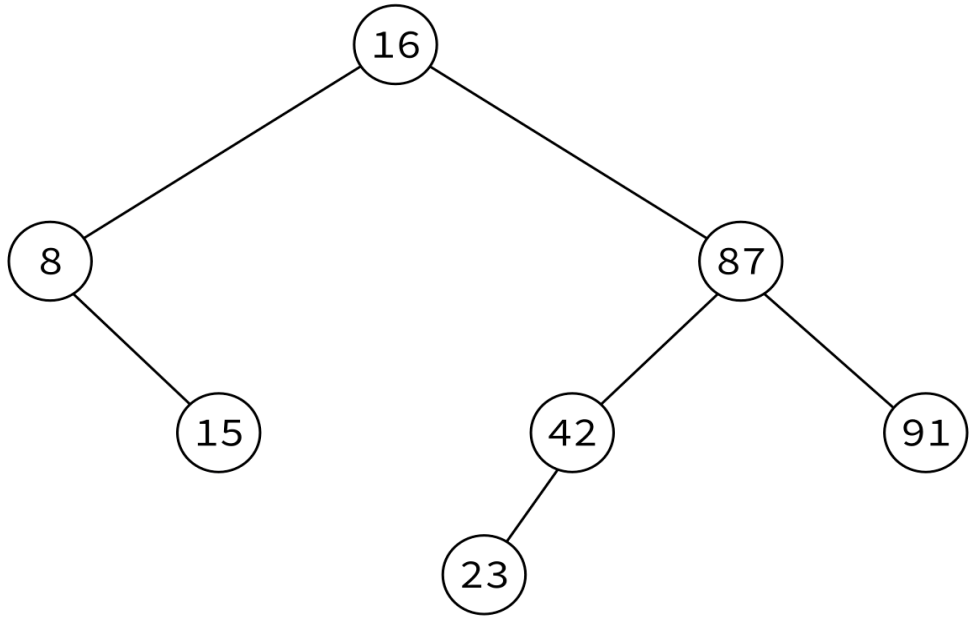
POSTORDER TRAVERSAL

Typical use: Delete the tree.

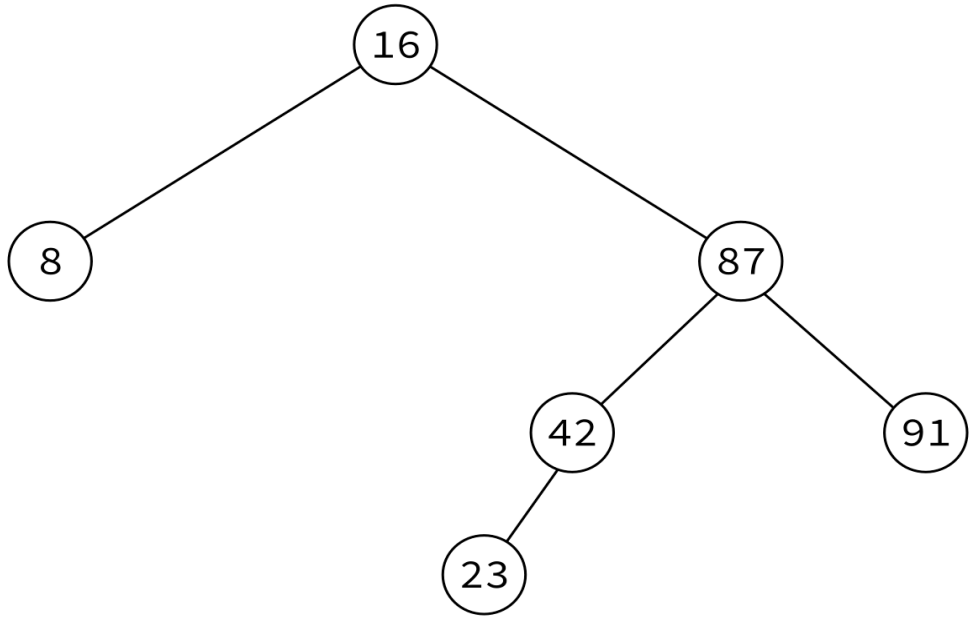
If you delete keys in postorder, then you will only ever be removing nodes without children.



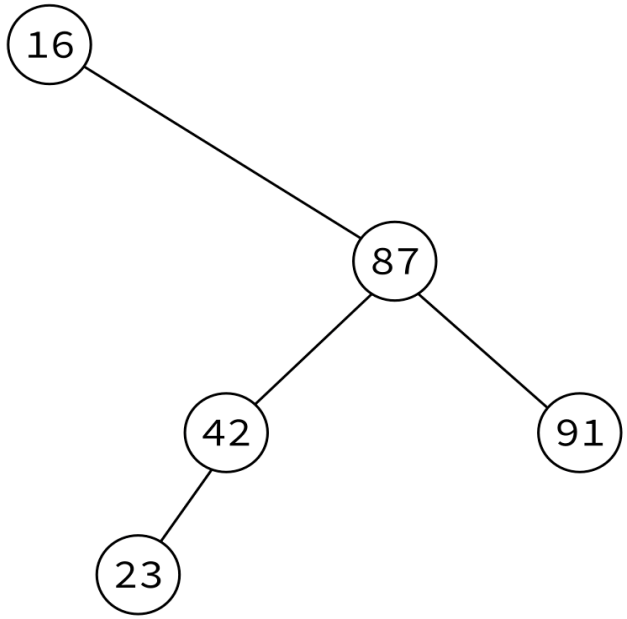
4, 15, 8, 23, 42, 91, 87, 16



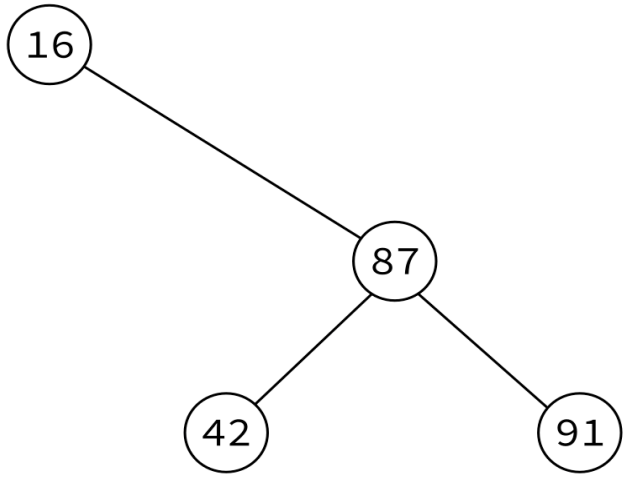
4, 15, 8, 23, 42, 91, 87, 16



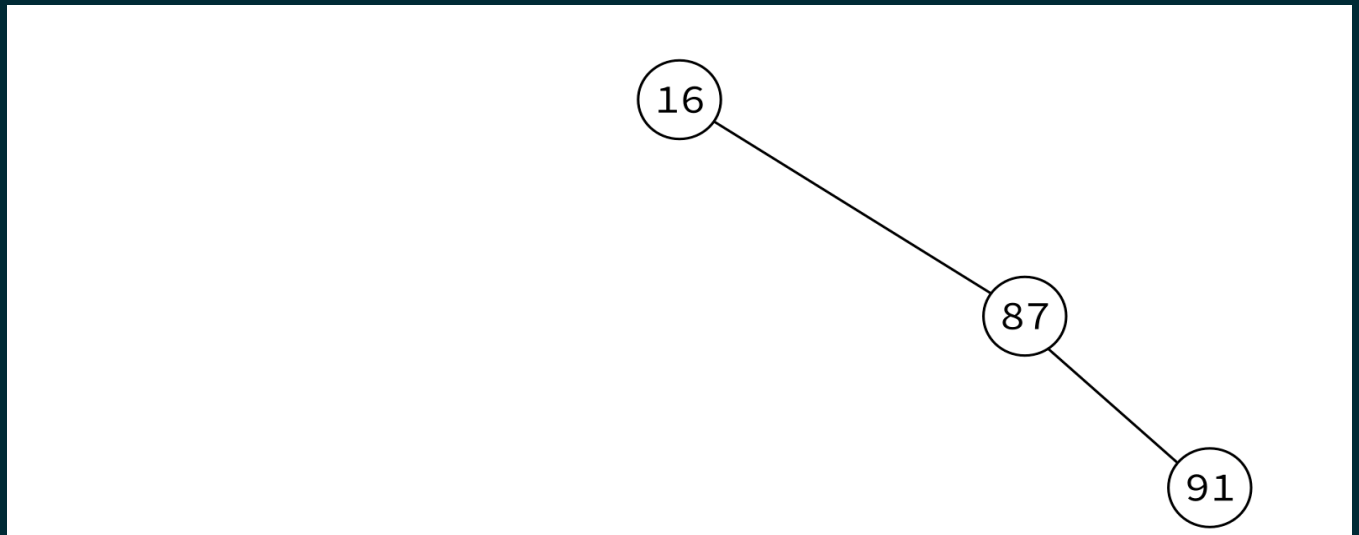
4, 15, 8, 23, 42, 91, 87, 16



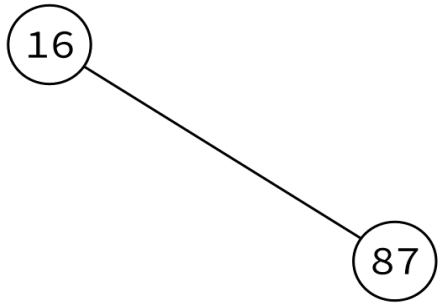
4, 15, 8, 23, 42, 91, 87, 16



4, 15, 8, 23, 42, 91, 87, 16



4, 15, 8, 23, 42, 91, 87, 16

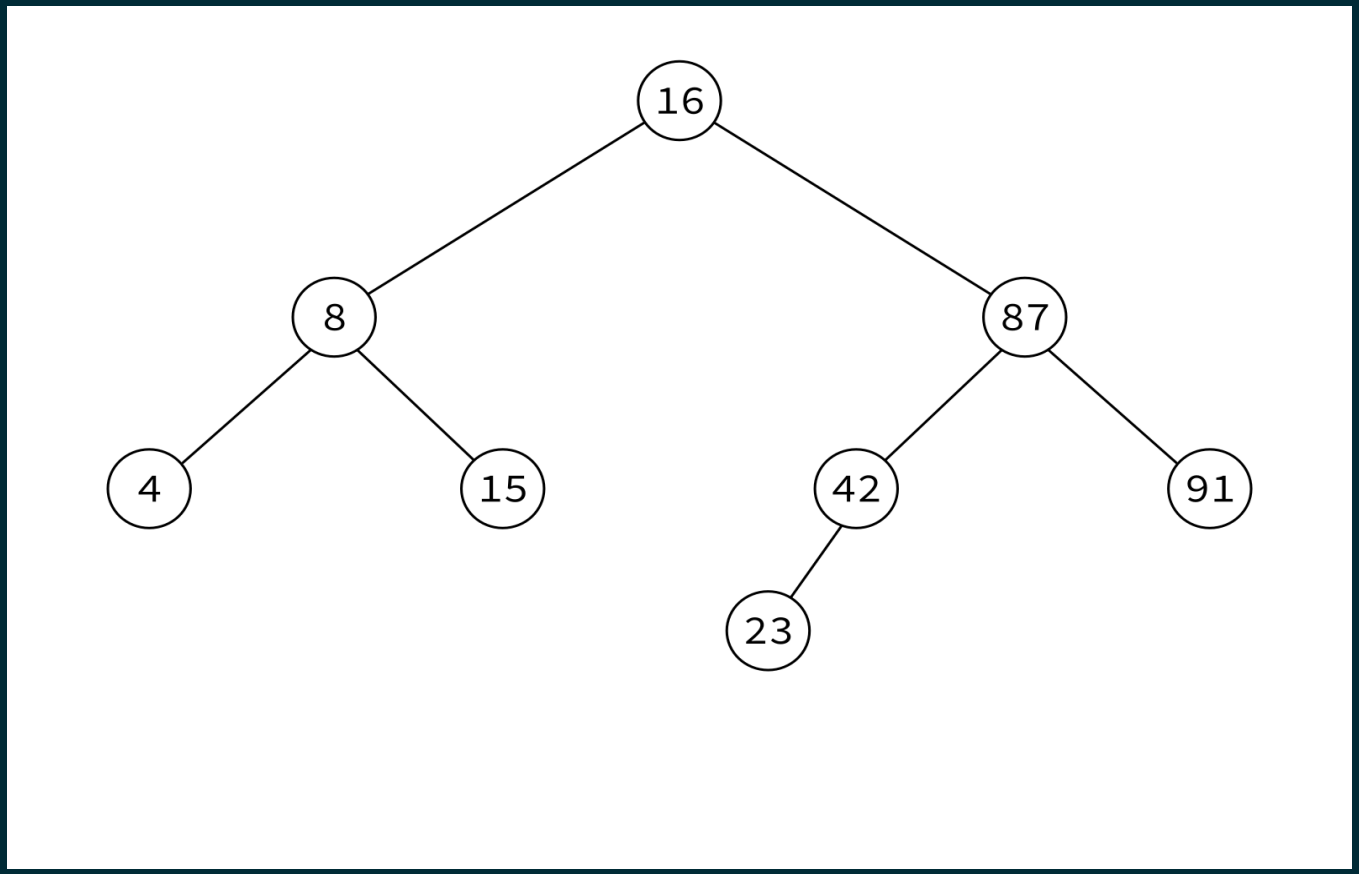


4, 15, 8, 23, 42, 91, 87, 16

4, 15, 8, 23, 42, 91, 87, 16

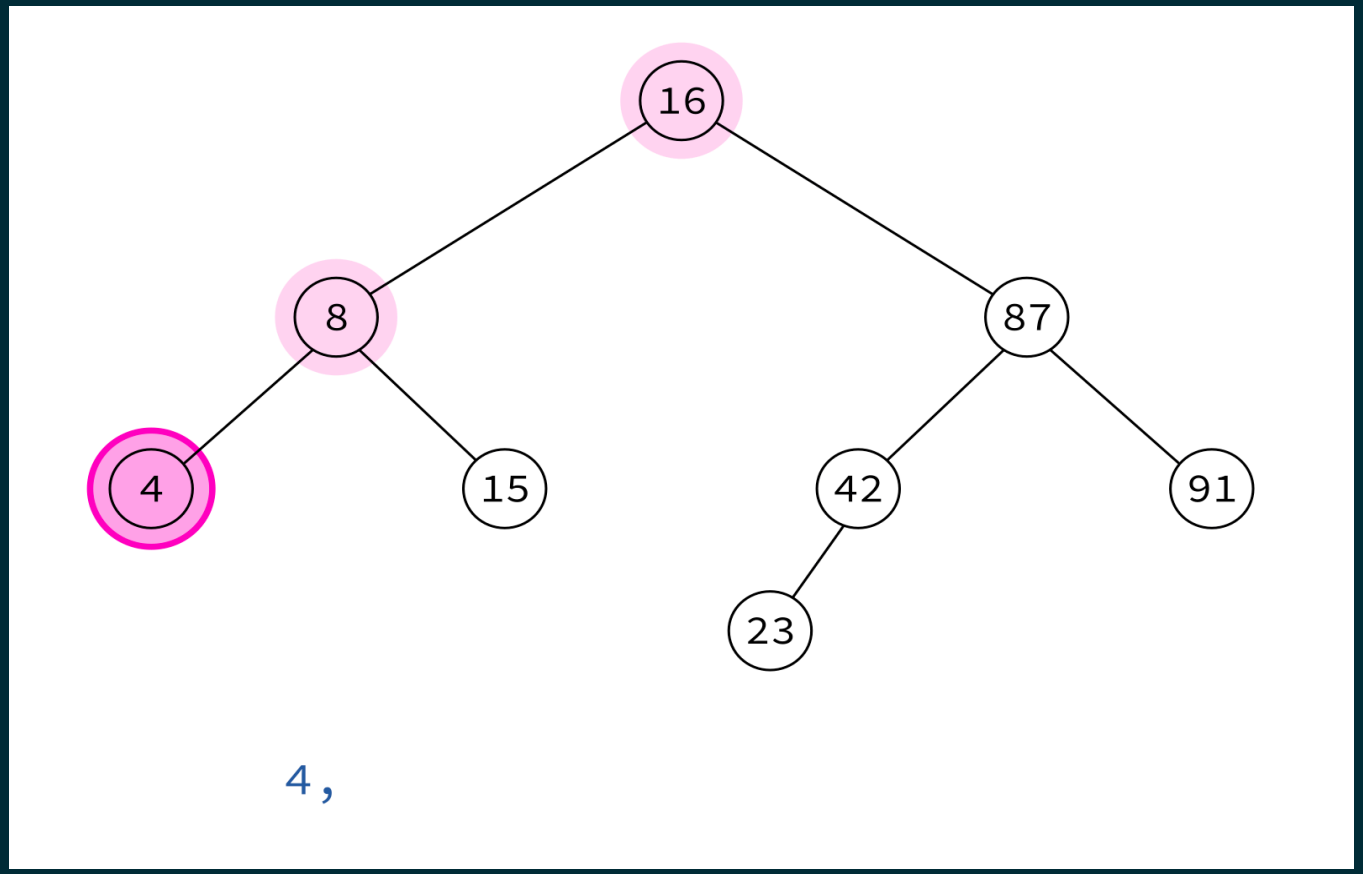
4, 15, 8, 23, 42, 91, 87, 16

INORDER TRAVERSAL



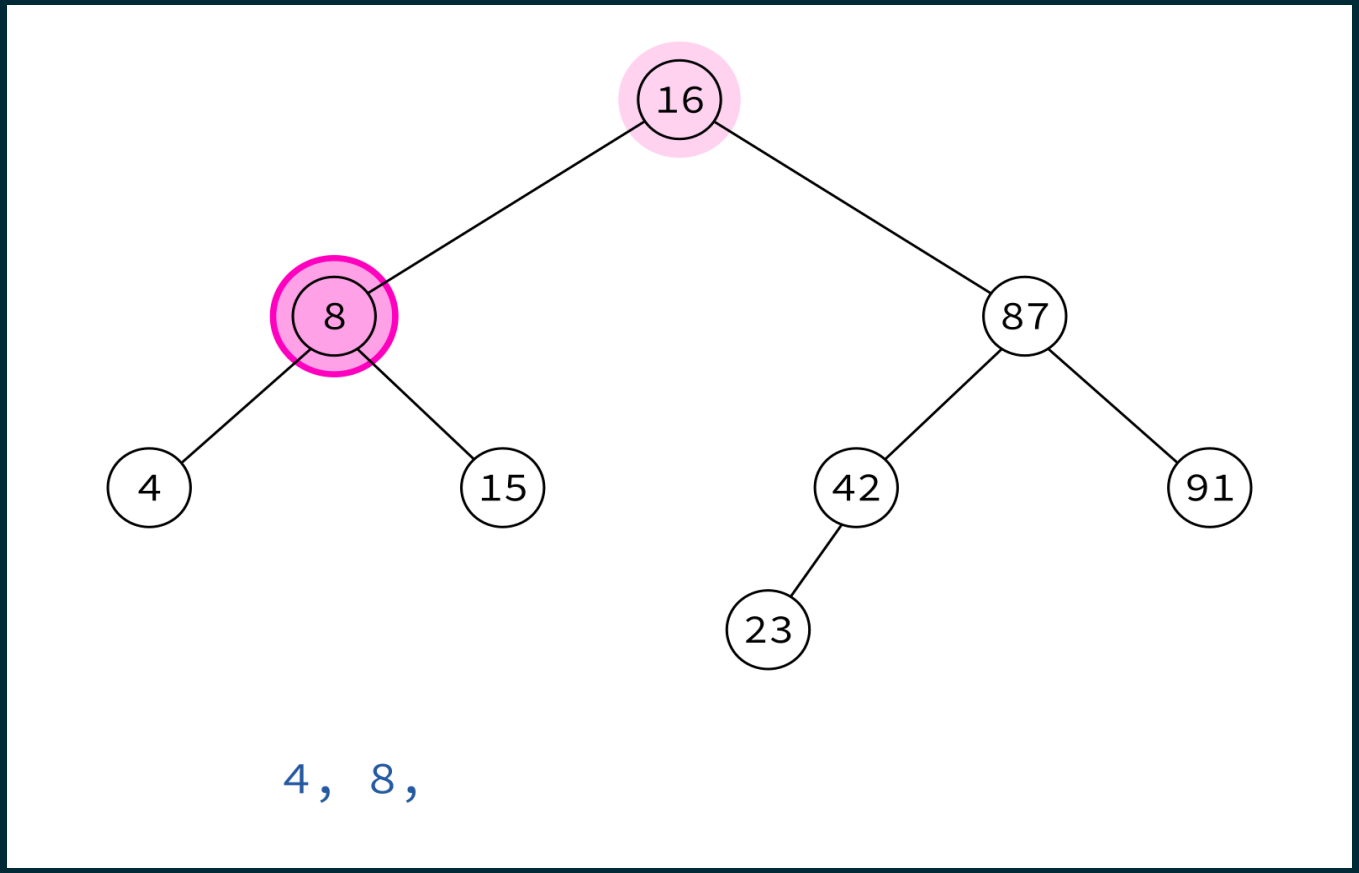
left, **node**, right

INORDER TRAVERSAL



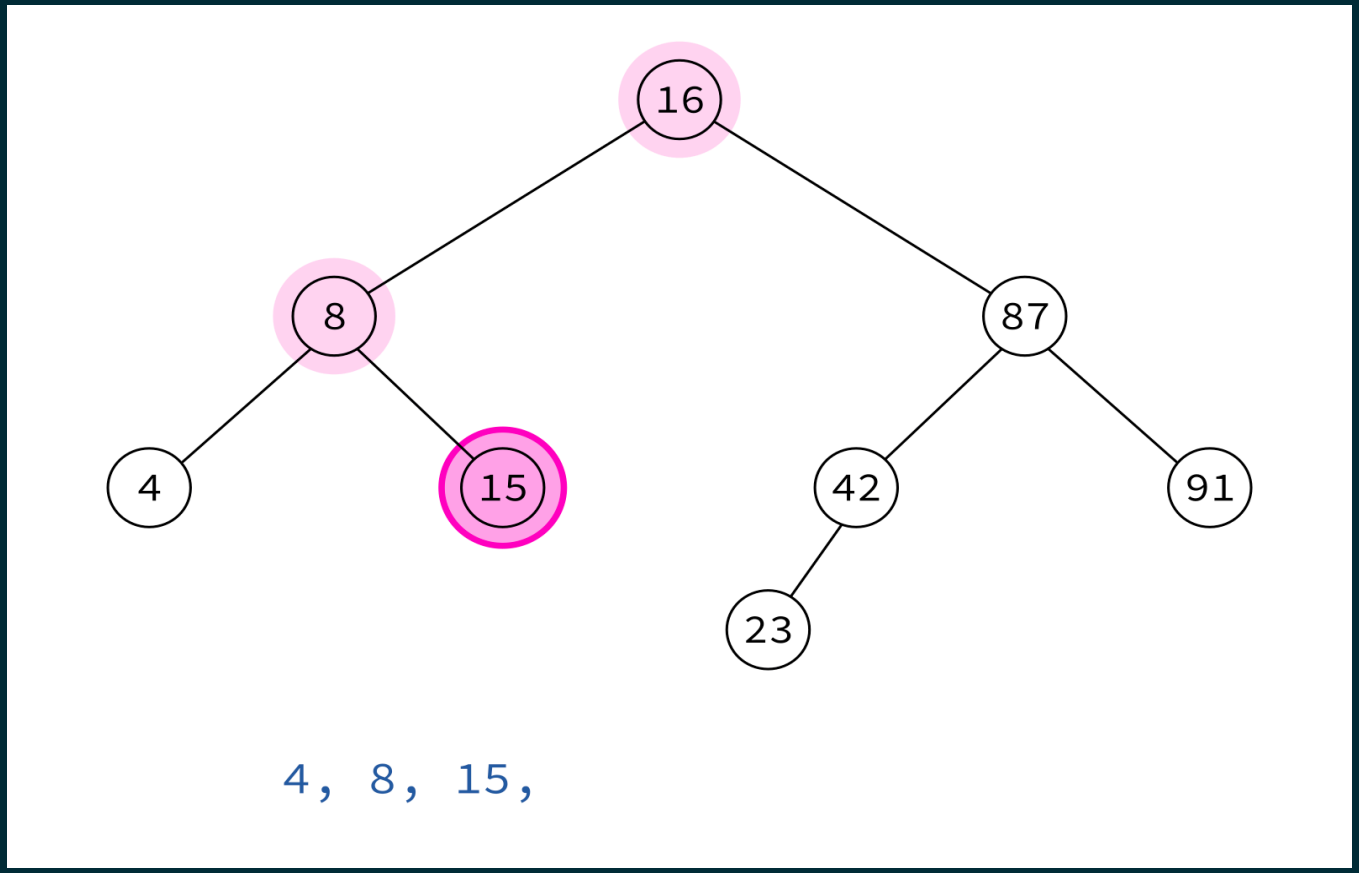
left, **node**, right

INORDER TRAVERSAL



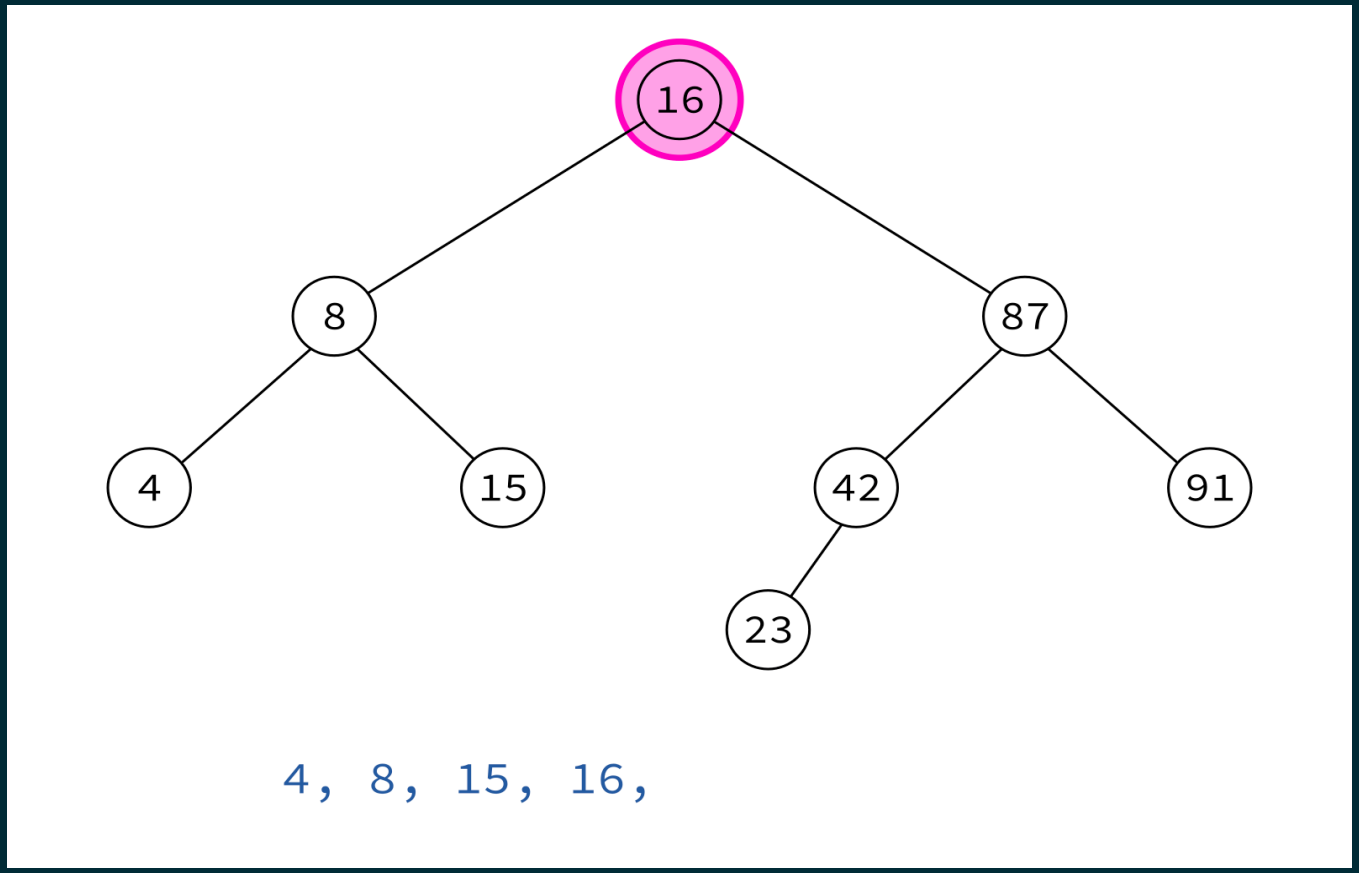
left, **node**, right

INORDER TRAVERSAL



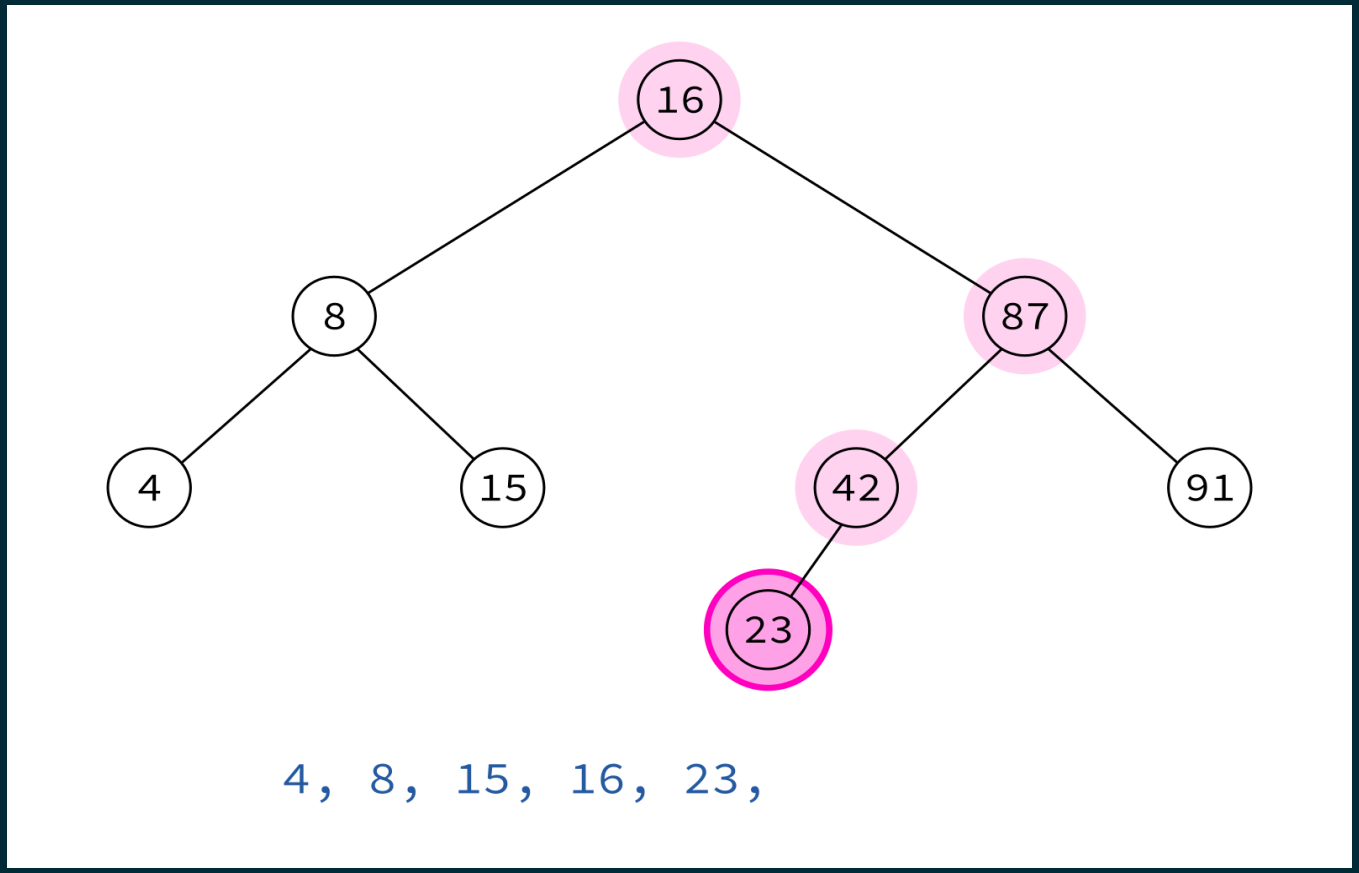
left, **node**, right

INORDER TRAVERSAL



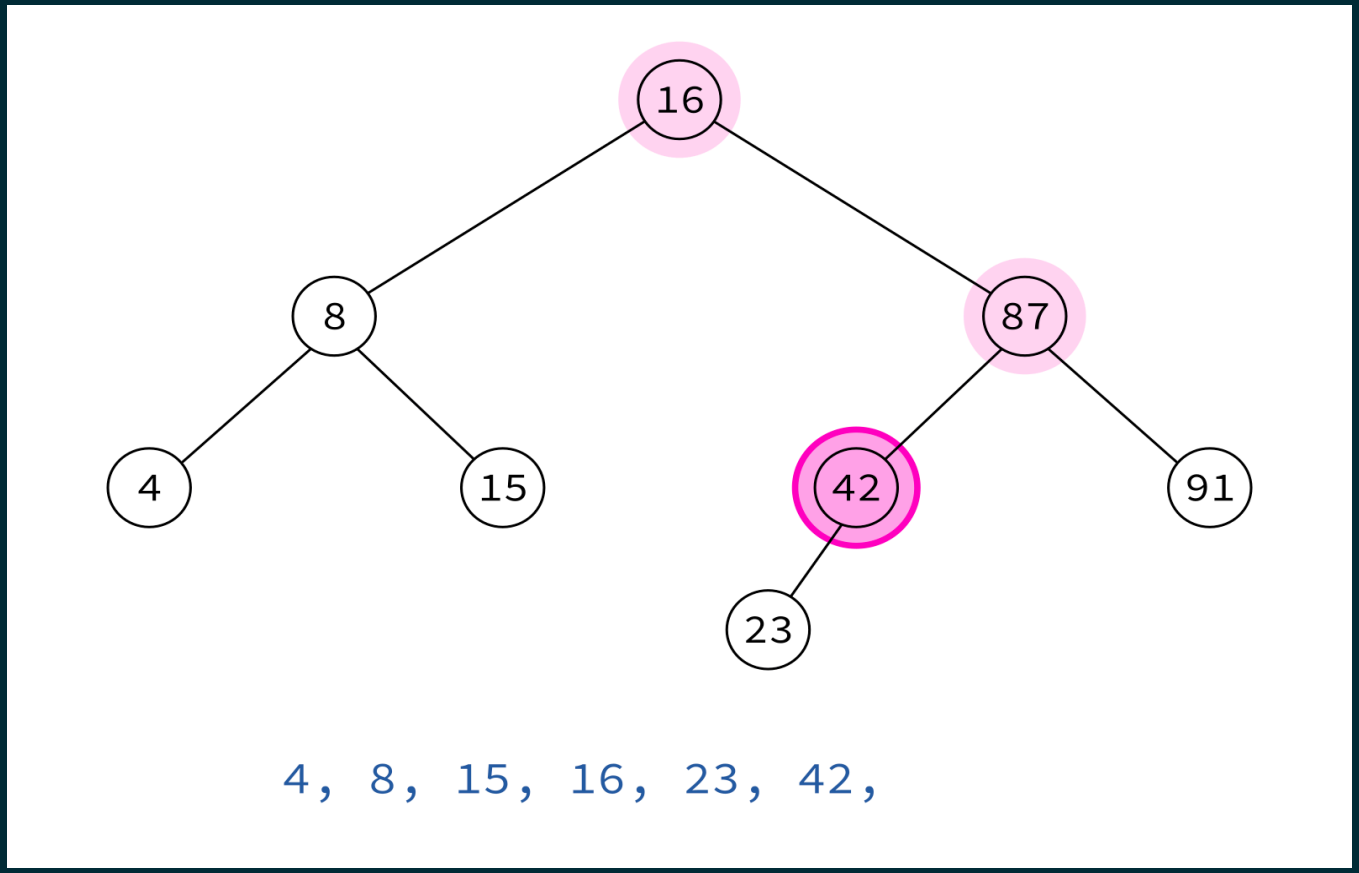
left, **node**, right

INORDER TRAVERSAL



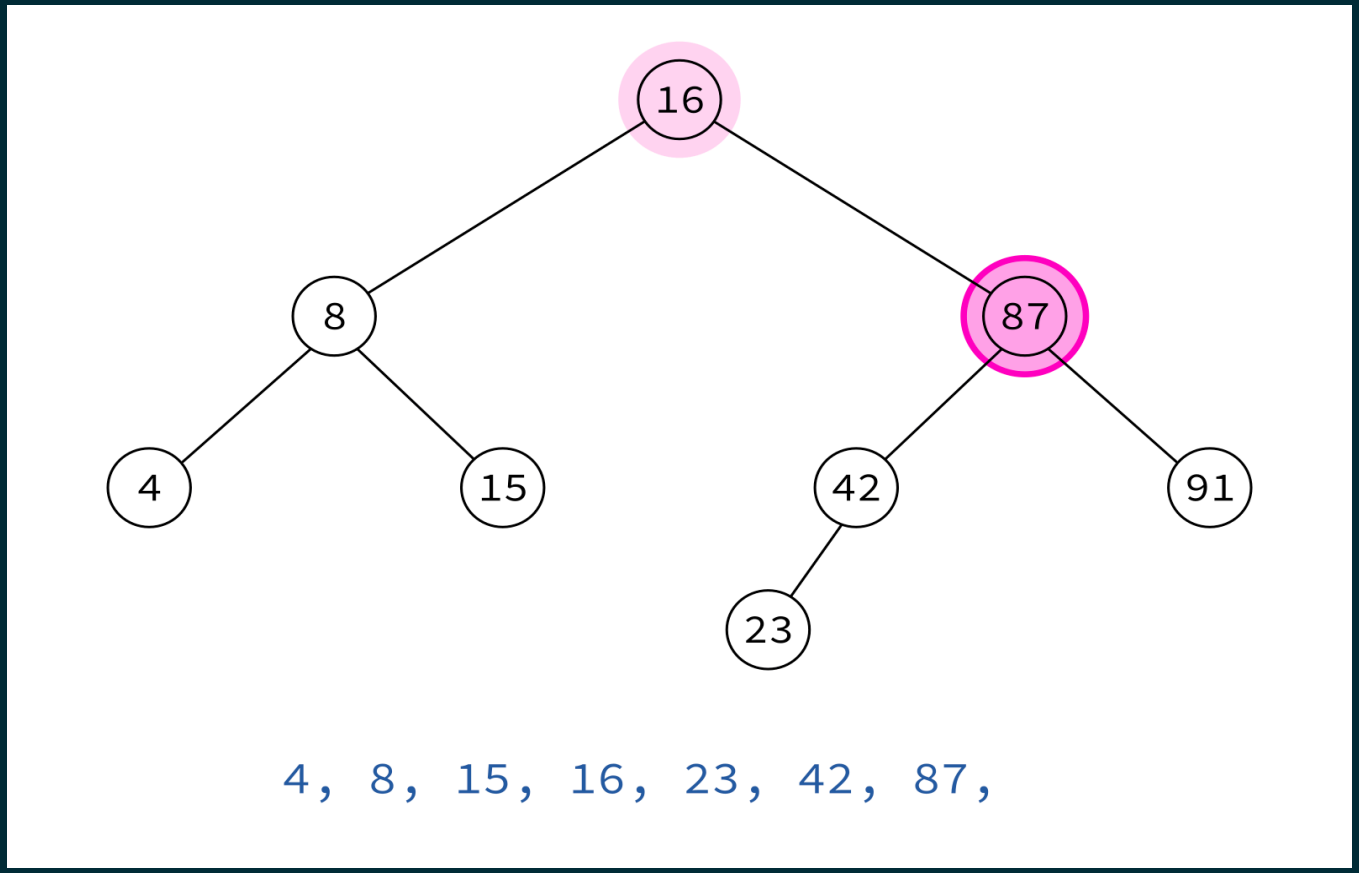
left, **node**, right

INORDER TRAVERSAL



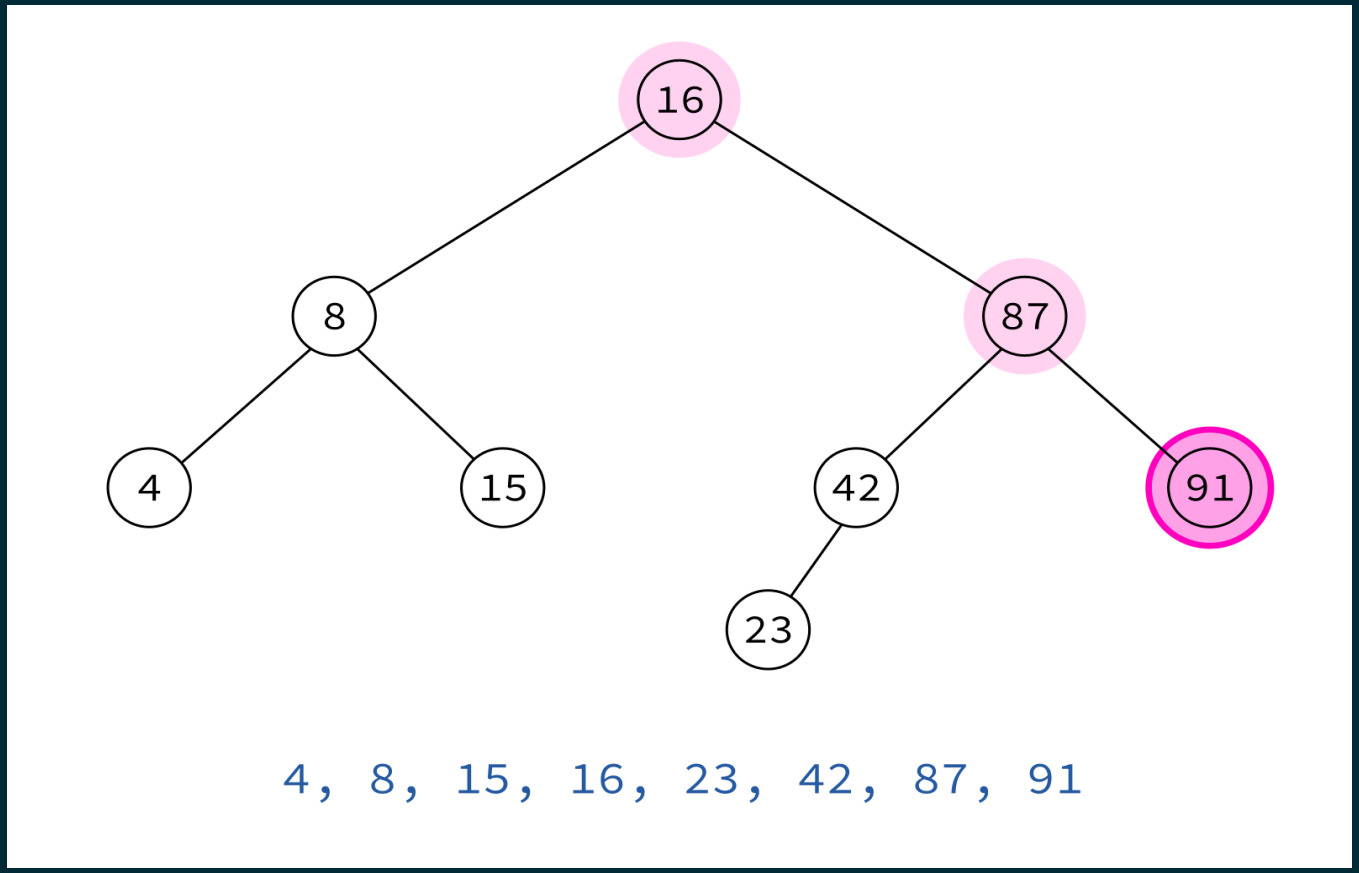
left, **node**, right

INORDER TRAVERSAL



left, **node**, right

INORDER TRAVERSAL



left, **node**, right

INORDER TRAVERSAL

Typical use: Turn a BST into a sorted list of keys.

REFERENCES

- In optional course texts:
  - *Problem Solving with Algorithms and Data Structures using Python* by Miller and Ranum, discusses binary trees in Chapter 7.
- Elsewhere:
  - Cormen, Leiserson, Rivest, and Stein discusses graph theory and trees in Appendices B.4 and B.5, and binary search trees in Chapter 12.

REVISION HISTORY

- 2021-03-01 Add another slide to `insert` explanation
- 2021-02-28 Initial publication