Today

- Inversion
- Insertion
- Traversals
- Deletion

Inversion

Homebrew

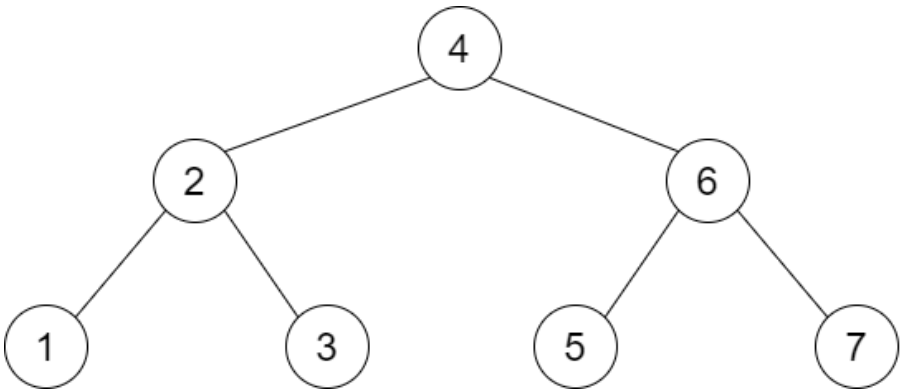
Screenshot [show]

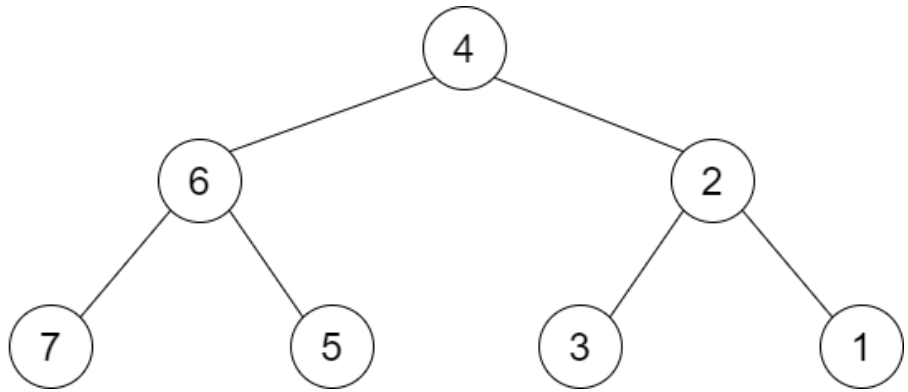
Original author(s)	Max Howell
Initial release	21 May 2009; 13 years ago[1]
Stable release	3.6.1 / 11 September 2022; 6 days ago[2]
Repository	github.com/Homebrew/brew
Written in	Ruby
Operating system	macOS, Linux
Available in	English
Type	Package manager
License	BSD 2-Clause License
Website	brew.sh

Inversion

- Given a binary tree, swap the left and right child for every node

Inversion



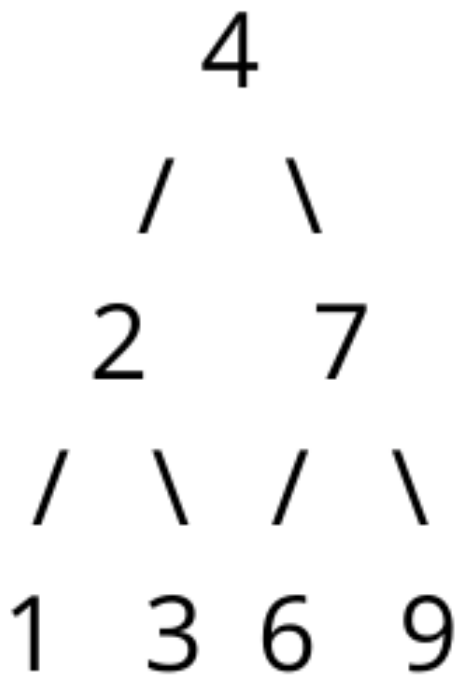


Inversion

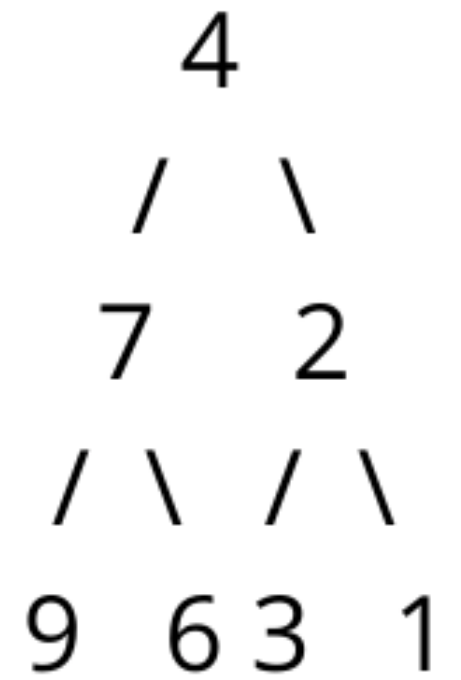
```
function invert(v)
  if v != null
    invert(v.left)
    invert(v.right)
    temp = v.left
    v.left = v.right
    v.right = temp
```

Inversion

Invert a binary tree



original tree



Inverted tree

Insertion

- How can we add a new node to a BST?
- How can we find where the node belongs?
- What happens if the tree is empty?

Insertion

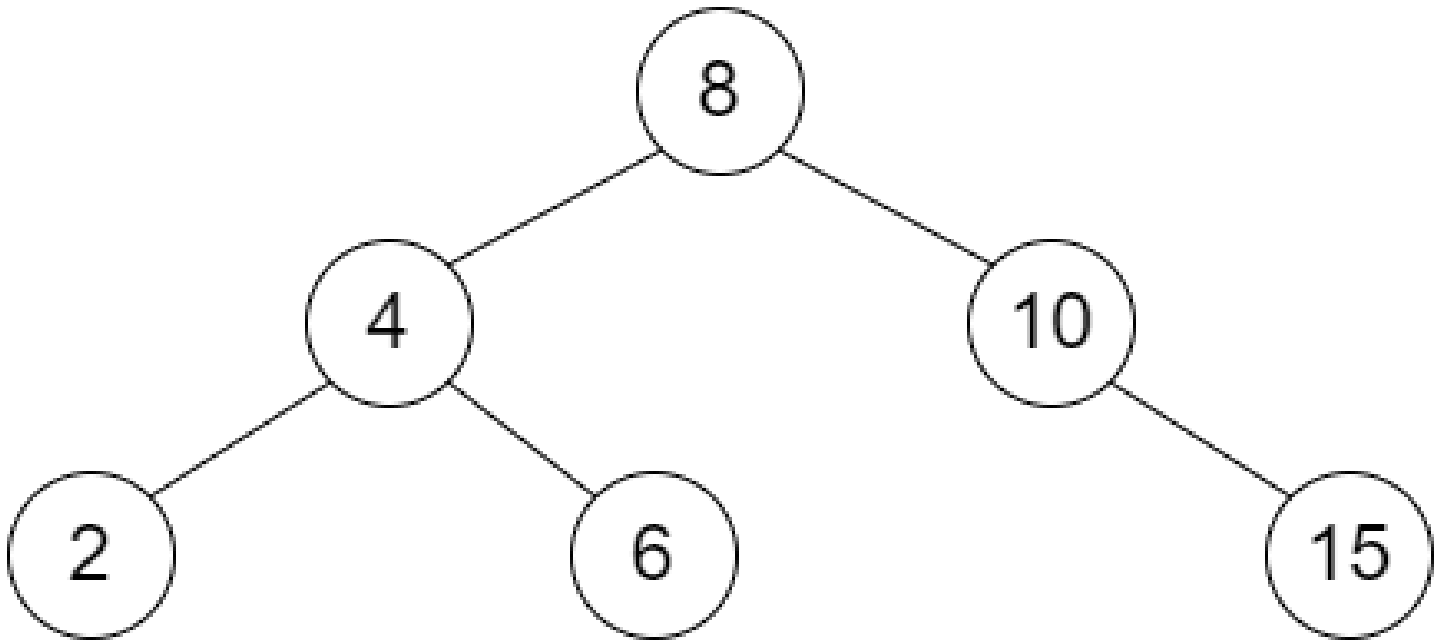
```
function insert(root, val)
  if root == null
    root.value = val
  else if val <= root.value
    root.left = insert(root.left, val)
  else if val > root.value
    root.right = insert(root.right, val)
  return root
```

What is wrong about this pseudo code?

```
function insert(val)
if root == null
   root = new Node(val);
else
   root = insert(root, val);
```

```
function insert(node, val)
  if val < node.value
    if(node.left!=null)
        node.left = insert(node.left, val)
    else node.left = new Node(val);
  else if val > node.value
    if(node.right!=null)
        node.right = insert(node.right, val)
    else node.right = new Node(val);
   //if node.value = val, ignore
return node
```

Insertion



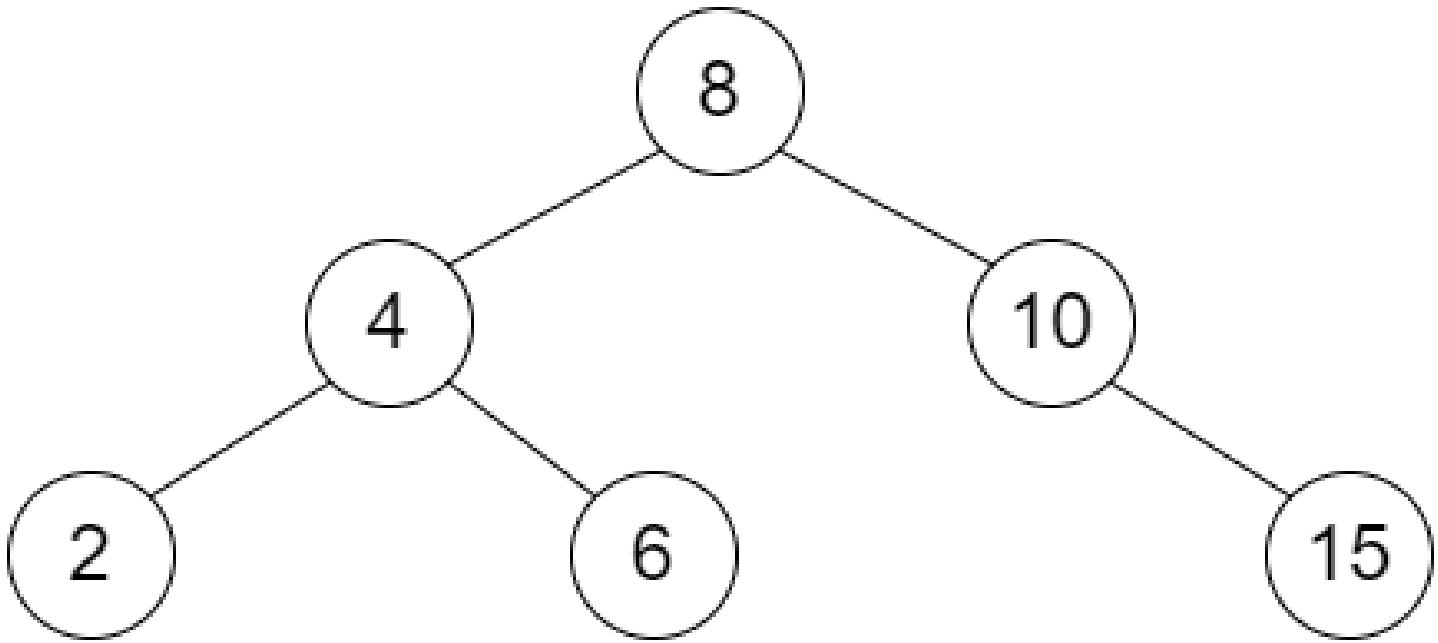
Pre-Order Traversal

- Root->Left->Right
- * The order they were added to the tree.*

Pre-Order Traversal

```
function preOrder(v)
  if v != null
    print(v.value)
    preOrder(left)
    preOrder(right)
```

Pre-Order Traversal



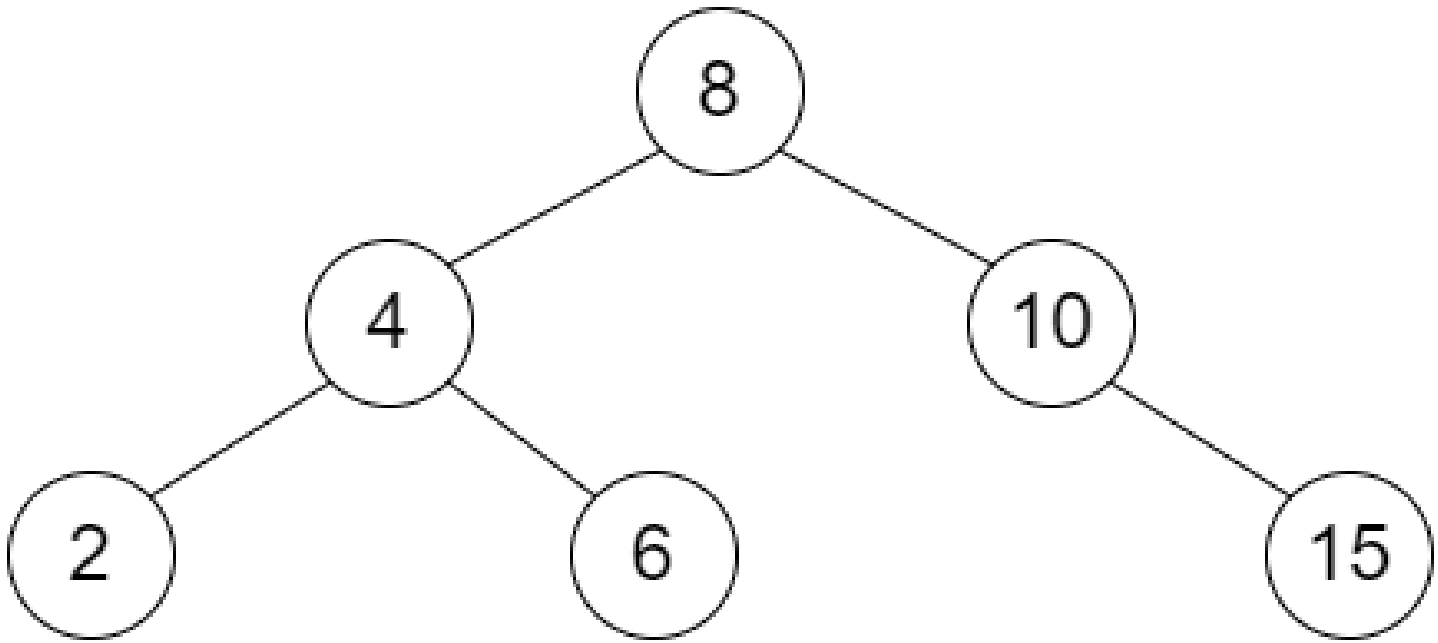
In-Order Traversal

- How can we print the values of a BST in sorted order?
- Left -> Root -> Right

In-Order Traversal

```
function inOrder(v)
  if v != null
    inOrder(left)
    print(v.value)
    inOrder(right)
```

In-Order Traversal



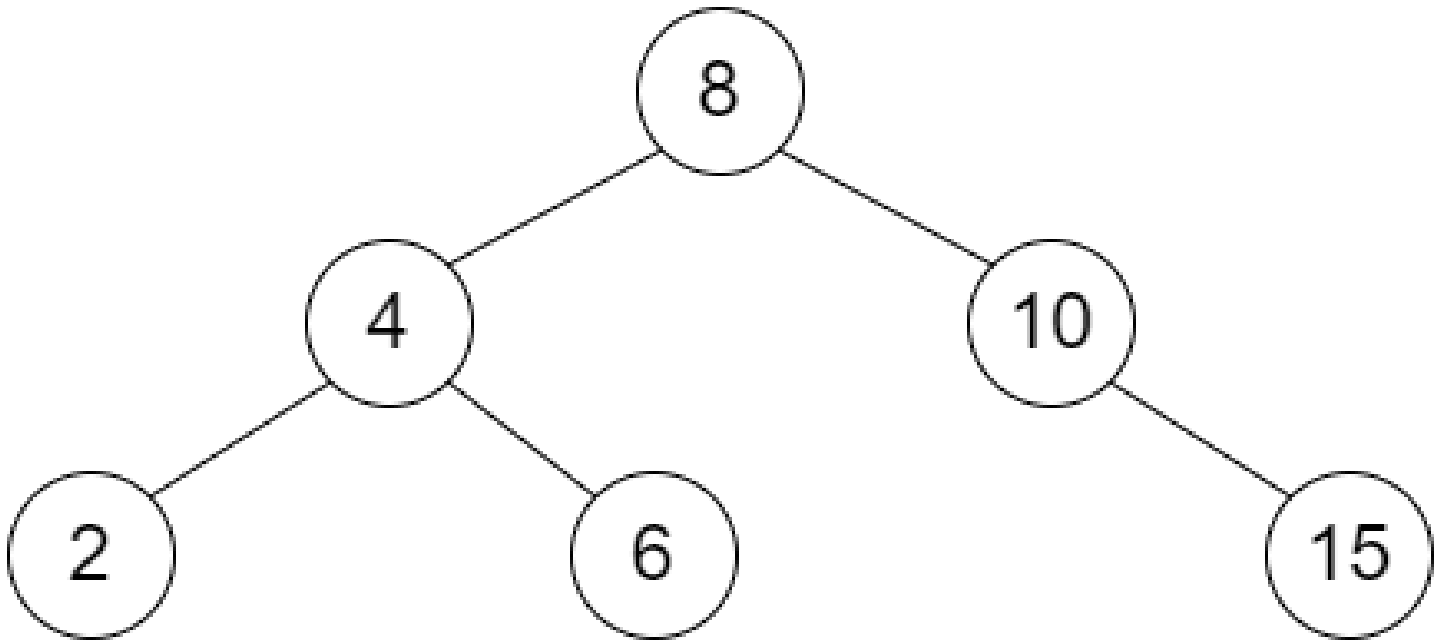
Post-Order Traversal

Left -> Right -> Root

Post-Order Traversal

```
function postOrder(v)
  if v != null
    postOrder(left)
    postOrder(right)
    print(v.value)
```

Post-Order Traversal

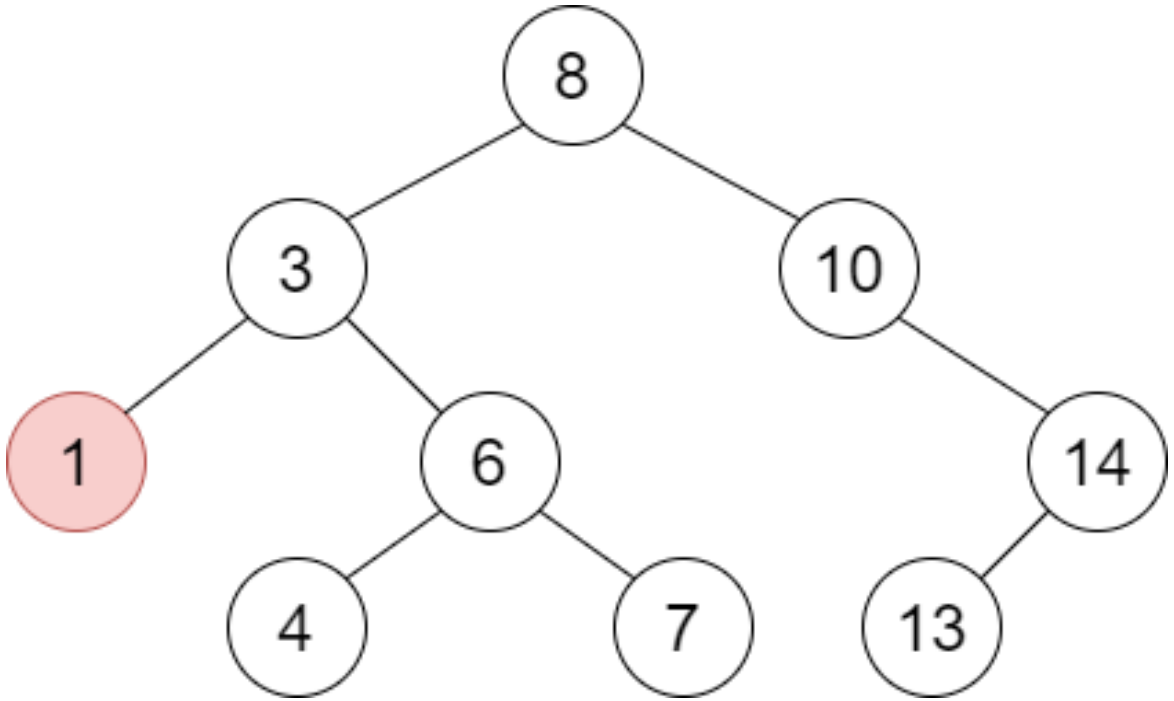


Deletion

- How do we remove a node from a BST?
- What cases do we need to consider?

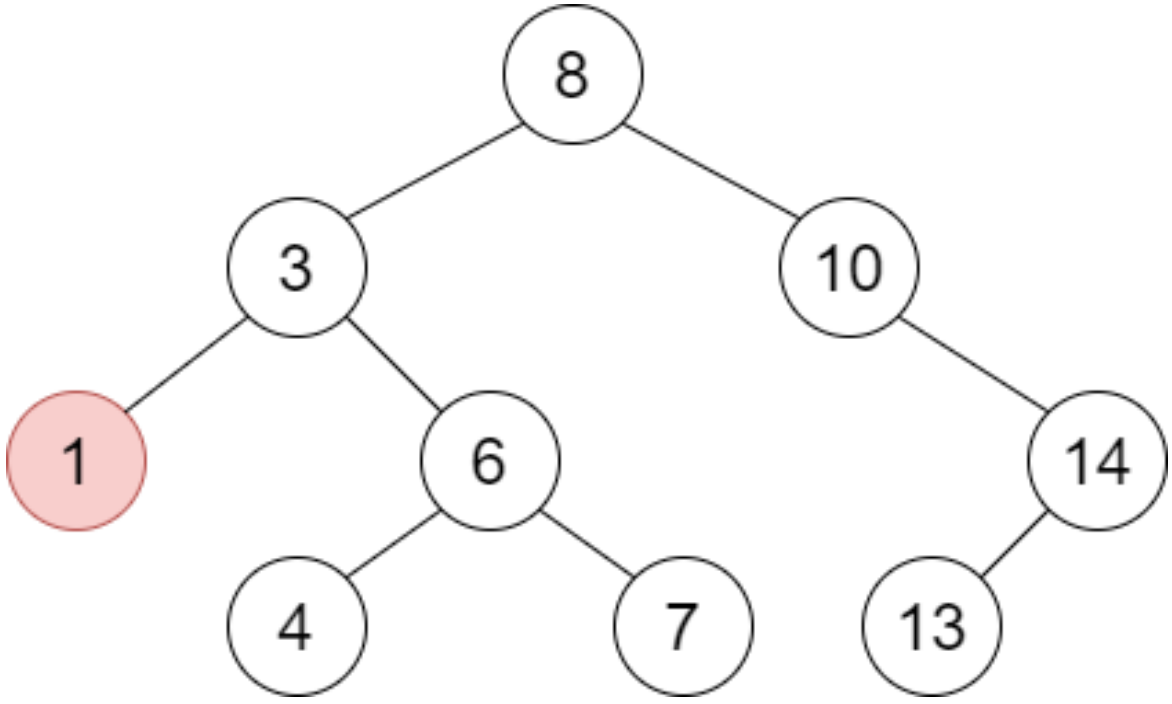
Deletion

No Children



Deletion

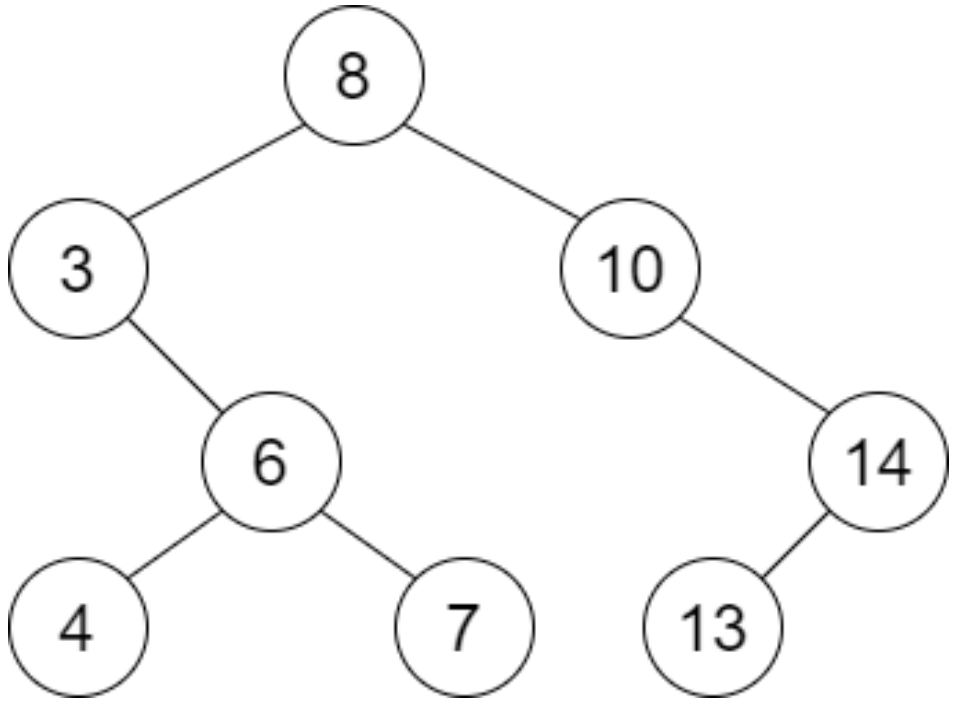
No Children



- If the node to be removed has no children, remove it

Deletion

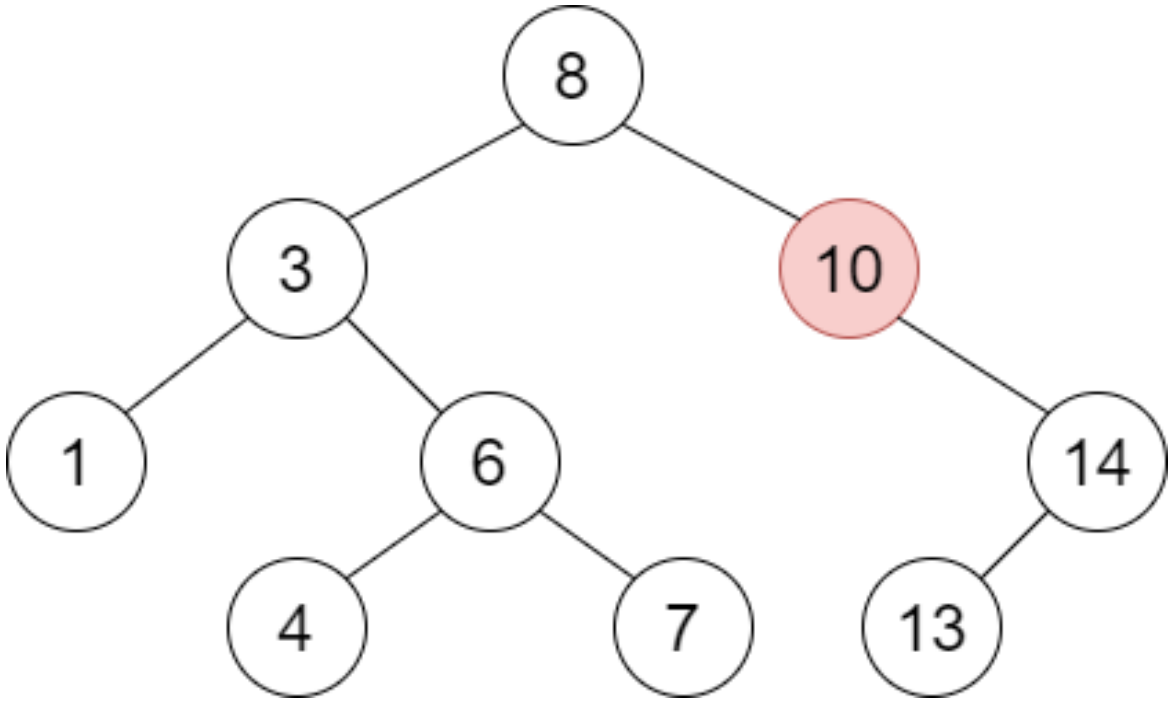
No Children



- If the node to be removed has no children, remove it

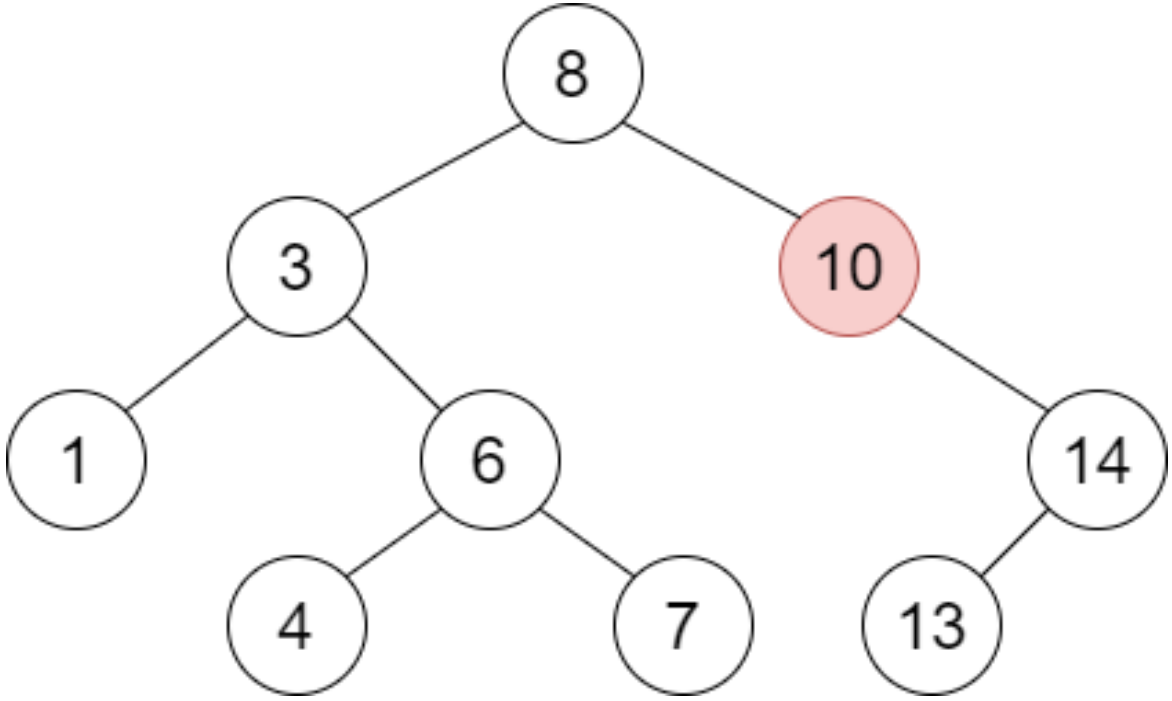
Deletion

One Child



Deletion

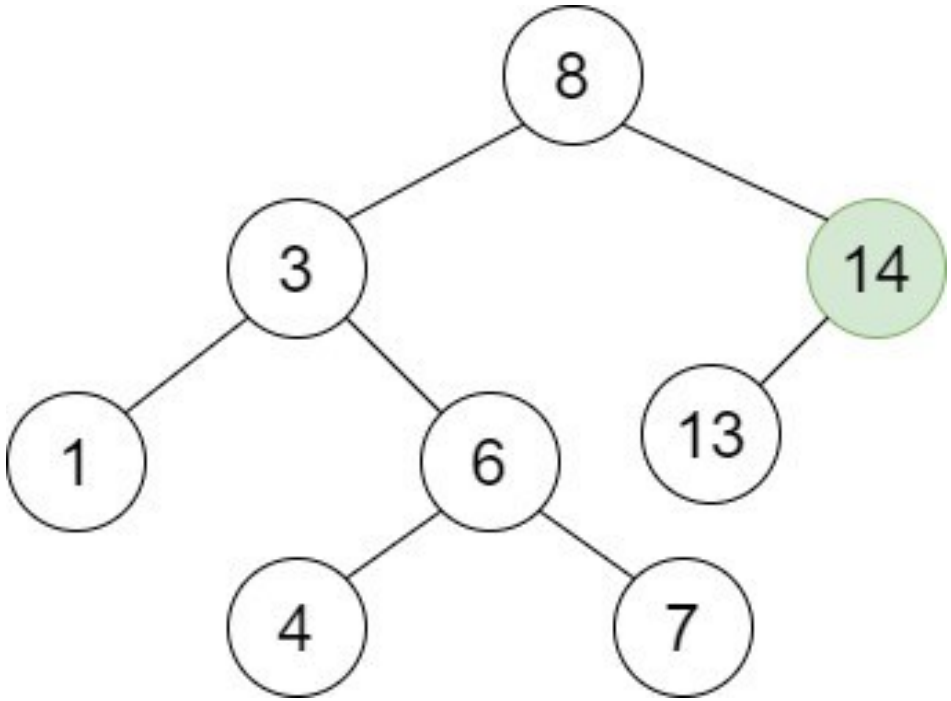
One Child



- If the node to be removed has one child, replace it with its child

Deletion

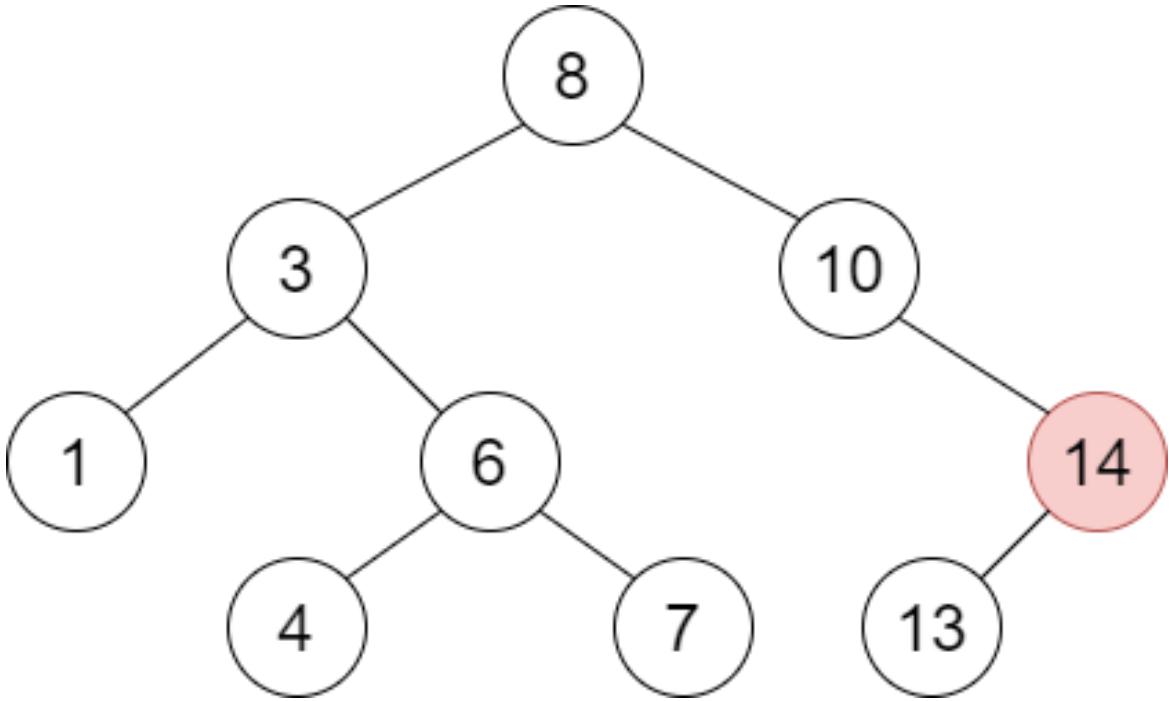
One Child



- If the node to be removed has one child, replace it with its child

Deletion

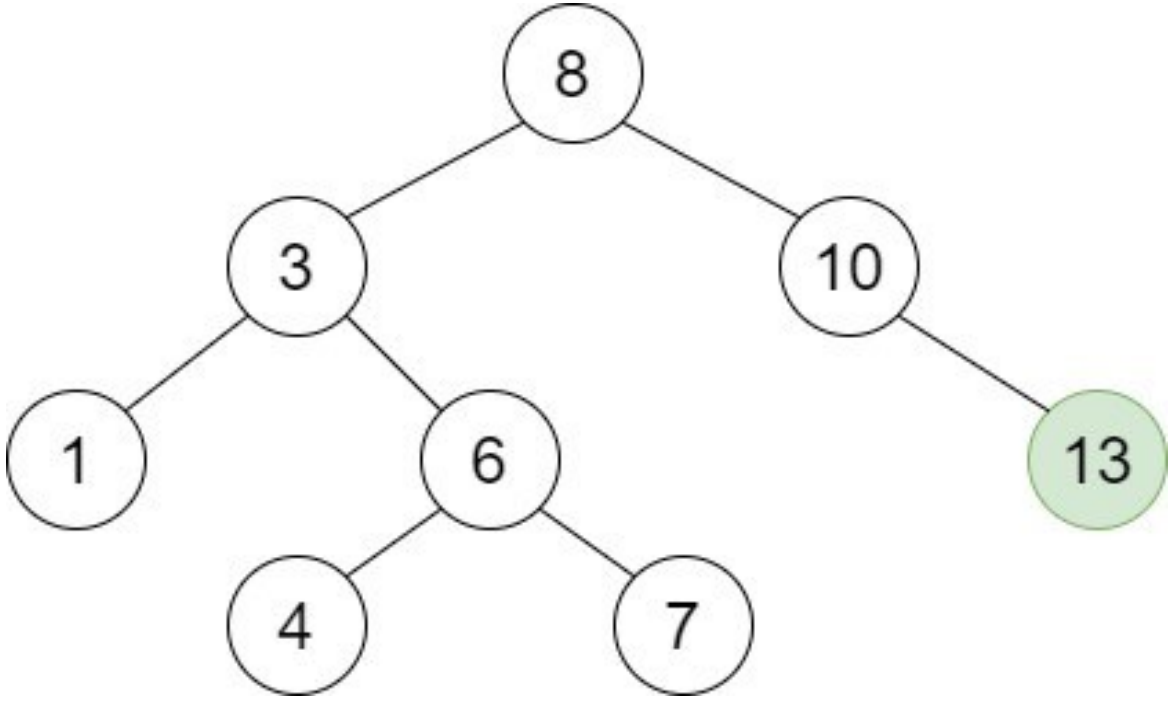
One Child



- If the node to be removed has one child, replace it with its child

Deletion

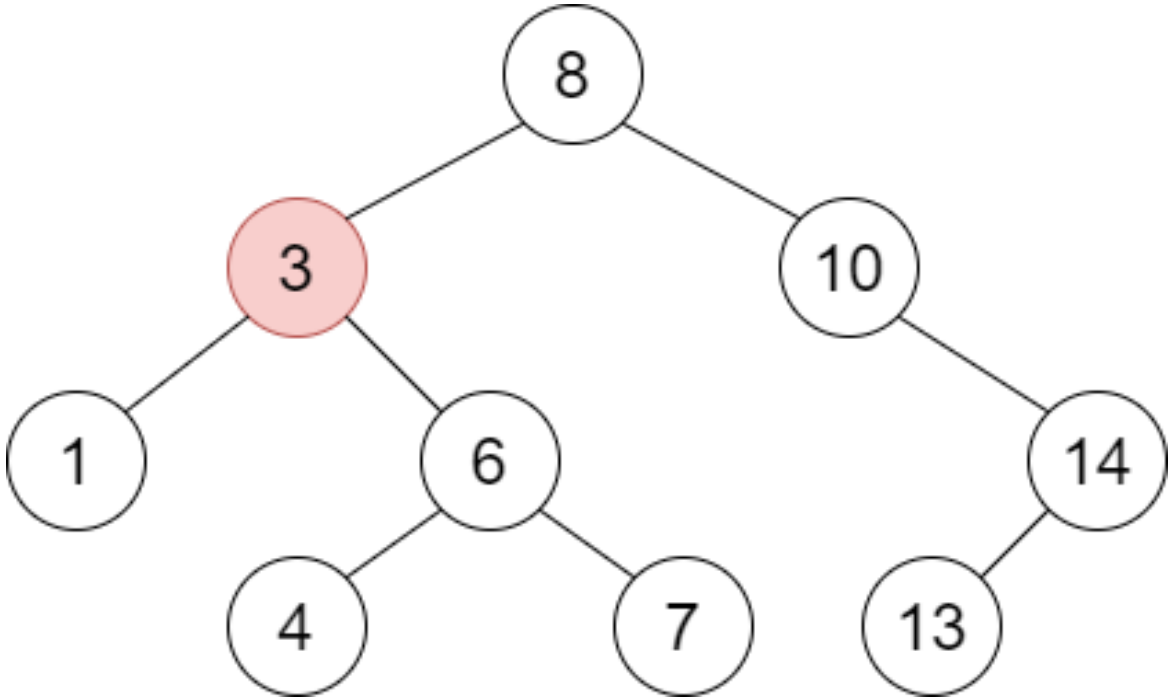
One Child



- If the node to be removed has one child, replace it with its child

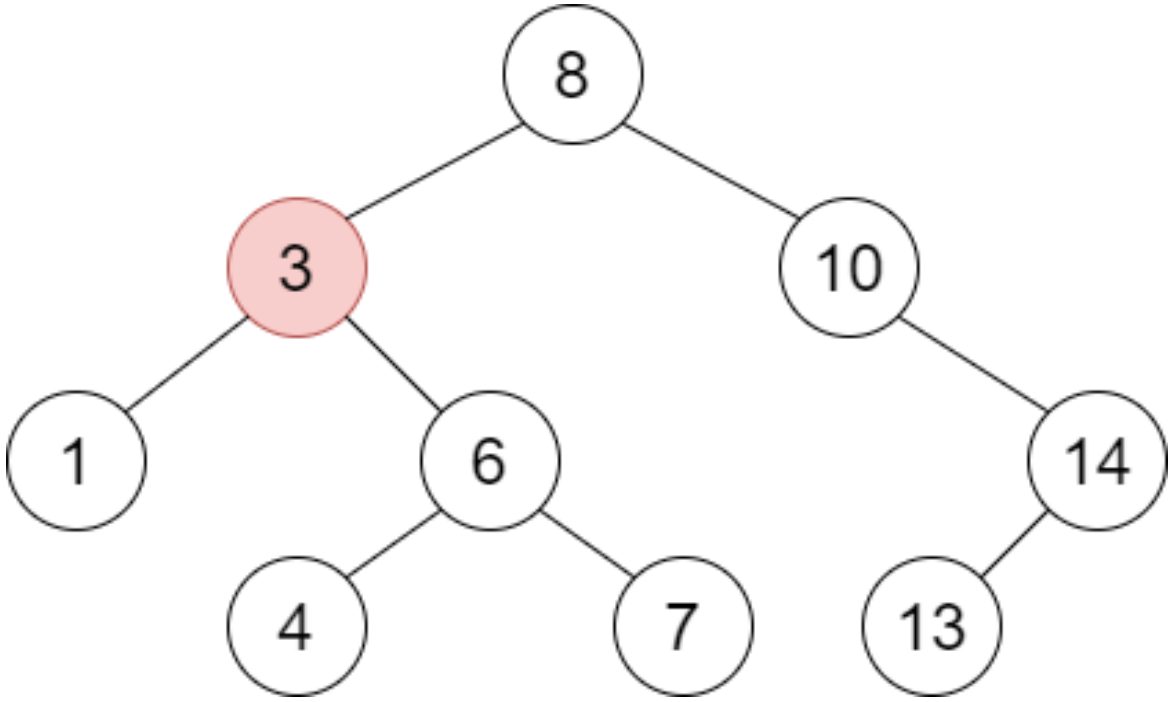
Deletion

Two Children



Deletion

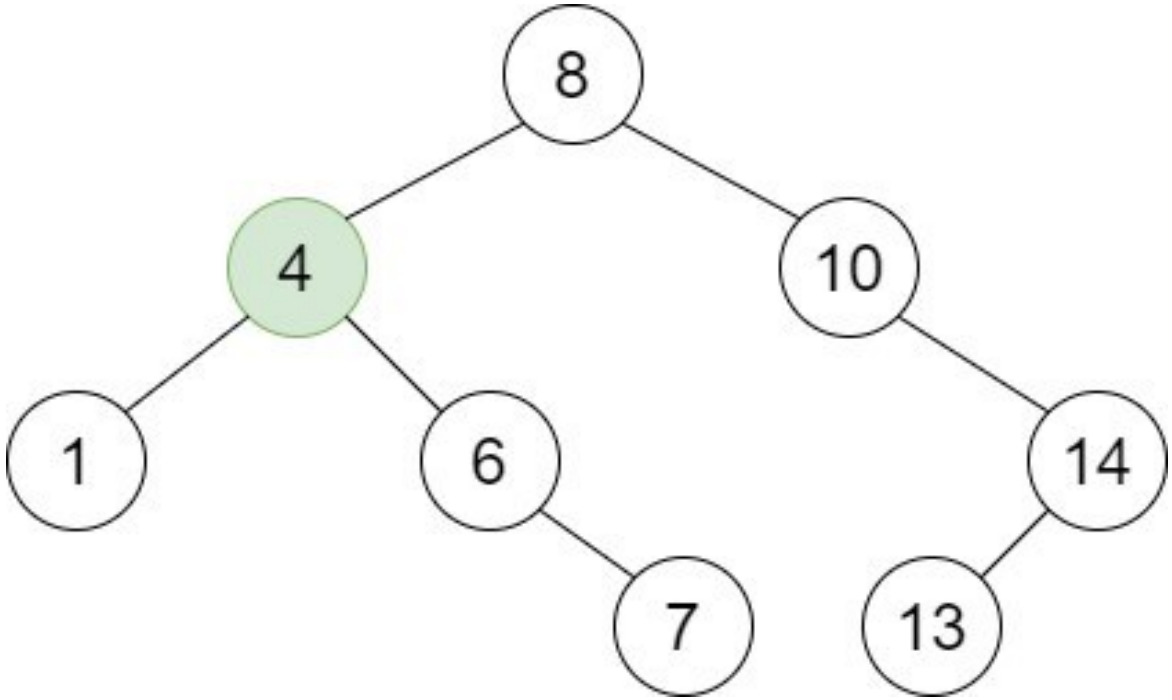
Two Children



- If the node to be removed has two children
  - Find the minimum node in the right subtree (or the maximum in the left subtree)
  - Replace this node with its right (left) child
  - Change the value of the node to be removed to the value of this node

Deletion

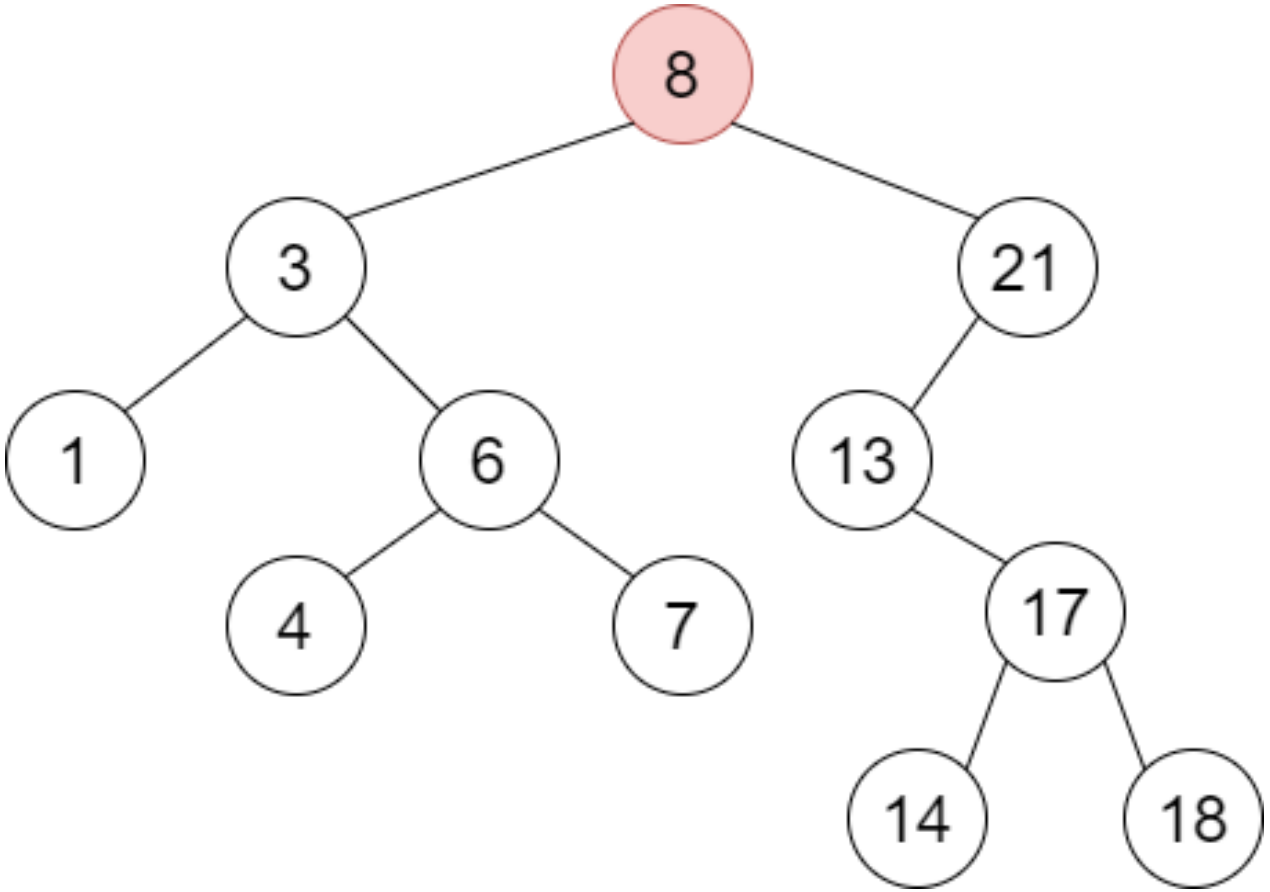
Two Children



- If the node to be removed has two children
  - Find the minimum node in the right subtree (or the maximum in the left subtree)
  - Replace this node with its right (left) child
  - Change the value of the node to be removed to the value of this node

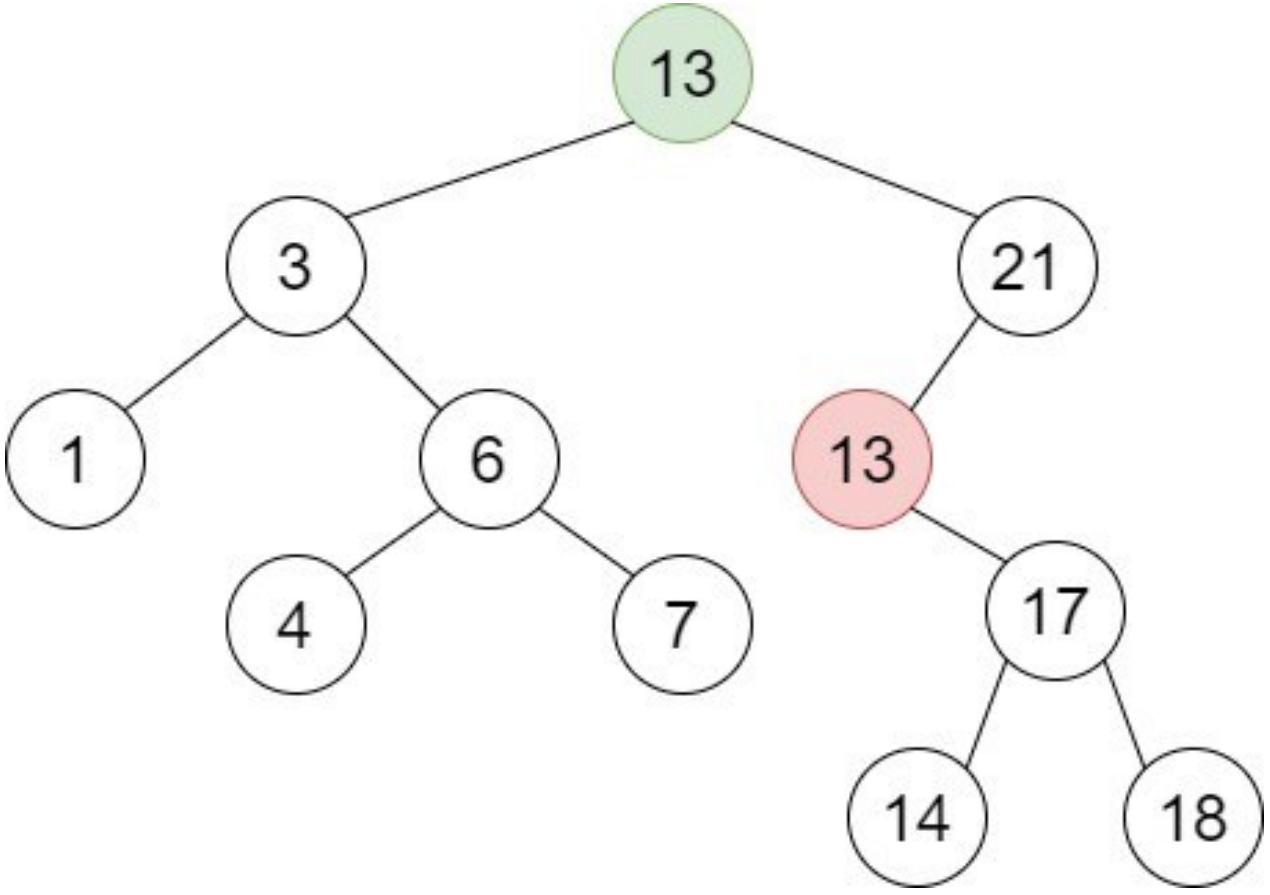
Deletion (Cascading)

Two Children



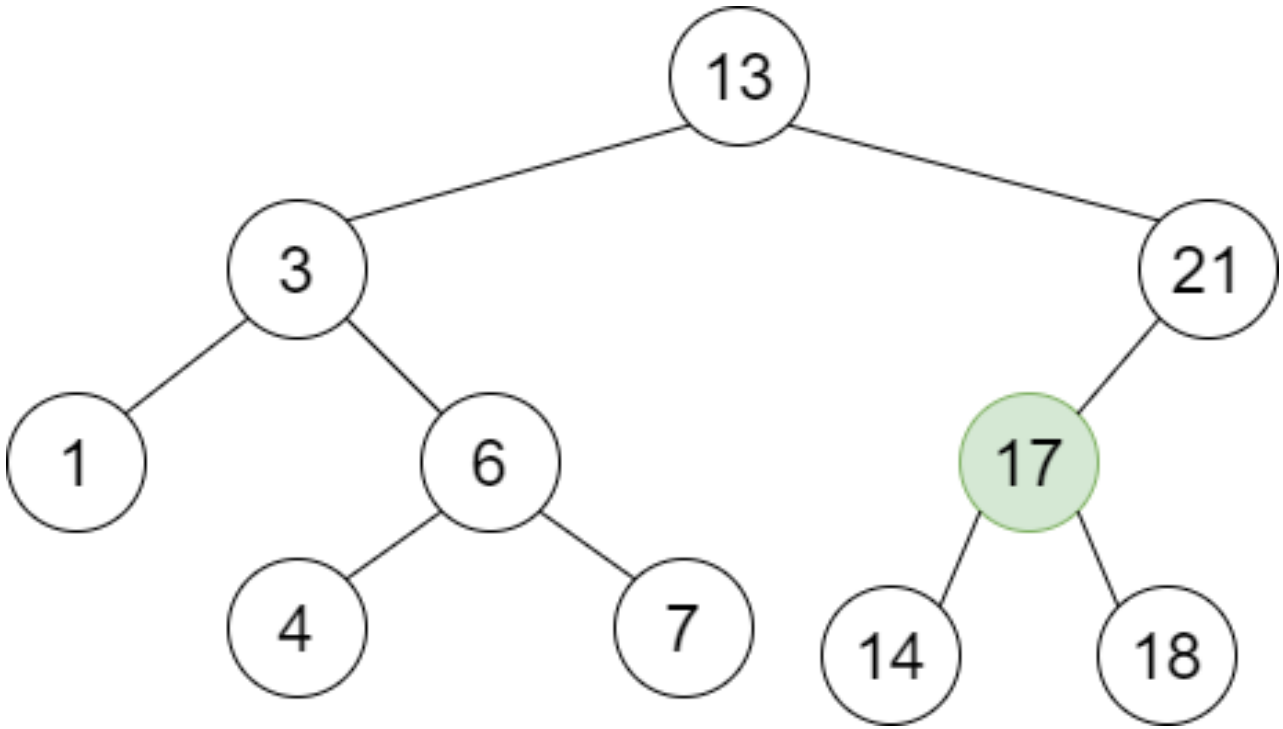
Deletion

Two Children



Deletion

Two Children



How to construct a BST?

Runtimes?

Project 1

Project 1

Just do it!



Project 1



Get it Done!