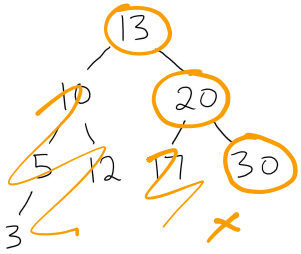
CSC148 - Investigating the Efficiency of Binary Search Trees

1. Recall our implementation of `BinarySearchTree.__contains__`:

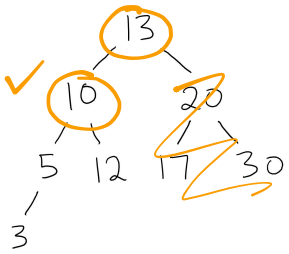
```
def __contains__(self, item: Any) -> bool:
    if self.is_empty():
        return False
    elif item == self._root:
        return True
    elif item < self._root:
        return self._left.__contains__(item)
    else:
        return self._right.__contains__(item)
```

Crucially, after comparing the item against the root of the BST, only *one* subtree is recursed into. To make sure you understand this idea, suppose we have the following BST, and perform three searches, for the items `25`, `10`, and `4`, respectively. Circle all the values that are compared against the target item when we do each search.

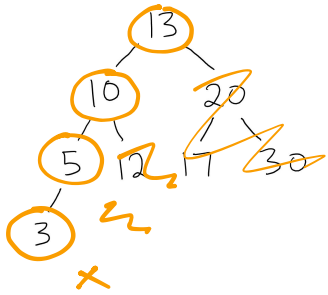
Search for `25`:



Search for `10`:



Search for `4`:

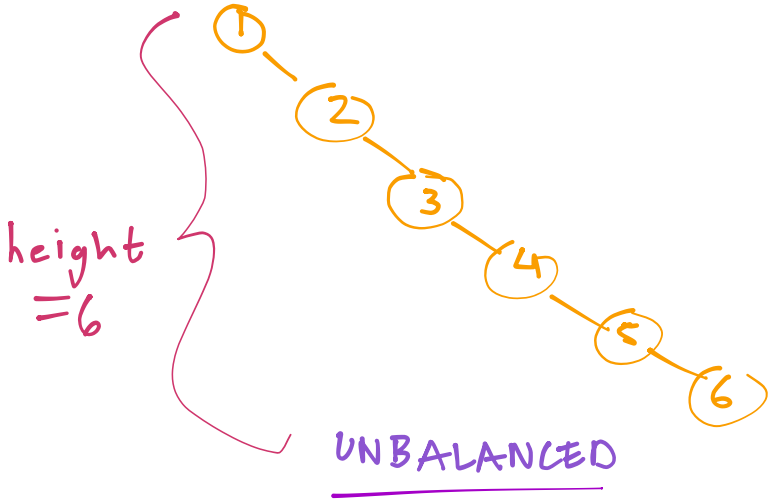


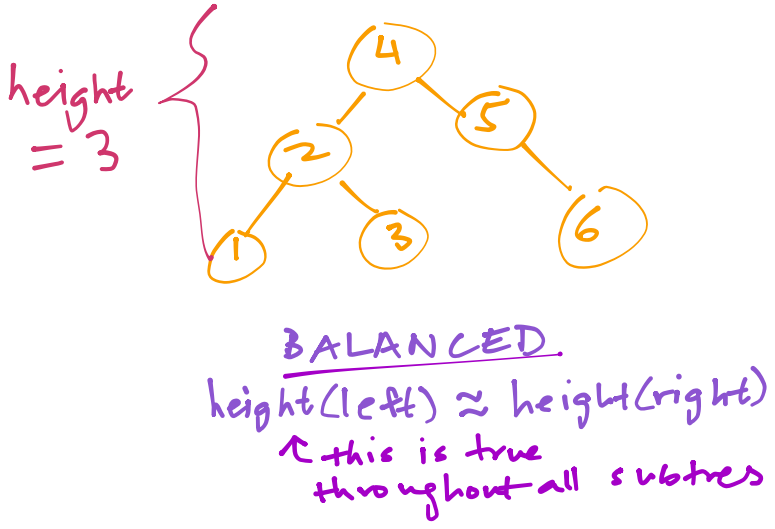
> Even though you should have circled a different number of tree values in each of the three diagrams, the number of values is always less than or equal to the *height* of each tree.
>
> Because `BinarySearchTree.__contains__` only ever recurses into one subtree, each recursive call goes down one level in the tree. So at most *one item per level* is ever compared against the `item` being searched for.
>
> So to better understand the running time of this algorithm, we need to investigate how the *height* of a BST can grow as we insert more items into it.

2. Recall that the *BST insertion algorithm* from last week's lab always inserts a new item into an "empty spot" at the bottom of a BST, so that the new item is a leaf of the tree.

   (a) Suppose we start with an empty BST, and then insert the items `1, 2, 3, 4, 5, 6` into the BST, in that order. Draw what the final BST looks like.

   (b) Suppose we start with an empty BST, and then insert the items `4, 2, 3, 5, 1, 6` into the BST, in that order. Draw what the final BST looks like.





3. Write down the *height* of each BST you drew in the previous question. (You can do so beside or underneath the diagram.)

4. Let n be a non-negative integer, and suppose we want a BST that contains the values `1, 2, ..., n`. What is the *maximum* possible height of the BST?

n worst possible height

5. Now suppose we want to find the *minimum* possible height of the BST. We're going to investigate this in an indirect way: for every height h, we're going to investigate what the maximum number of values can be stored in a binary tree of height n. Let's try to find a pattern.

   (a) Suppose we have a BST of height **0**. What's the maximum possible number of values in the BST? (Hint: there's only one kind of BST with height 0.)

   → empty BST

   Zero

   (b) Suppose we have a BST of height **1**. What's the maximum possible number of values in the BST? (Draw an example.)

   1 ① 1 node at most

   (c) Suppose we have a BST of height **2**. What's the maximum possible number of values in the BST? (Draw an example.)

   1×2 new nodes → $\leq$ 3 nodes at most

   (d) Repeat for heights 3 and 4.

   $1 \times 2 \times 2$ new nodes → 7 nodes

   15 nodes total

   $1 \times 2 \times 2 \times 2$

   (e) Suppose we have a BST of height **148**. What's the maximum possible number of values in the BST? (*Don't* draw an example, but find a pattern from your previous answers.)

   $2^{148} - 1$

   (f) Suppose we have a BST of height h and that contains n values. Write down an inequality of the form $n \leq \ldots$ to relate n and h. (This is a generalization of your work so far.)

   $n \leq 2^h - 1$

   total nodes ≤ max # of nodes

   (g) Finally, take your inequality from the previous part and isolate h. This produces the answer to our original question: what is the minimum height of a BST with n values?

   $n+1 \leq 2^h - 1 + 1$

   $n+1 \leq 2^h$

   $\log_2(n+1) \leq \log_2(2^h)$

   $\log(n+1) \leq h$

   $h \geq \log_2(n+1)$

   min height of a tree with n nodes

height	most # of nodes	next layer adds
0	0	1
1	1	$1 \times 2 = 2^1$
2	3	$1 \times 2 \times 2 = 2^2$
3	7	$1 \times 2 \times 2 \times 2 = 2^3$
4	15	$2^4 = 16$
5	31	$2^5 = 32$
⋮	⋮	
h	$2^h - 1$	

max # of nodes in a tree of height h