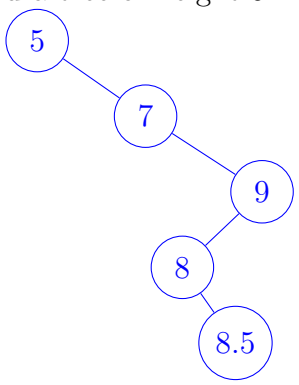
Opening Questions

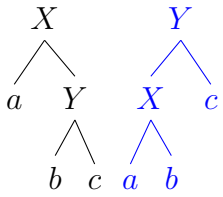
1. In your own words explain how you insert a new key in a binary search tree. When you have a value, you compare the value you're inserting to the value in the node. If the inserting value is less than the value, you recursively call the Insert function with the left node. If the value was greater, you recursively call Insert on the right node.

2. Give an example of a series of 5 keys inserted one at a time into a binary search tree that will yield a tree of height 5.



The runtime of Insert is $O(h)$ where h is the height of the tree.

3. If we do a left rotate on node X below, explain which left and/or right child links need to be changed.



This tree maintains the relationships of the first tree. c was already larger than Y, which is maintained. b was smaller than Y but larger than X, which has also been maintained. But now the height of C has been decreased by 1.

Tree depth vs Tree Height

The length of the path from the root r to a node x is the depth of x in T. The height of a node in a tree is the number of edges on the longest simple downward path from the node to a leaf, and the height of a tree is the height of its root. The height of a tree is also equal to the largest depth of any node in the tree.

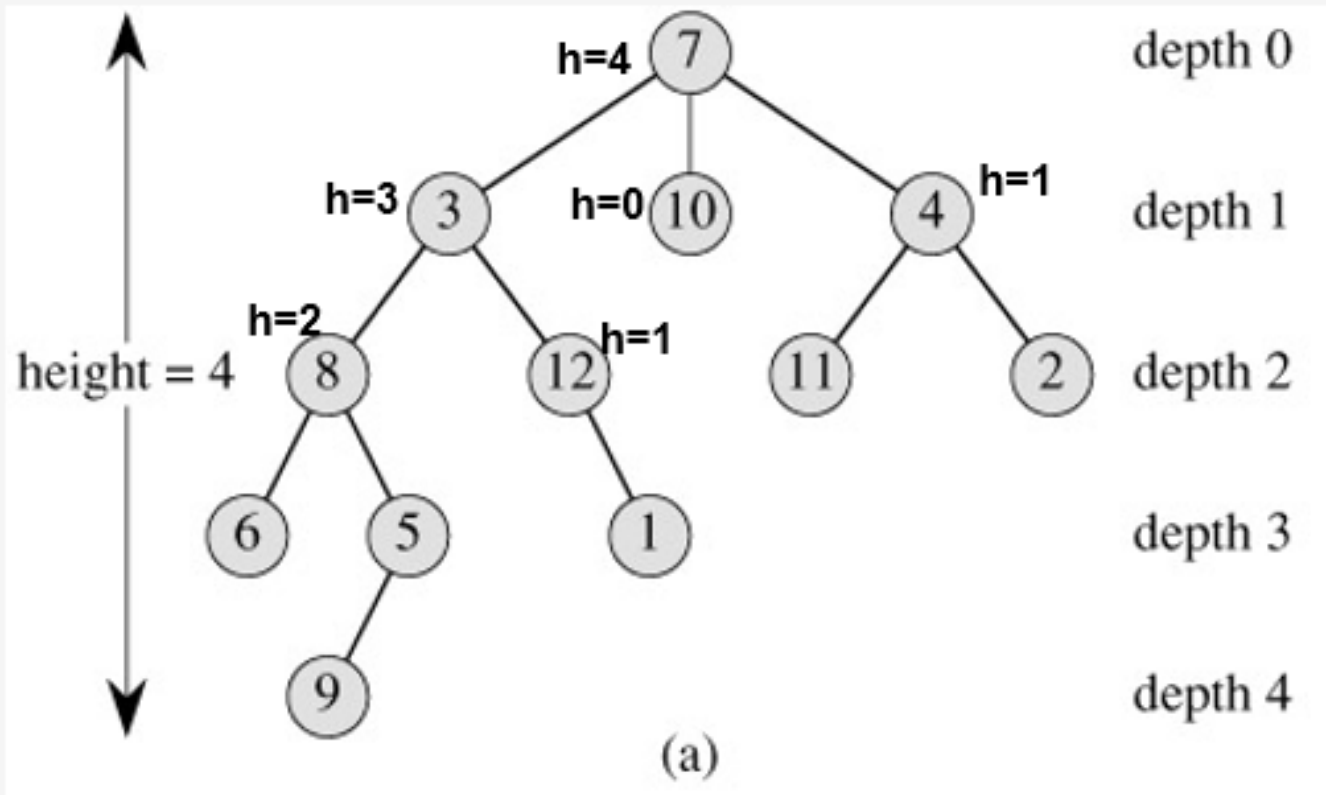
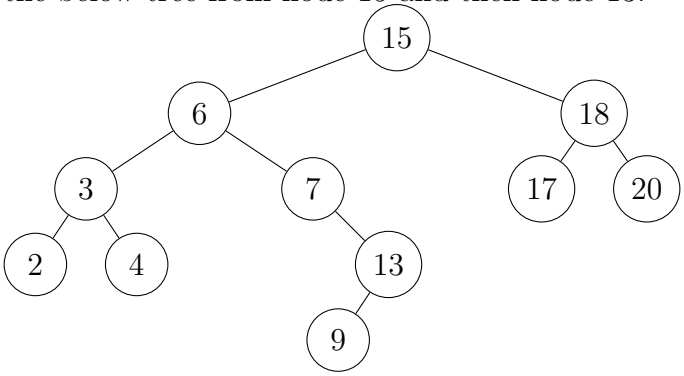


Figure 10.1: Height vs Depth of a Tree

BST Operations

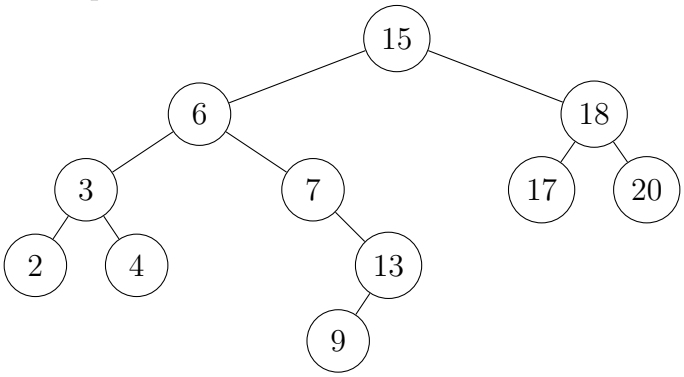
1) Write pseudocode for BST Successor (next larger value) (or Predecessor). Demonstrate on the below tree from node 15 and then node 13.



Algorithm 10.1 Binary Search Tree Successor (Best: $O(1)$, Worst: $O(h)$)

```
1: function SUCCESSOR(p: Node) ▷ The Successor is usually the min of the graph in the right node, essentially, right than left-left-...-left.
2:     if p.right exists then
3:         return MIN(p.right)
4:     else
5:         go up until you go up on left pointer
6:         if never go up left then
7:             return null                          ▷ You started in the largest value.
8:         end if
9:     end if
10: end function
```

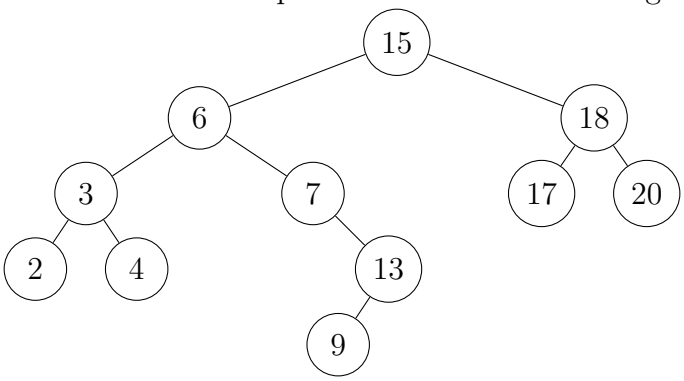
2) Write pseudocode for BST Insert. Demonstrate on the below tree to insert 5 and then 19.



Algorithm 10.2 Binary Search Tree Insert ($O(h)$)

```
1:  function INSERT(key)
2:      if root == null then
3:          root ← NODE(key)
4:          return
5:      end if
6:      p ← root
7:      q ← null                                  ▷ Following the parent node
8:      while p ≠ null do
9:          if p.val > key then
10:             q ← p
11:             p ← p.left                        ▷ Go Left
12:         else
13:             q ← p
14:             p ← p.right                       ▷ Go Right
15:         end if
16:     end while
17:     if q.val > key then
18:         q.left ← NODE(key)
19:     else
20:         q.right ← NODE(key)
21:     end if
22: end function
```

3) What are the three possible cases when deleting a node from a BST?



- No children
  - Set the $parent.left$ or $parent.right$ (depending on where you came from) to null; $O(1)$.
- One child
  - Make the parent of the child point to the child of the node you're deleting (essentially splicing out the node you're deleting); $O(1)$.
- Two children

– Swap the predecessor or successor value into the node, and then deleting the value where the predecessor—successor was.

Algorithm 10.3 Delete A Node with 2 Children from a BST

```
1: function DELETE2CHILD(p: Node)
2:     successor ← SUCCESSOR(p)                ▷ Or Predeccesor, O(h)
3:     p.value ← successor.value               ▷ O(1)
4:     DELETE(successor)                       ▷ O(h)
5: end function
```

4) Write pseudocode for BST Delete (assume you already have a pointer to the node)

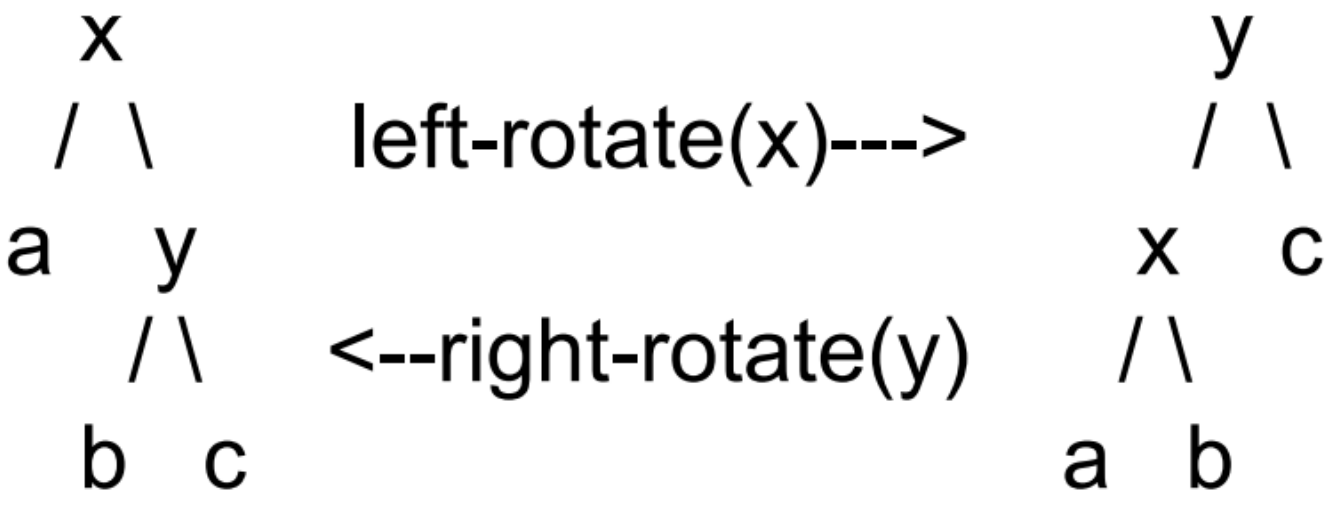
Algorithm 10.4 Binary Search Tree Delete

```
1: function DELETE(p: Node)                    ▷
2:
3: end function
```

BST Rotations

Local operation in a search tree that maintains the BST property and possibly alters the height of the BST.

x and y are nodes; a, b, c are sub trees



5. Write pseudocode for LeftRotate (or RightRotate). What is the worst case runtime?

Algorithm 10.5 Binary Search Tree Left Rotate

```
1: function LeftRotate(p:Node, z:Node)    ▷ To make this right rotate, switch all lefts with rights and rights with left.
2:     savey ← p.right                    ▷ z is the parent of p
3:     if z.left = p then
4:         z.left ← savey
5:     else
6:         z.right ← savey
7:     end if
8:     p.right ← z.left
9:     y.left ← p
10:    savey.left ← p
11: end function
```