

Binary Search Trees: Insert

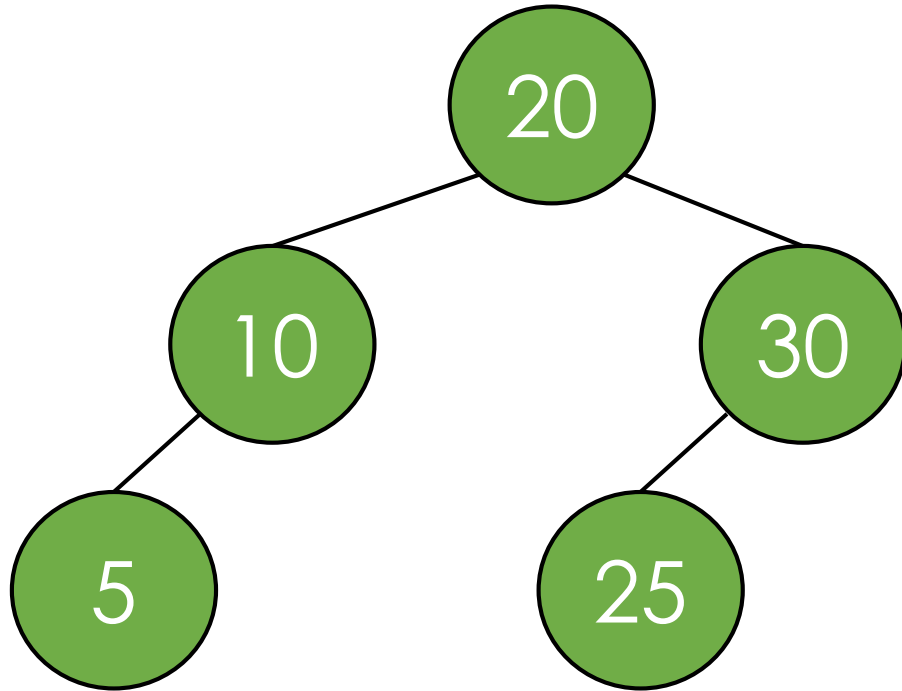


This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/)
by Christine Alvarado, Mia Minnes, and Leo Porter, 2015.

By the end of this video you will be able to...

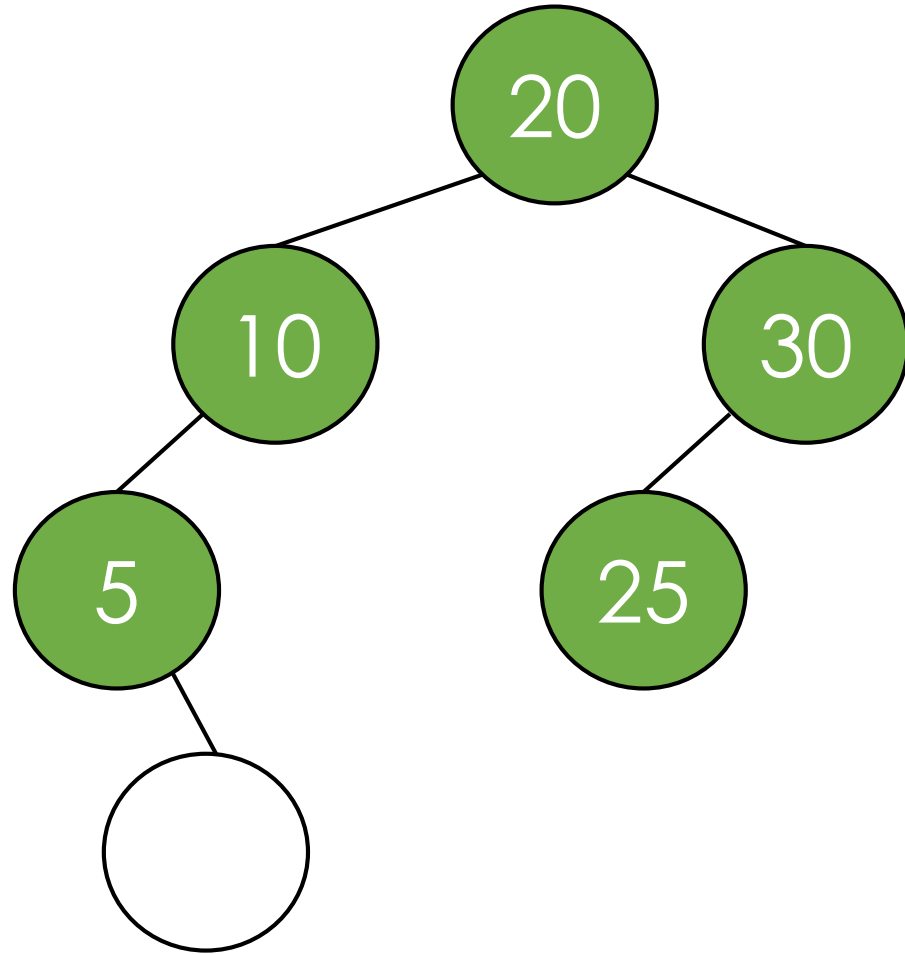
- Insert an item into a Binary Search Tree

Binary Search Tree - Insertion



**Where should we
insert 7?**

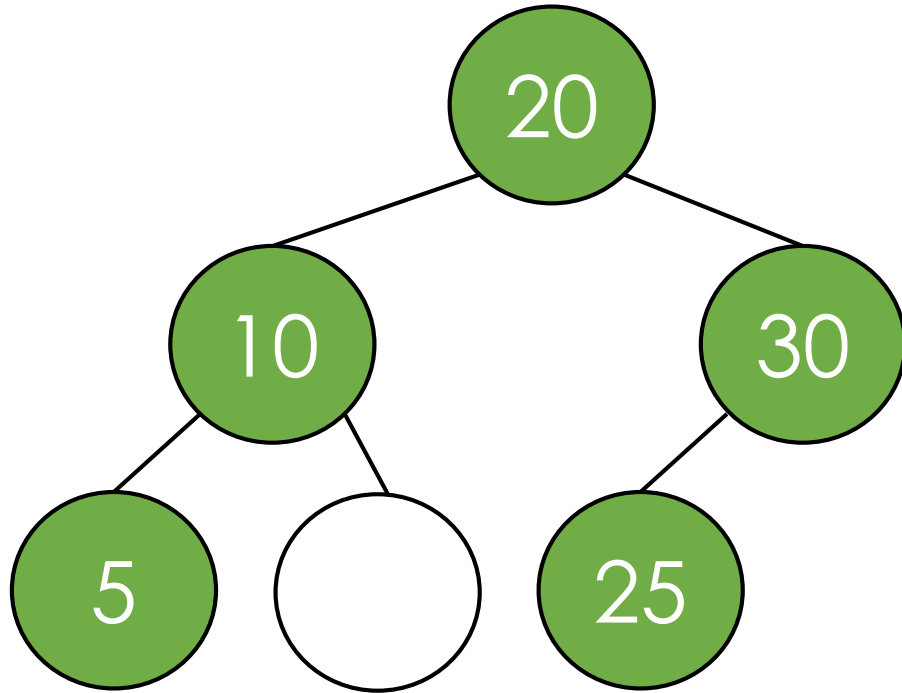
Binary Search Tree - Insertion



Option A

**Where should we
insert 7?**

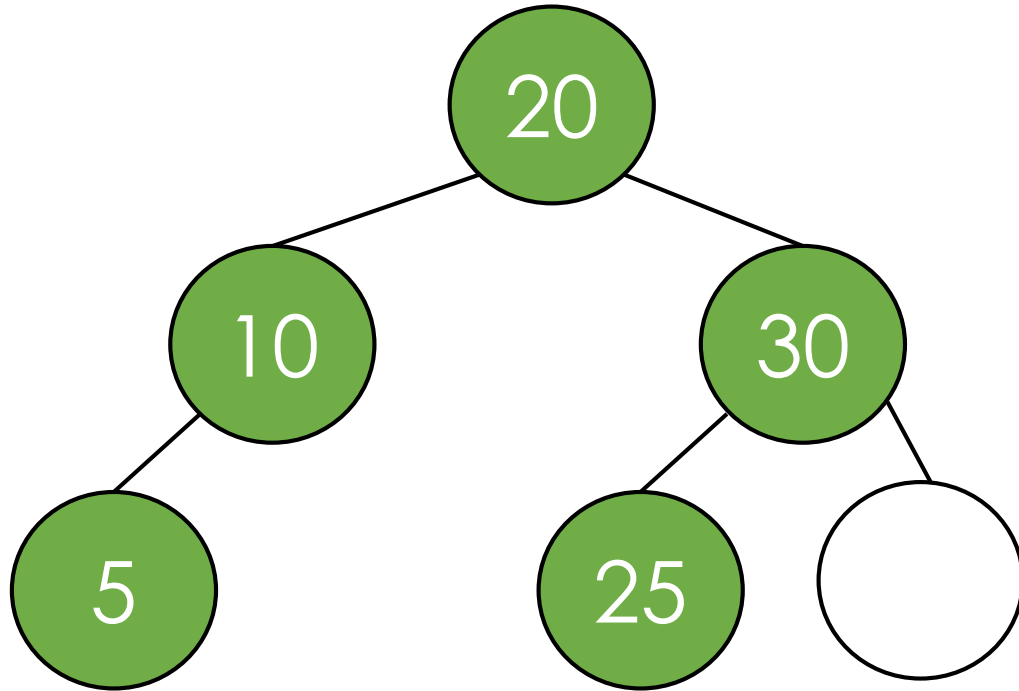
Binary Search Tree - Insertion



Option B

**Where should we
insert 7?**

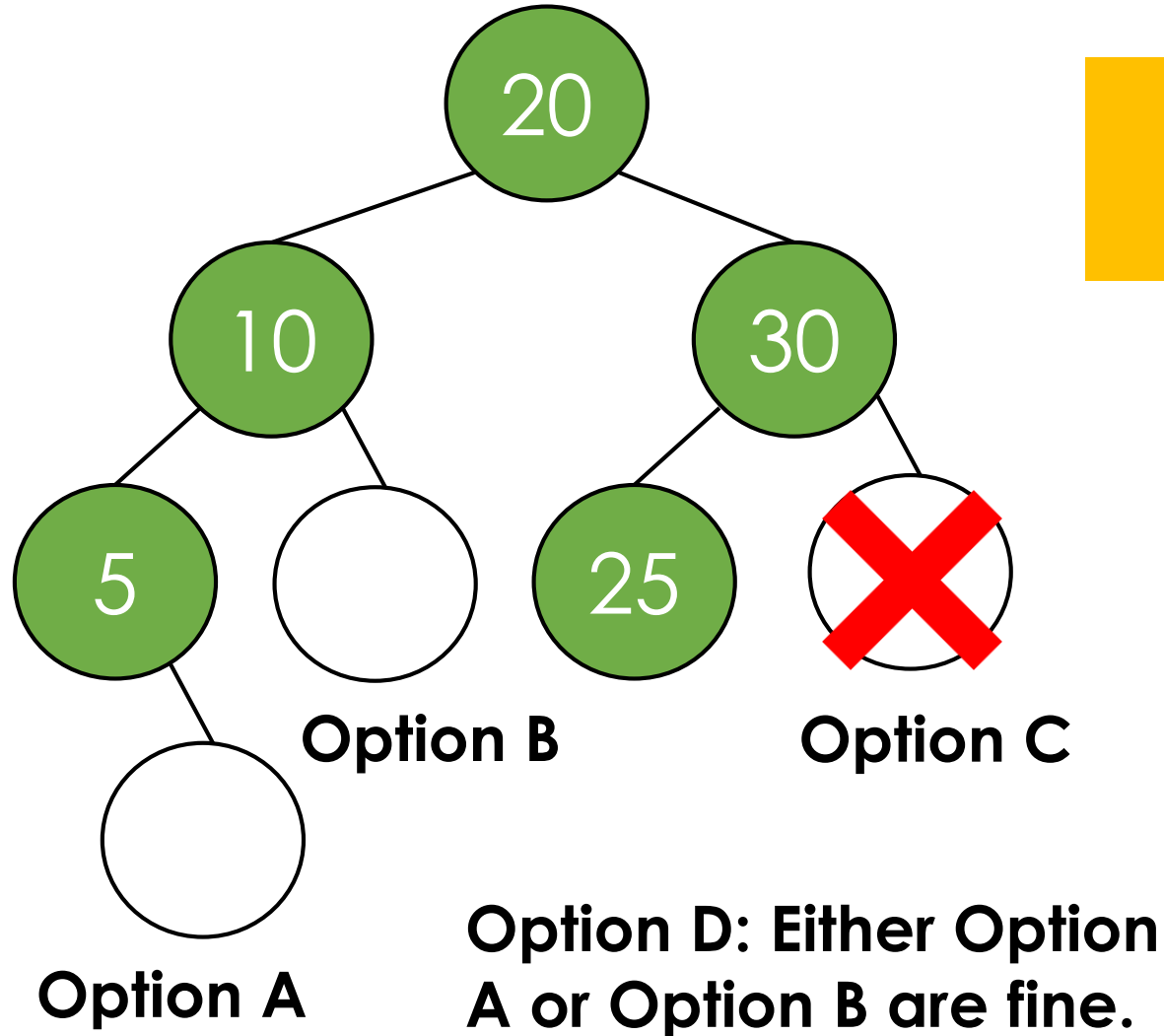
Binary Search Tree - Insertion



**Where should we
insert 7?**

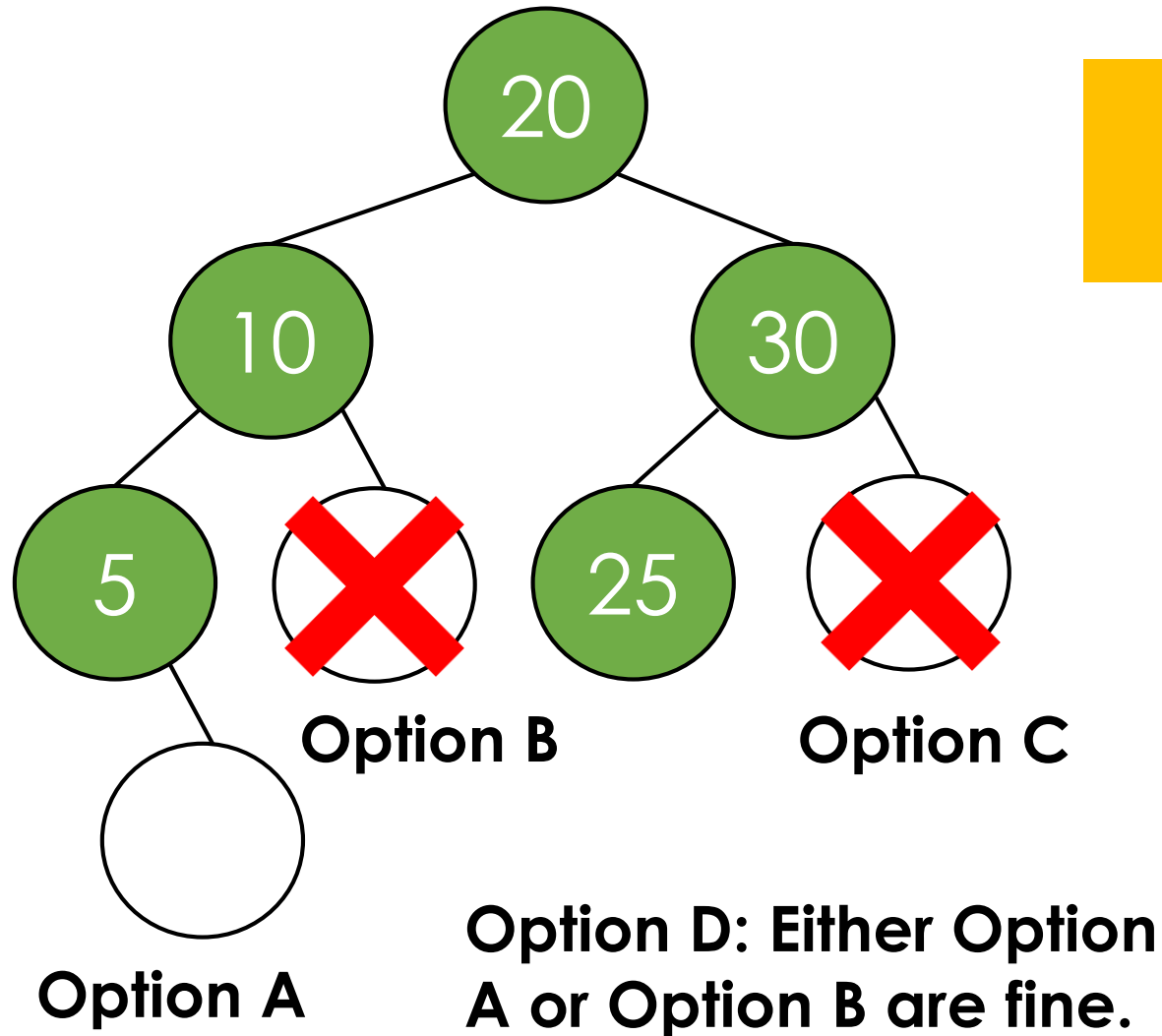
Option C

Binary Search Tree - Insertion



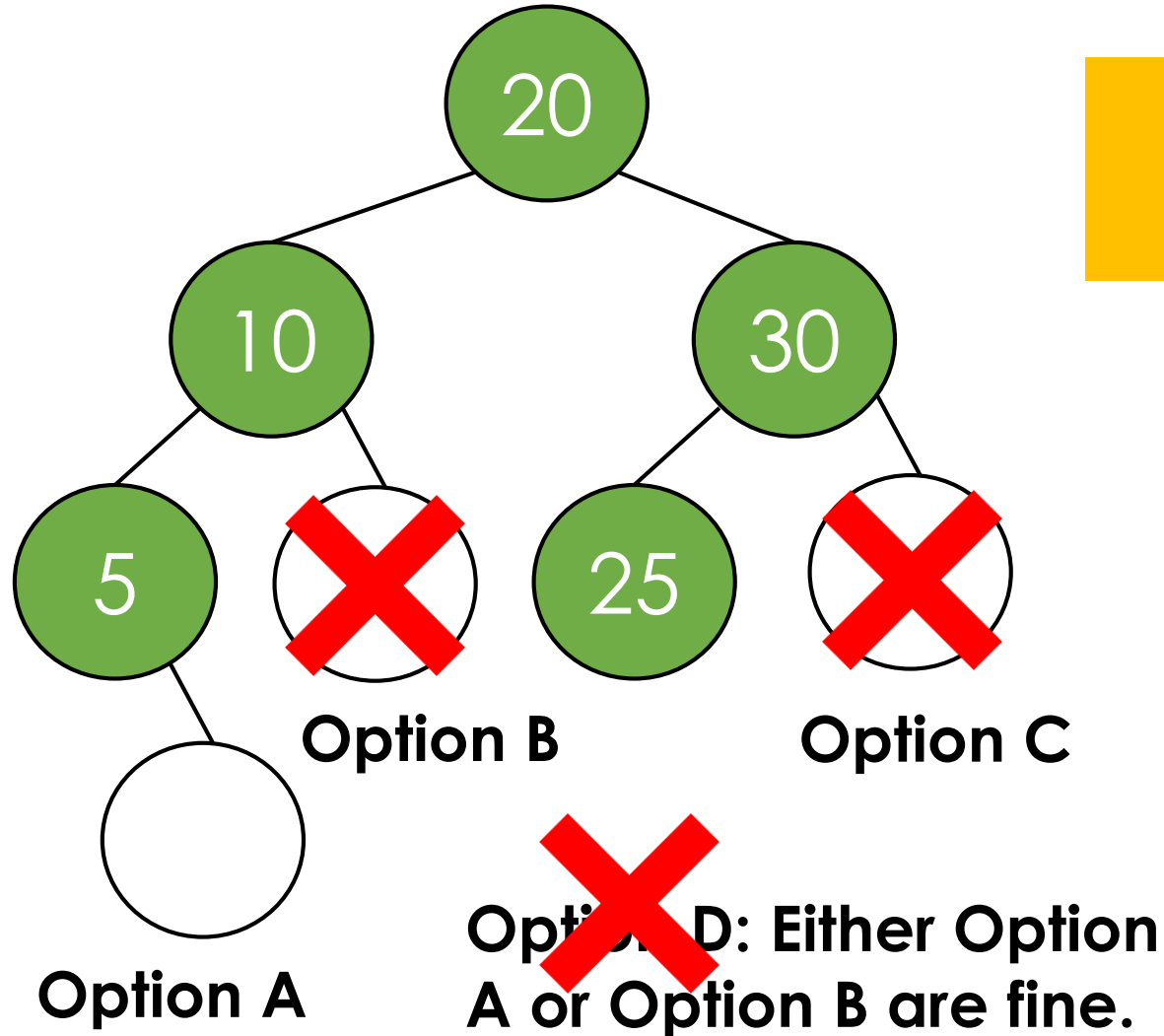
**Where should we
insert 7?**

Binary Search Tree - Insertion



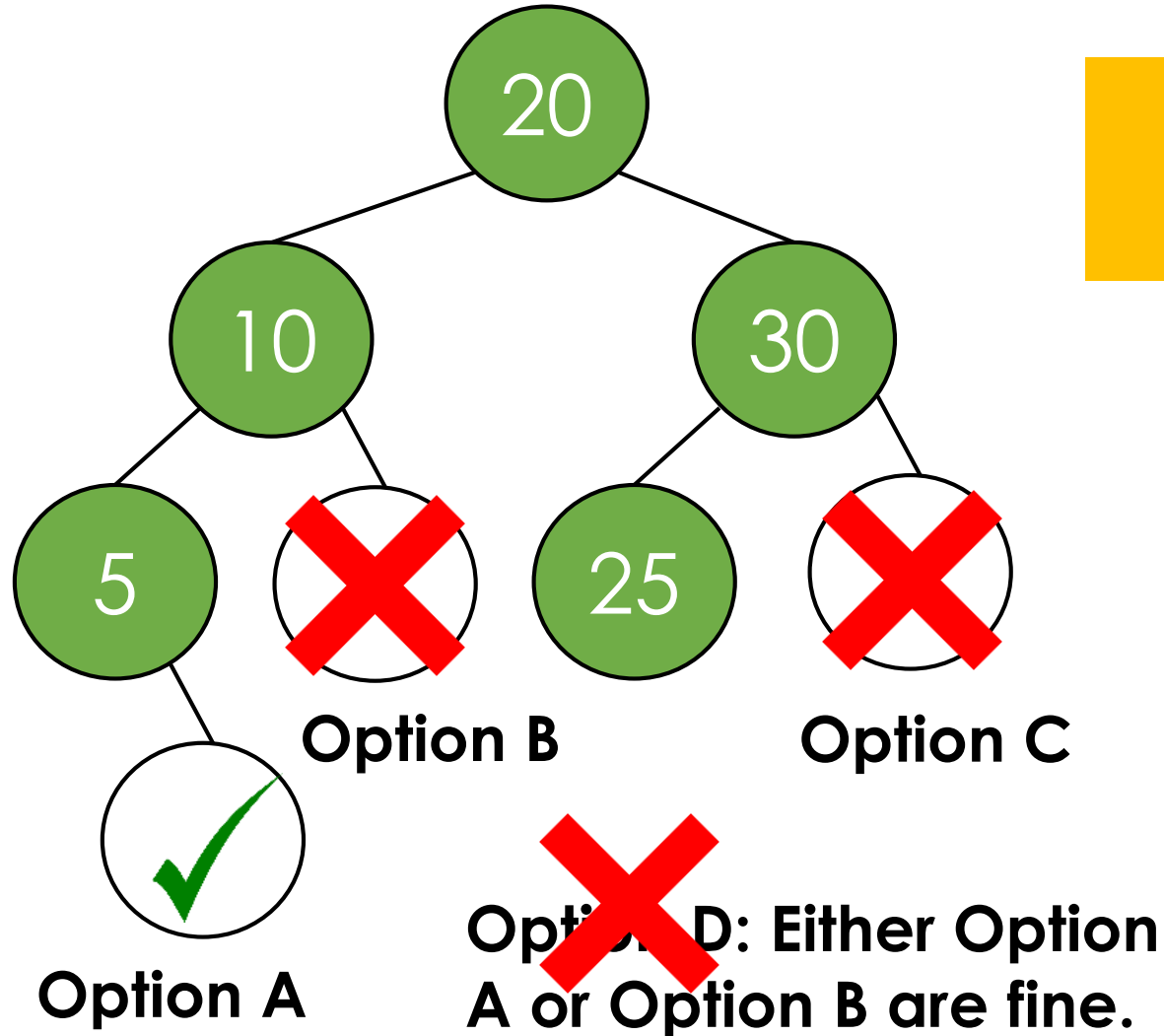
**Where should we
insert 7?**

Binary Search Tree - Insertion



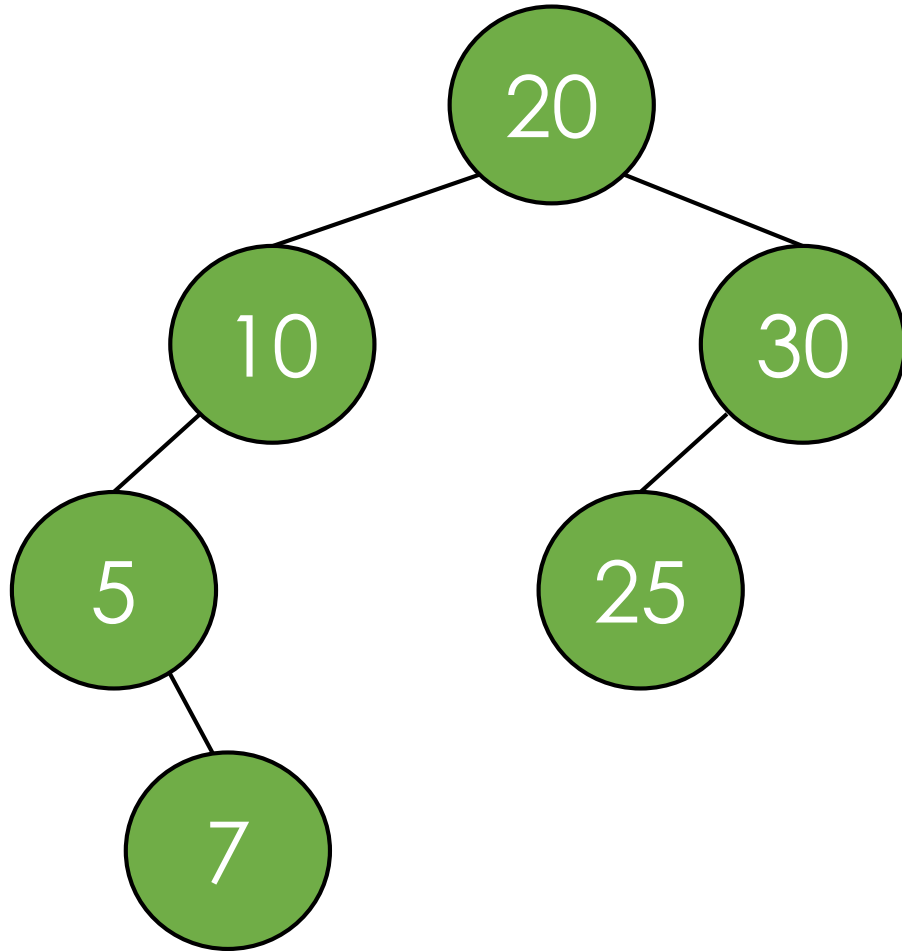
Where should we
insert 7?

Binary Search Tree - Insertion

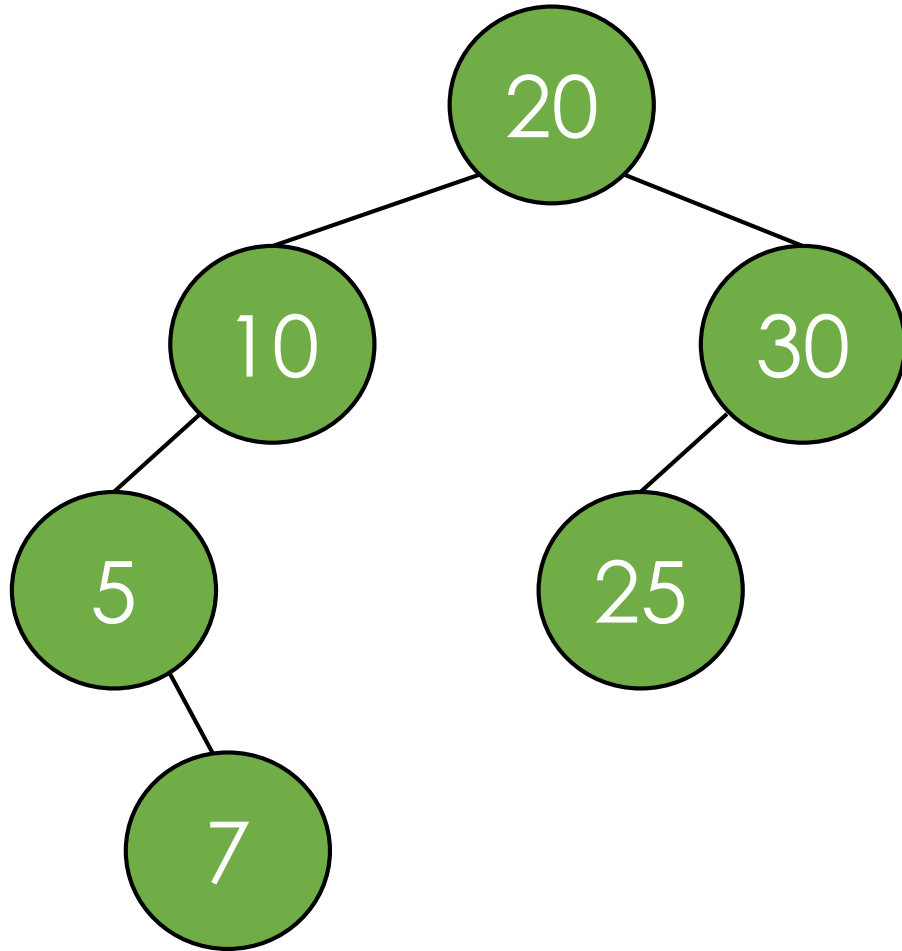


Where should we
insert 7?

Binary Search Tree - Insertion

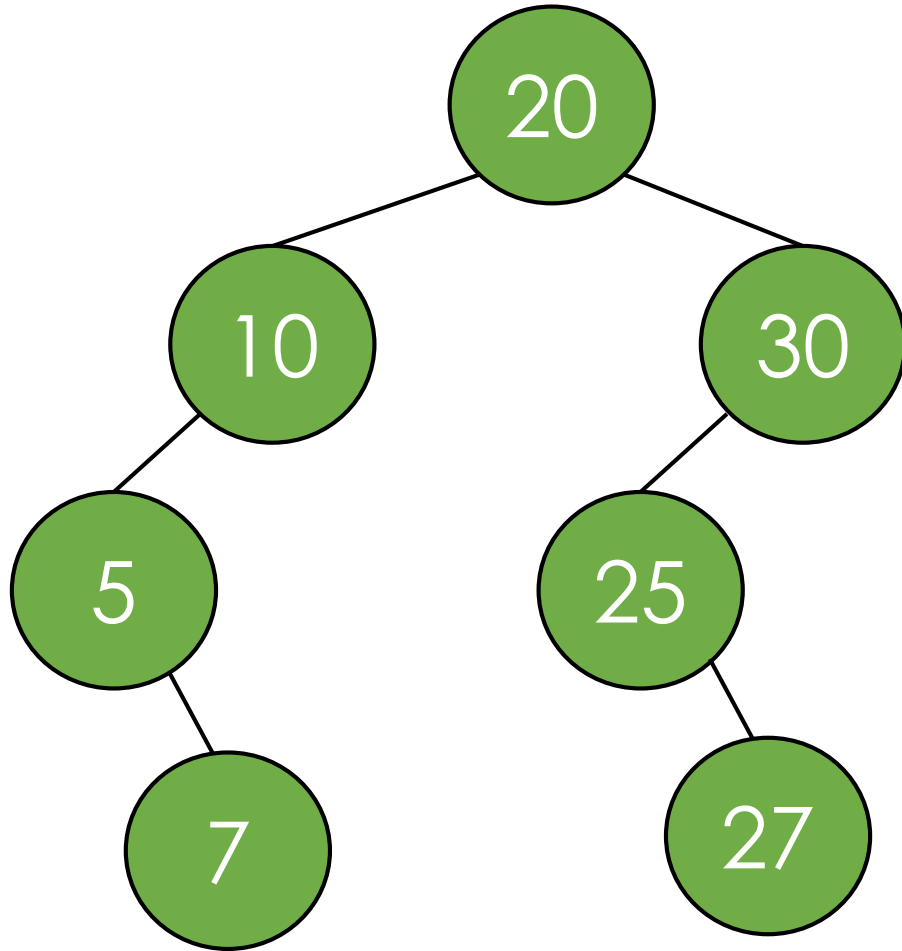


Binary Search Tree - Insertion



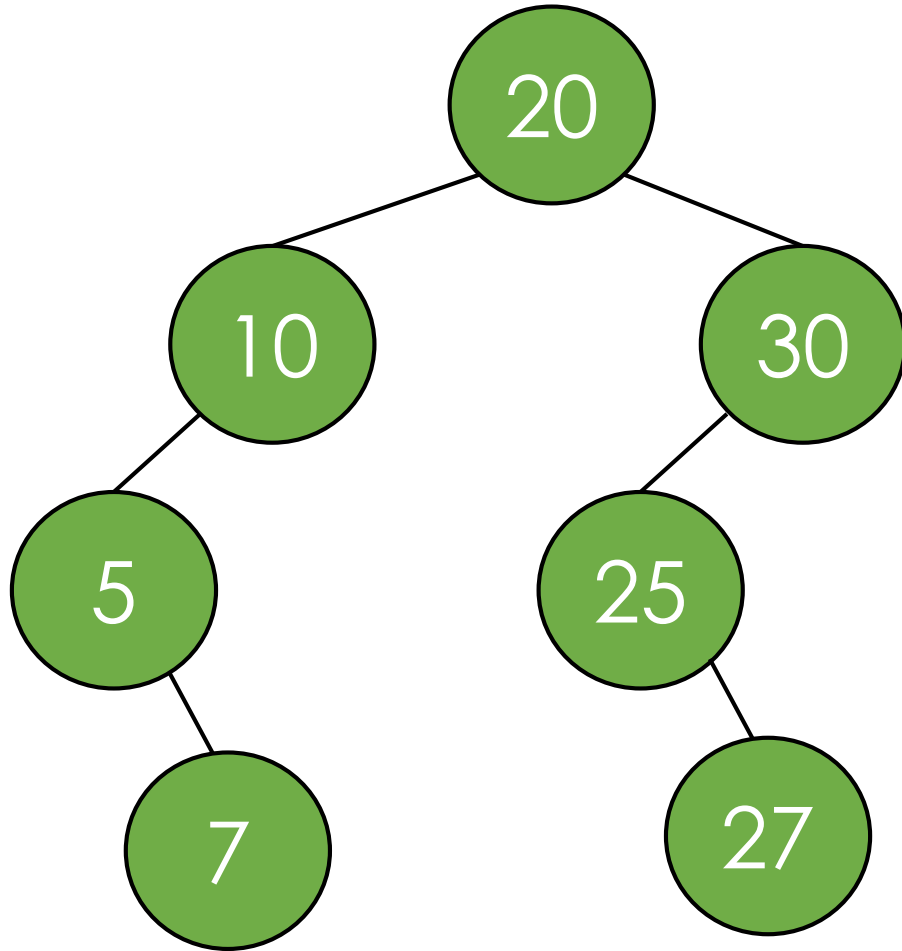
Insert 27?

Binary Search Tree - Insertion



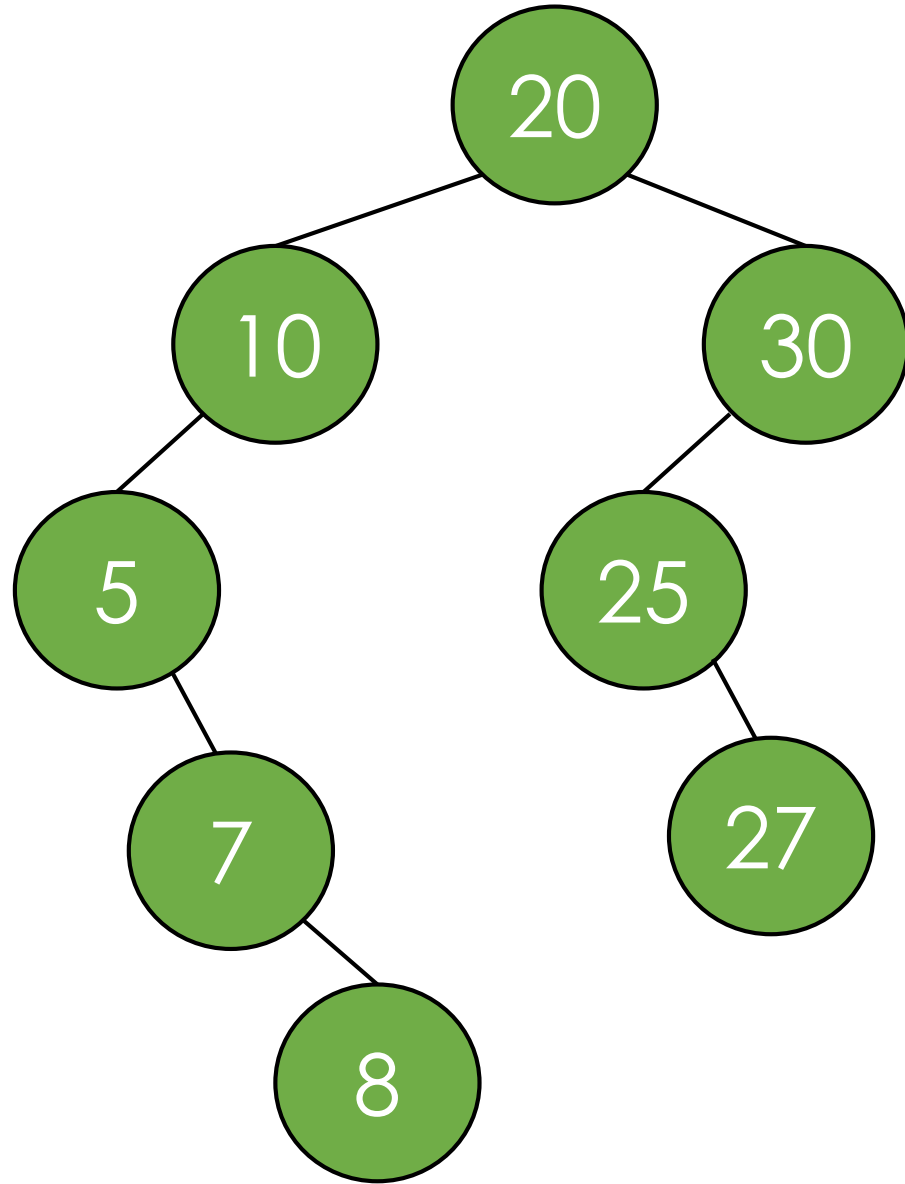
Insert 27?

Binary Search Tree - Insertion



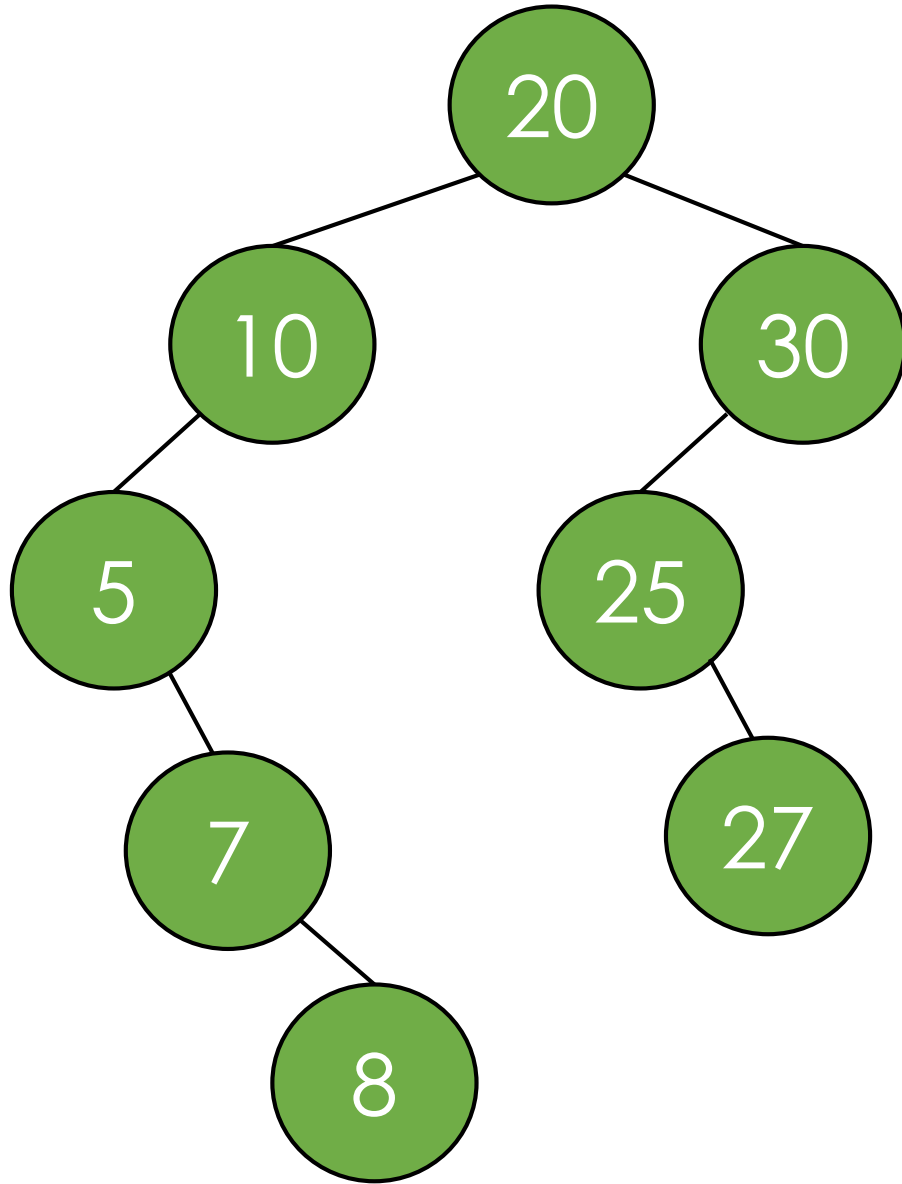
Insert 8?

Binary Search Tree - Insertion



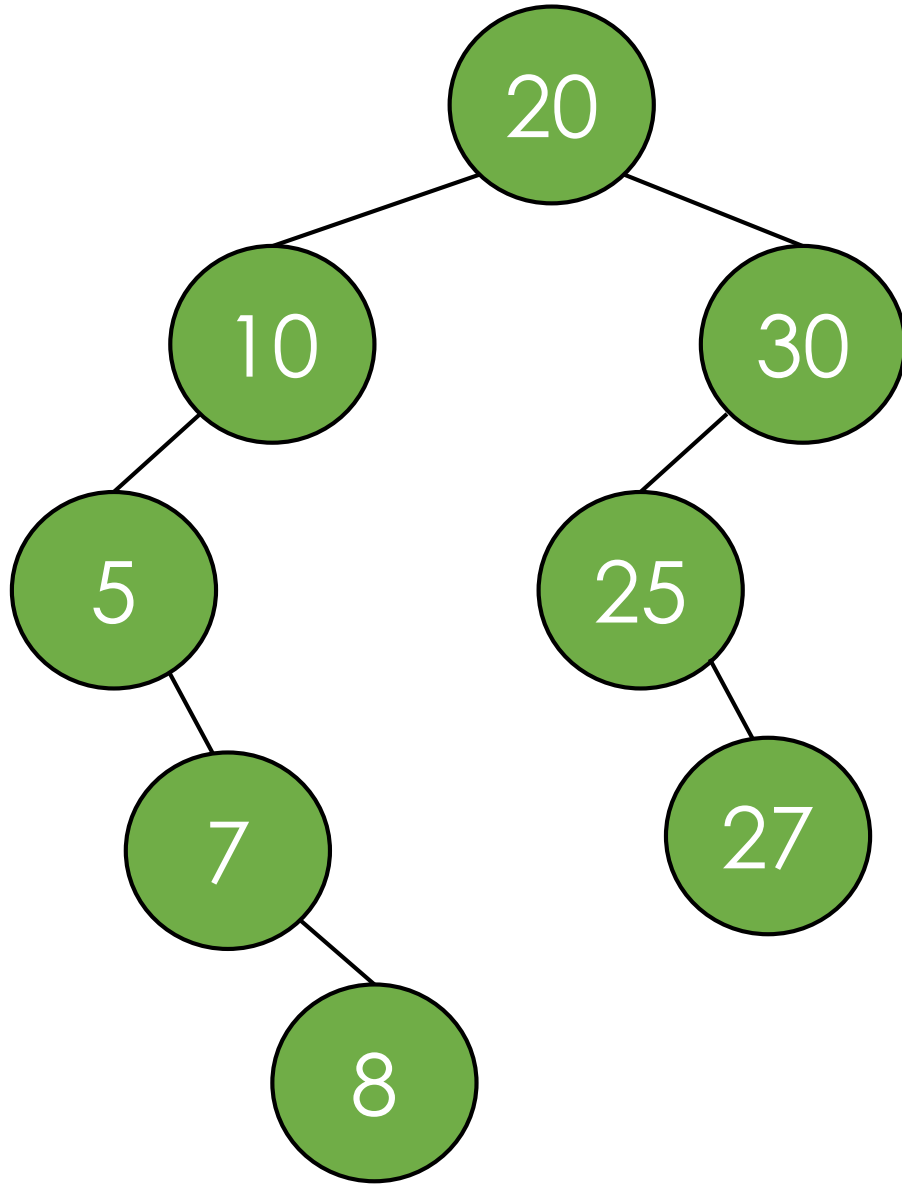
Insert 8?

Binary Search Tree - Insertion



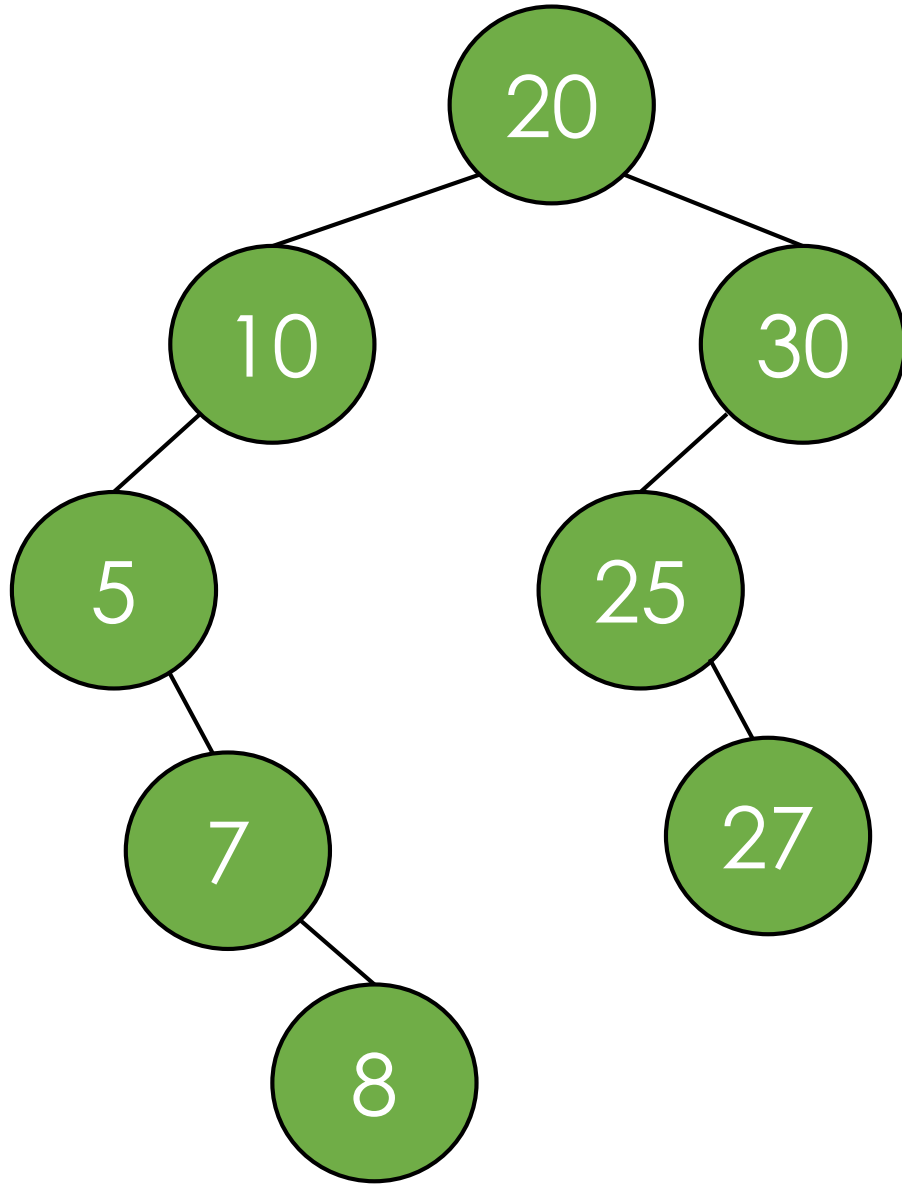
There's no rule that
BSTs will be **full** trees
(or **balanced**).

Binary Search Tree - Insertion



Again, this is solved cleanly with either recursion or iteration.

Binary Search Tree - Insertion



Again, this is solved cleanly with either recursion or iteration.

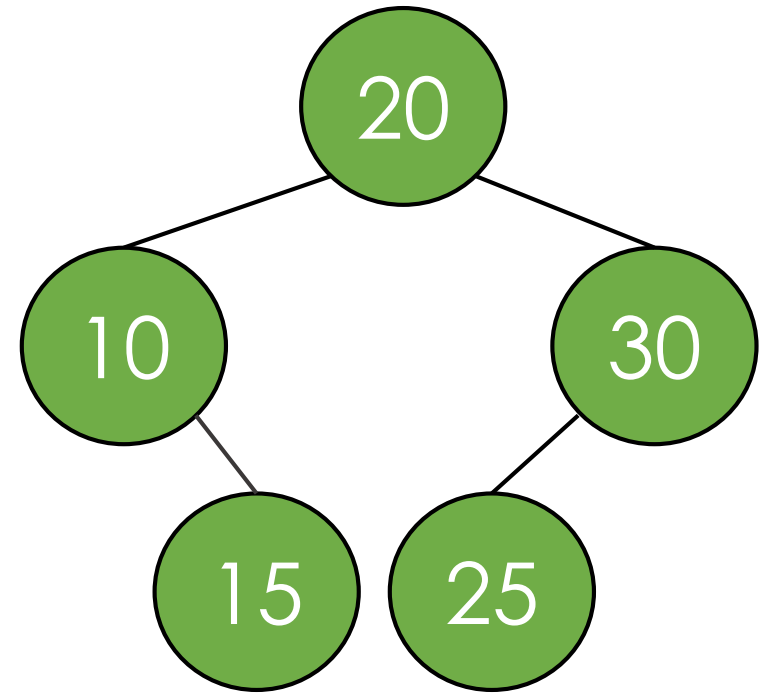
We'll show you how to do it iteratively in the support video

Next step

- Deletion in a BST

BST: Insert Algorithm

```
insert(item, node):  
    if (node.val == item)  
        return false  
    else if (node.val < item)  
        if (node.right == null)  
            add as right child, return true  
        insert(item, node.right)  
    else // node.val > item  
        if (node.left == null)  
            add as left child, return true  
        insert(item, node.left)
```



BST: Insert Algorithm

insert(item, node):

if (node.val == item)

return false

else if (node.val < item)

if (node.right == null)

add as right child, return true

insert(item, node.right)

else // node.val > item

if (node.left == null)

add as left child, return true

insert(item, node.left)



**If BST doesn't
allow duplicates**

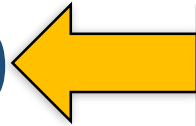
BST: Insert Algorithm

insert(item, node):

if (node.val == item)

return false

else if (node.val < item)



add to right tree

if (node.right == null)

add as right child, return true

insert(item, node.right)

else // node.val > item

if (node.left == null)

add as left child, return true

insert(item, node.left)

BST: Insert Algorithm

insert(item, node):

if (node.val == item)

return false

else if (node.val < item)

if (node.right == null)

add as right child, return true


insert(item, node.right)

else // node.val > item

if (node.left == null)

add as left child, return true

insert(item, node.left)



**If the right tree is
empty, add
node there**

BST: Insert Algorithm

```
insert(item, node):  
    if (node.val == item)  
        return false  
    else if (node.val < item)  
        if (node.right == null)  
            add as right child, return true  
        insert(item, node.right)  
    else // node.val > item  
        if (node.left == null)  
            add as left child, return true  
        insert(item, node.left)
```



**Insert into right
tree**

BST: Insert Algorithm

```
insert(item, node):  
    if (node.val == item)  
        return false  
    else if (node.val < item)  
        if (node.right == null)  
            add as right child, return true  
        insert(item, node.right)  
    else // node.val > item  
        if (node.left == null)  
            add as left child, return true  
        insert(item, node.left)
```



**Insert into left
(same idea)**