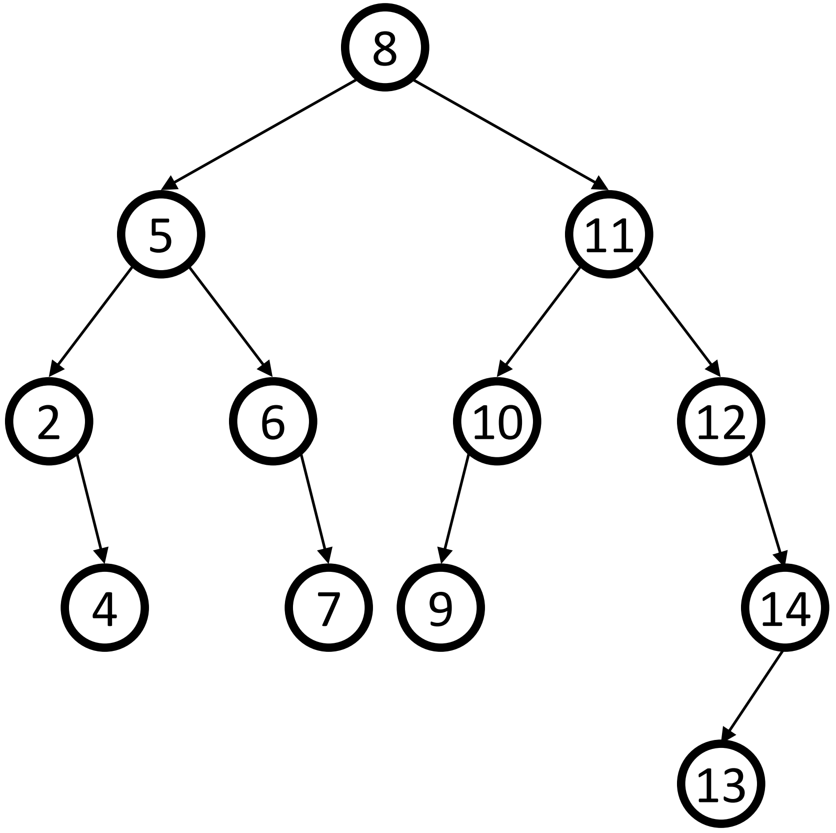
SNS COLLEGE OF TECHNOLOGY
(AUTONOMOUS), COIMBATORE - 35

Binary Search Tree (BST) Data Structure

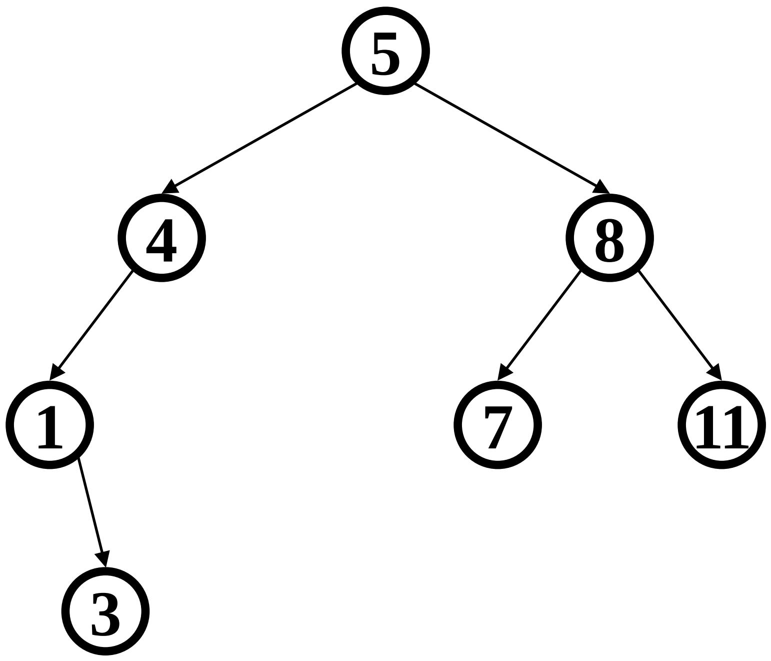
Binary Search Tree (BST) Data Structure

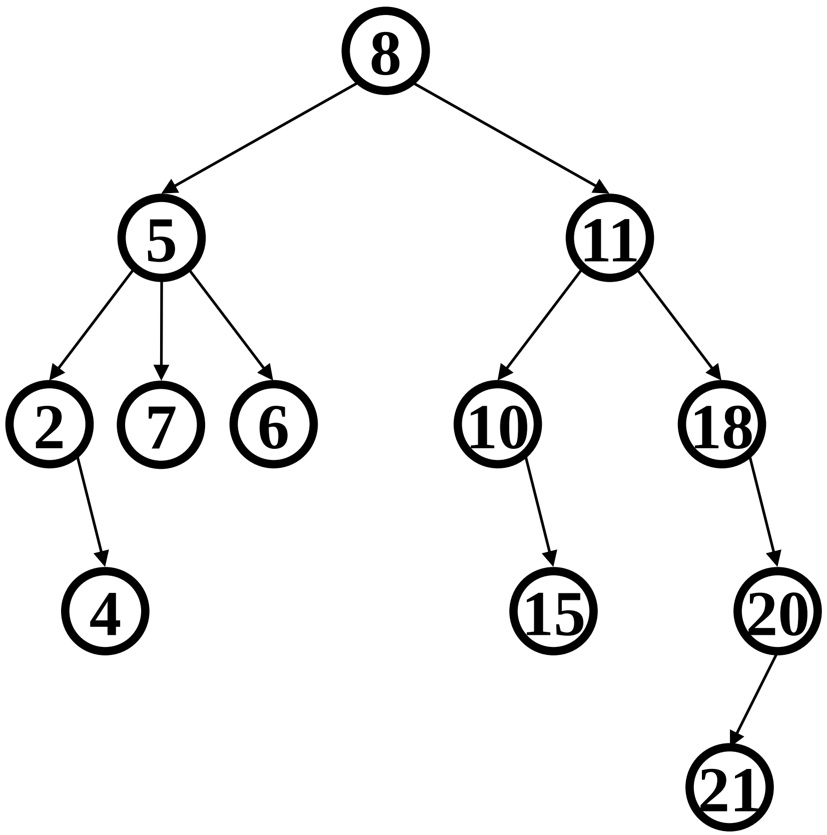
- Structure property (binary tree)
  - Each node has ≤ 2 children
  - Result: keeps operations simple
- Order property
  - Result: straight-forward to find any given value



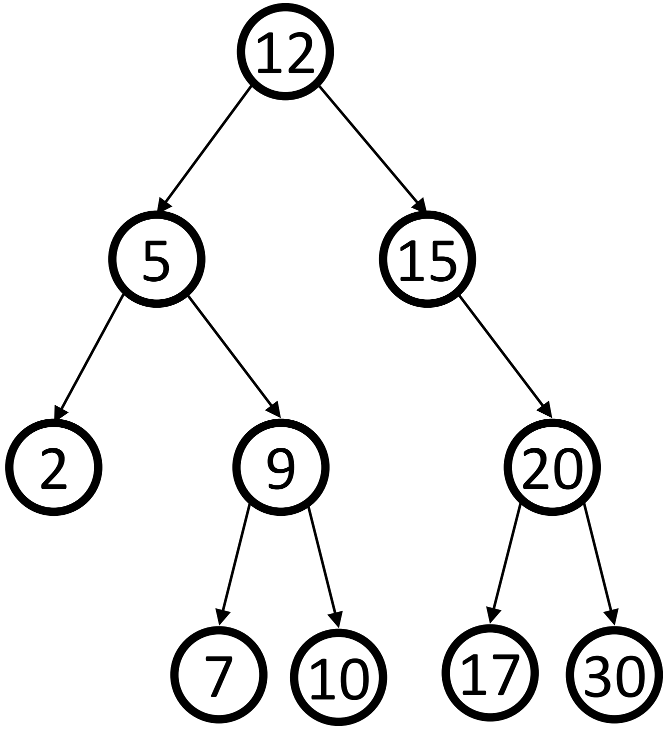
A binary *search* tree is a type of binary tree (but not all binary trees are binary search trees!)

Practice: are these BSTs?

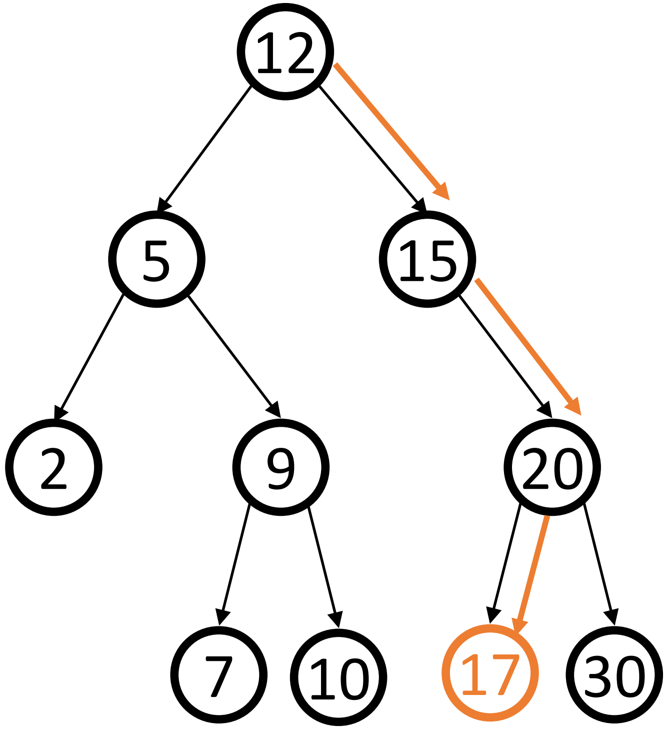




How do we `find(value)` in BST's?



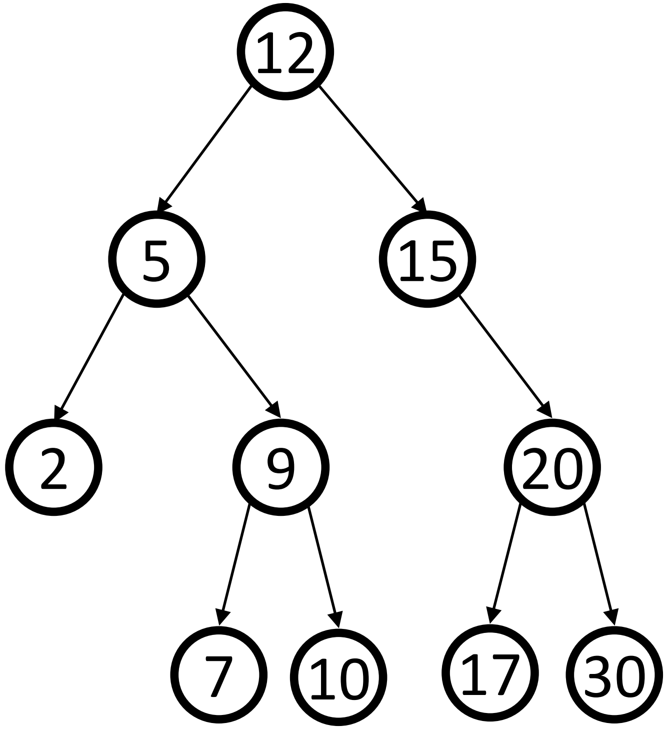
find in BST: Recursive Version



```
Data find(Data value, Node root){
 if(root == null)
   return null;
 if(key < root.value)
   return find(value, root.left);
 if(key > root.value)
   return find(value, root.right);
 return root.value;
}
```

What is the running time?

`find` in BST: Iterative Version

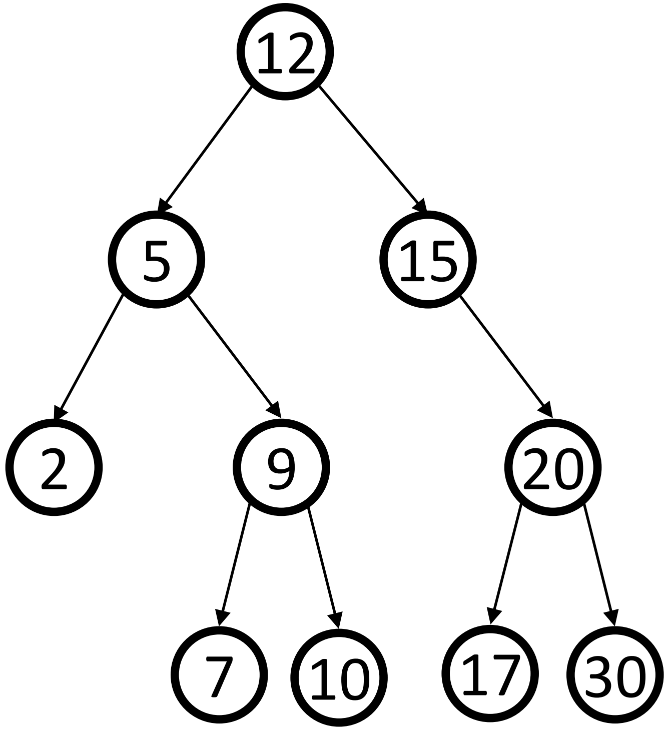


```
Data find(Object value, Node root){
 while(root != null
       && root.value != value) {
  if (value < root.value)
    root = root.left;
  else (value > root.value)
    root = root.right;
 }
 if(root == null)
    return null;
 return root.value;
}
```

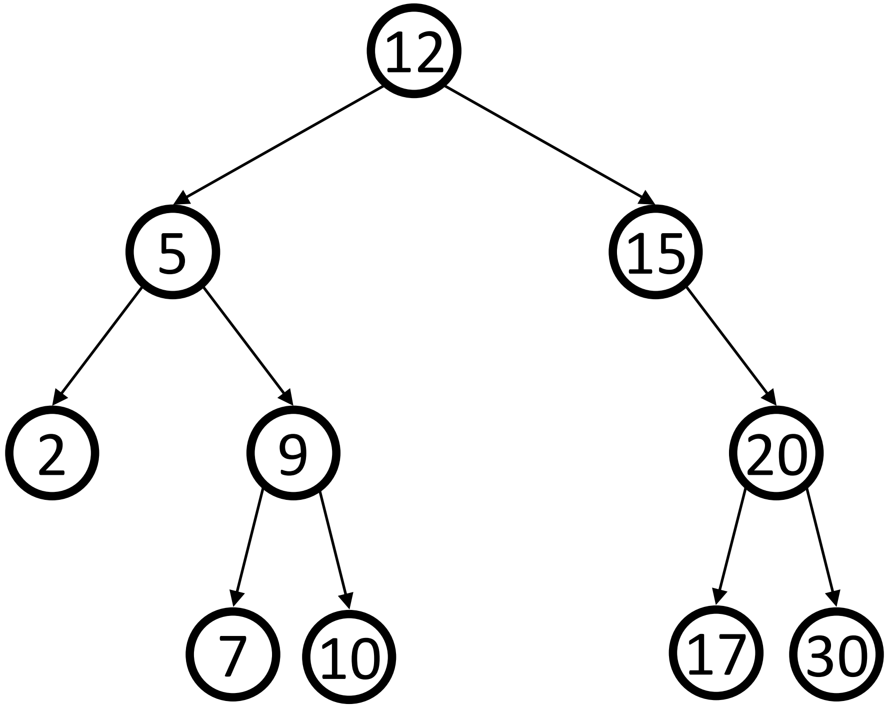
Other BST "Finding" Operations

`findMin`: Find *minimum* node

`findMax`: Find *maximum* node



insert in BST



```
insert(13)
insert(8)
insert(31)
```

Worst-case running time:

Practice with `insert`, primer for `delete`

Start with an empty tree. Insert the following values, in the given order:

`14, 2, 5, 20, 42, 1, 4, 16`

Then, changing as few nodes as possible, delete the following in order:

`42, 14`

What would the root of the resulting tree be?

A. `2`

B. `4`

C. `5`

D. `16`

`delete` in BST

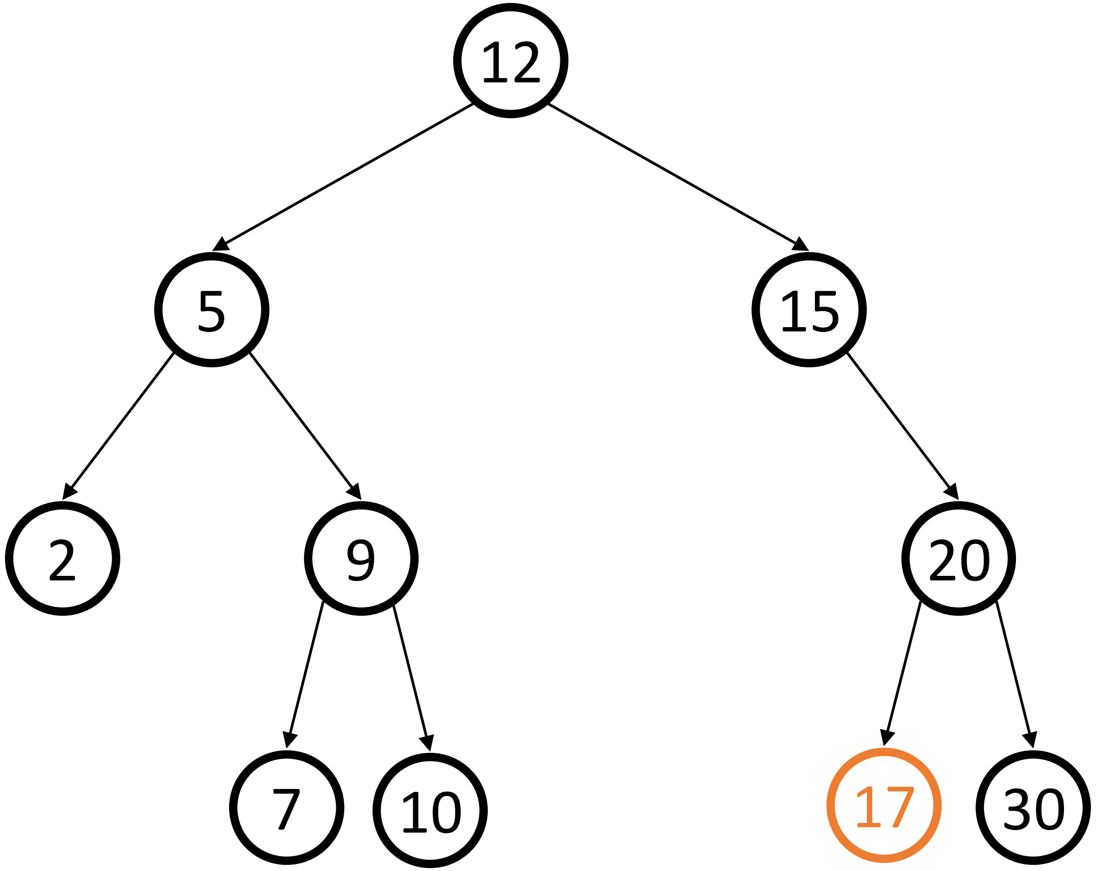
- Why might `delete` be harder than `insert`?

- Basic idea:

- Three potential cases to fix:

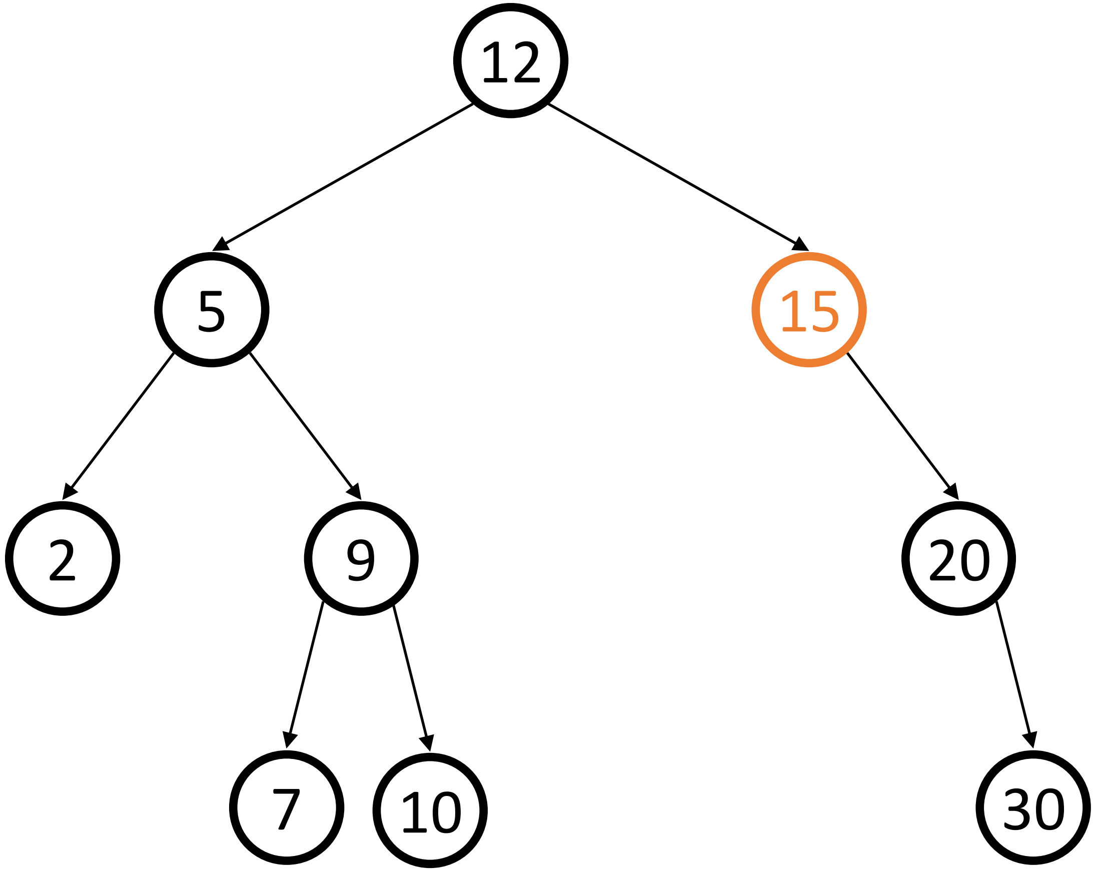
delete case: Leaf

delete(17)



delete case: One Child

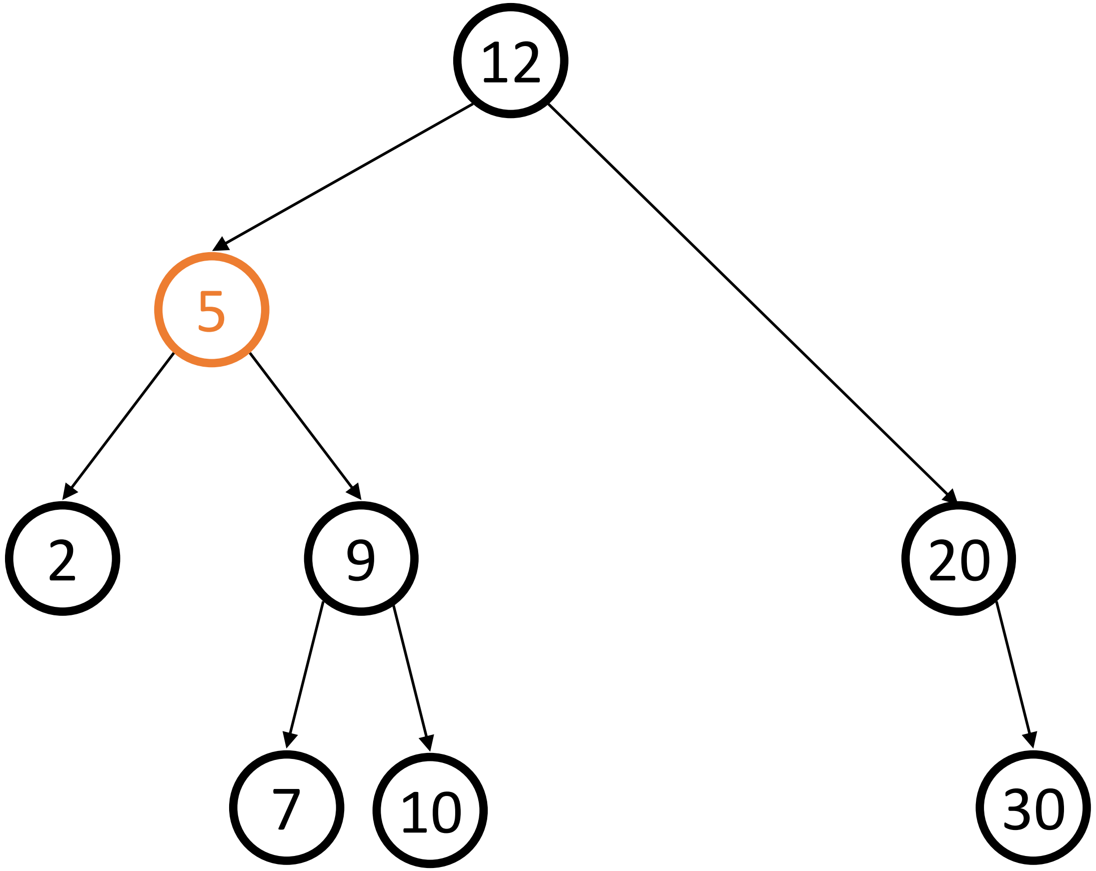
delete(15)



delete case: Two Children

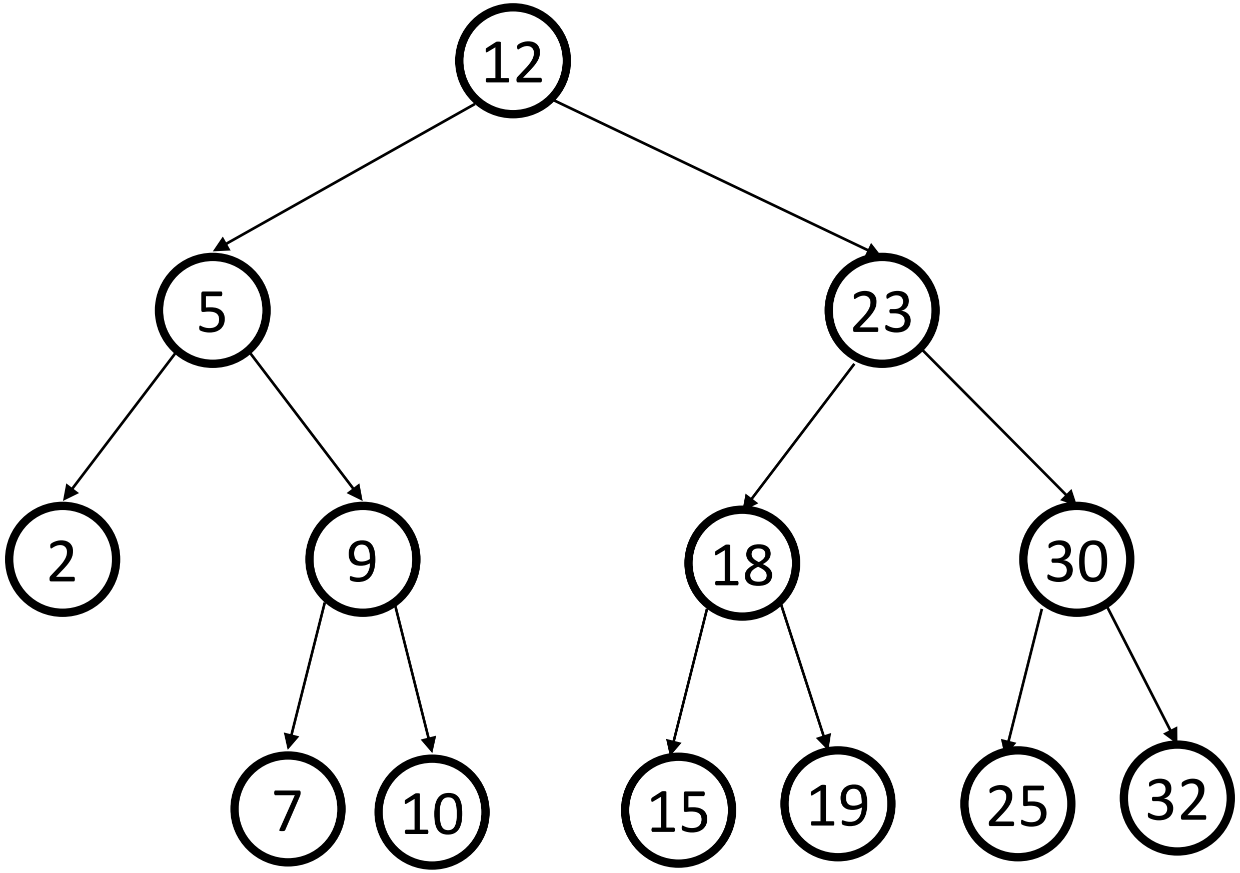
delete(5)

What can we replace 5 with?



`delete` case: Two Children (example #2)

`delete(23)`



`delete` through Lazy Deletion

- Lazy deletion can work well for a BST
  - Simpler
  - Can do "real deletions" later as a batch
  - Some inserts can just "undelete" a tree node

- But
  - Can waste space and slow down find operations
  - Make some operations more complicated:
    - e.g., **`findMin`** and **`findMax`**?

buildTree for BST

Let’s consider `buildTree` (insert values starting from an empty tree)

Insert values 1, 2, 3, 4, 5, 6, 7, 8, 9 into an empty BST

- If inserted in given order, what is the tree?

- What big-O runtime for buildTree on this sorted input?

- Is inserting in the reverse order any better?

buildTree for BST

Insert values 1, 2, 3, 4, 5, 6, 7, 8, 9 into an empty BST

What we if could somehow re-arrange them

- median first, then left median, right median, etc.
  5, 3, 7, 2, 1, 4, 8, 6, 9

- What tree does that give us?

- What big-O runtime?