Outline

Approaching BST

Making a decision

We discussed the trade-offs between linked and array-based implementations of sequences (back in Section 4.7).

Linked lists are efficient for insertion and deletion operations, while a sorted array allows for efficient searching (recall binary search algorithm), although requires $\Theta(n)$ for insertion and deletion operations.

Let's combine the best of both worlds!

Approaching BST

Making a decision

We discussed the trade-offs between linked and array-based implementations of sequences (back in Section 4.7).

Linked lists are efficient for insertion and deletion operations, while a sorted array allows for efficient searching (recall binary search algorithm), although requires $\Theta(n)$ for insertion and deletion operations.

Let's combine the best of both worlds!

Approaching BST

Making a decision

We discussed the trade-offs between linked and array-based implementations of sequences (back in Section 4.7).

Linked lists are efficient for insertion and deletion operations, while a sorted array allows for efficient searching (recall binary search algorithm), although requires $\Theta(n)$ for insertion and deletion operations.

Let's combine the best of both worlds!

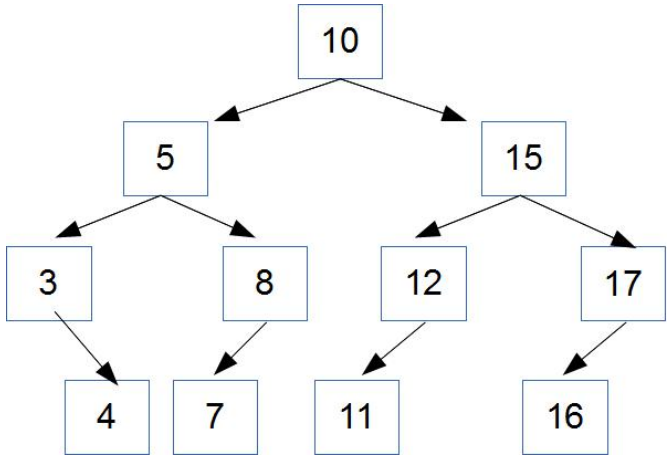
The Binary Search Property

A Binary Tree is "Sorted"

A Binary Search Tree, or BST, is a binary tree where every node has the following property:

- Each value in the left subtree is less than the value at the node.
- Each value in the right subtree is greater than the value at the node.

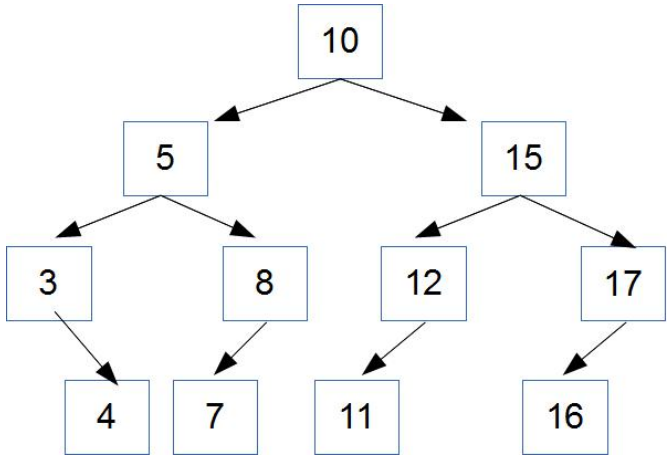
No duplicates!



The Binary Search Property

Binary Search With A Binary Tree

- Start at the root
- If the value is there, we are done
- If the value is less than the node value, search the left subtree
- If the value is greater than the node value, search the right subtree



The Binary Search Property

Performance (Running Time) To Find A Value

- Average Performance is $\Theta(\log n)$.
  If the tree is not too unbalanced, then we divide the number of items to search in half at each node. This is actually a binary search.

- Worst-Case Performance is $\Theta(n)$.
  If the tree branches only to one side (left or right) this is the same as linear search.

TreeNode Class

Recursive Definition Of A TreeNode Class in Python

```
class TreeNode(object):
    def __init__(self, data = None, left=None, right=None):
        self.item = data
        self.left = left # TreeNode or None
        self.right = right # TreeNode or None
```

TreeNode Class

Recursive Definition of a TreeNode Class in C++

```
//TreeNode.h
class TreeNode{

public:
   TreeNode(int item, TreeNode* left = NULL, TreeNode*
right = NULL);

private:
   TreeNode(const TreeNode &tnode); // no Copy Constructor
   void operator=(const TreeNode &tnode); // no assignment
operator
   int _item;
   TreeNode* _left;
   TreeNode* _right;
};
```

See the files TreeNode.h, TreeNode.cpp, and usingTreeNode.cpp.

Implementing A BST

in Python: __init__ (Constructor)

```
from TreeNode import TreeNode

class BST (object):

   def __init__(self):

      """ creates empty binary search tree """

      self.root = None
```

Implementing A BST

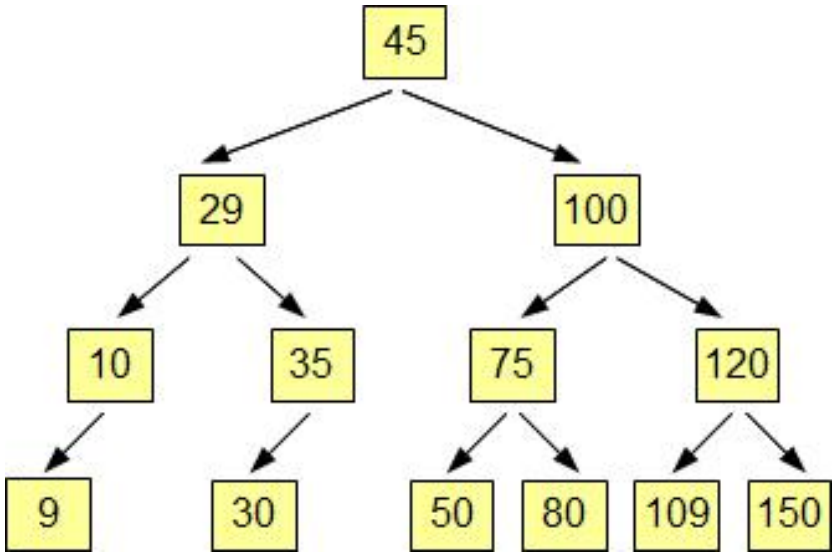
in C++ : BST.h

```
// BST.h
class BST{

public:
   BST() { _root = NULL; } // creates empty tree
    BST(); // destructor

private:
   TreeNode *_root;
};
```

Insertion into BST

Inserting into BST

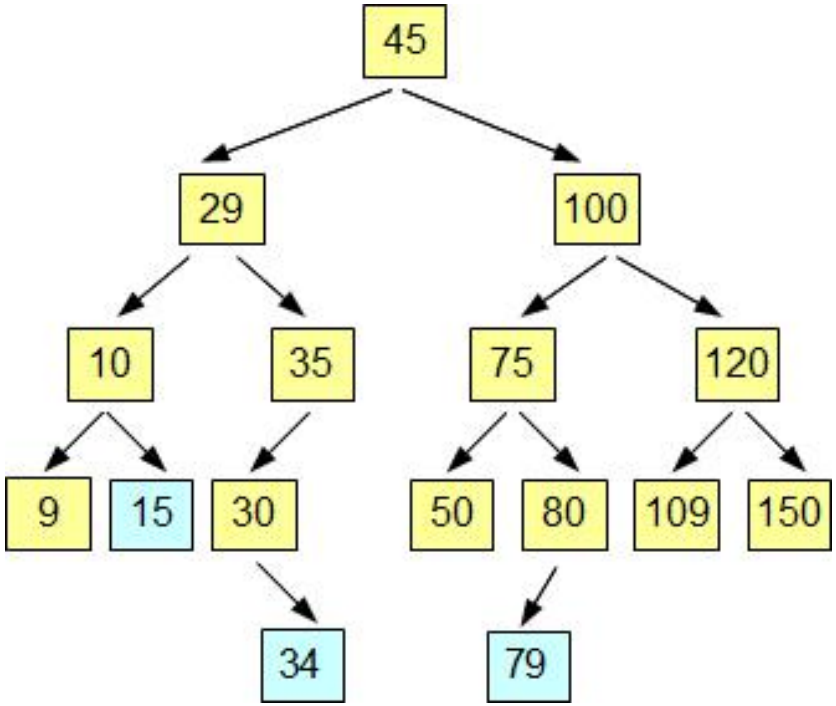
For the following BST, let's insert values 15, 34, and 79.



Insertion into BST

Inserting into BST

After insertion of 15, 34, and 79:



Insertion into BST

Insertion: iterative or recursive

Trees are a naturally recursive data structure.

Therefore insertion can be implemented recursively, but let's start with iterative version first.

See the files BST.h and BST.cpp.

Insertion into BST

Search in BST tree: Python

```
def find(self, item):

   """ post:  returns item from BST; None otherwise """

   node = self.root
   while node is not None and not(node.item == item):
      if item < node.item:
         node = node.left
      else:
         node = node.right

   if node is None:
      return None
   else:
      return node.item
```

Insertion into BST

Search in BST tree: C++

Deletion from BST

Removing nodes from the tree

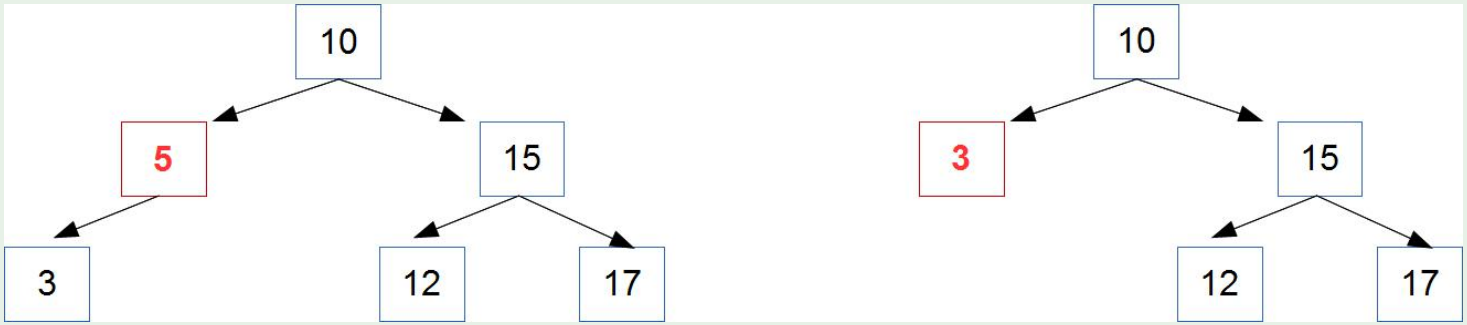
Removing a specific item from a BST is a bit tricky. List of cases:

- the node to be removed is a leaf:
  then we can simply drop it off the tree
  (reference in its parent node is set to None)
- the node to be removed has a single child:
  then we can simply reset the reference from its parent to the reference to the node's child instead.
- the node to be removed has two children:
  leave the node in place, but replace its contents (i.e. value), i.e. find an easily deletable node whose contents can be transferred into the target node, while maintaining the tree's binary search property.
  (Two options there: rightmost node of the left subtree (our book), or leftmost node of the right subtree)

Deletion from BST

Example 1

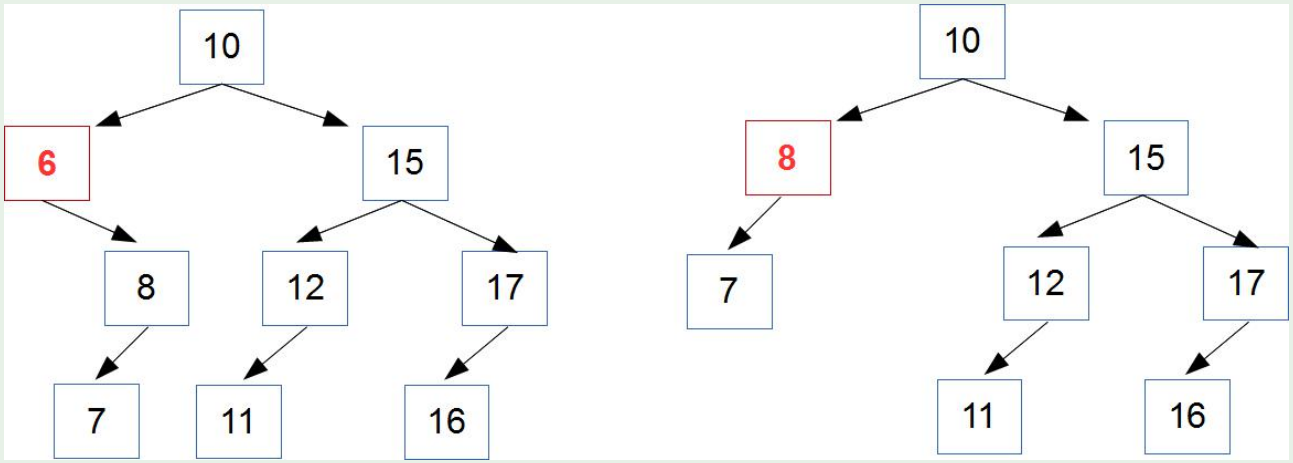
Let's delete 5 from a BST:



Deletion from BST

Example 2

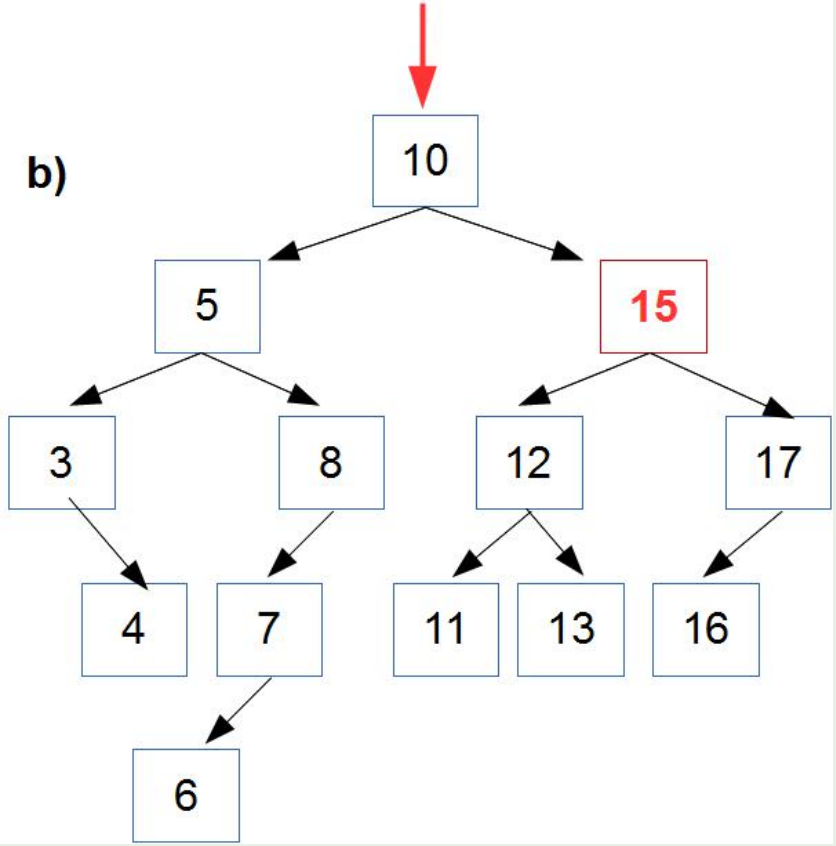
Let's delete 6 from a BST:



Deletion from BST

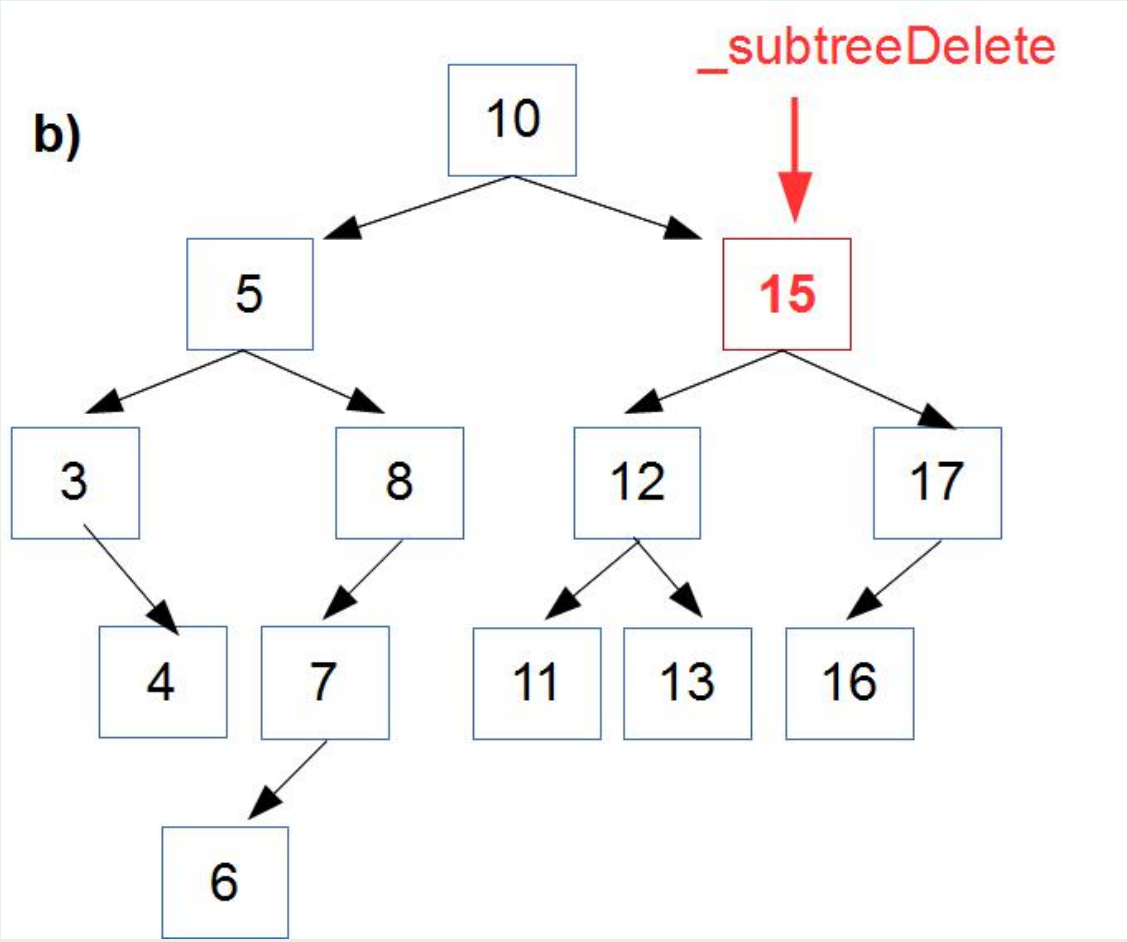
Example 3

Let's delete 15 from the given BST:



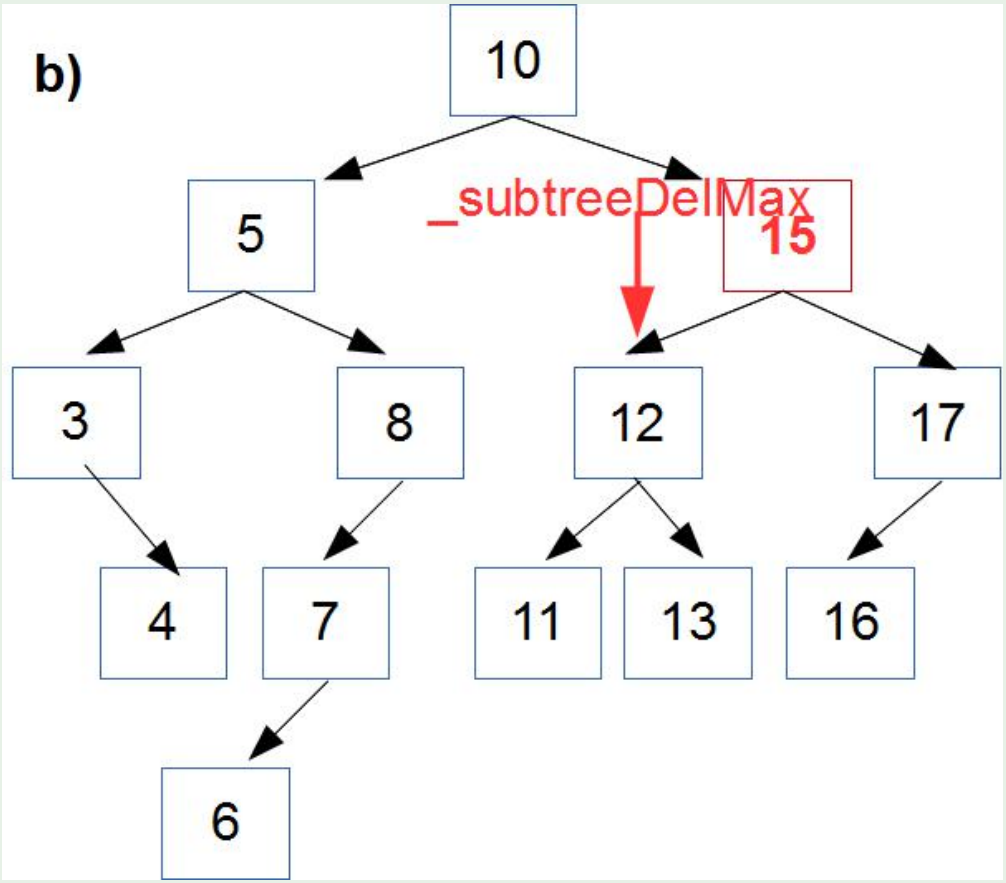
Deletion from BST

Example 3



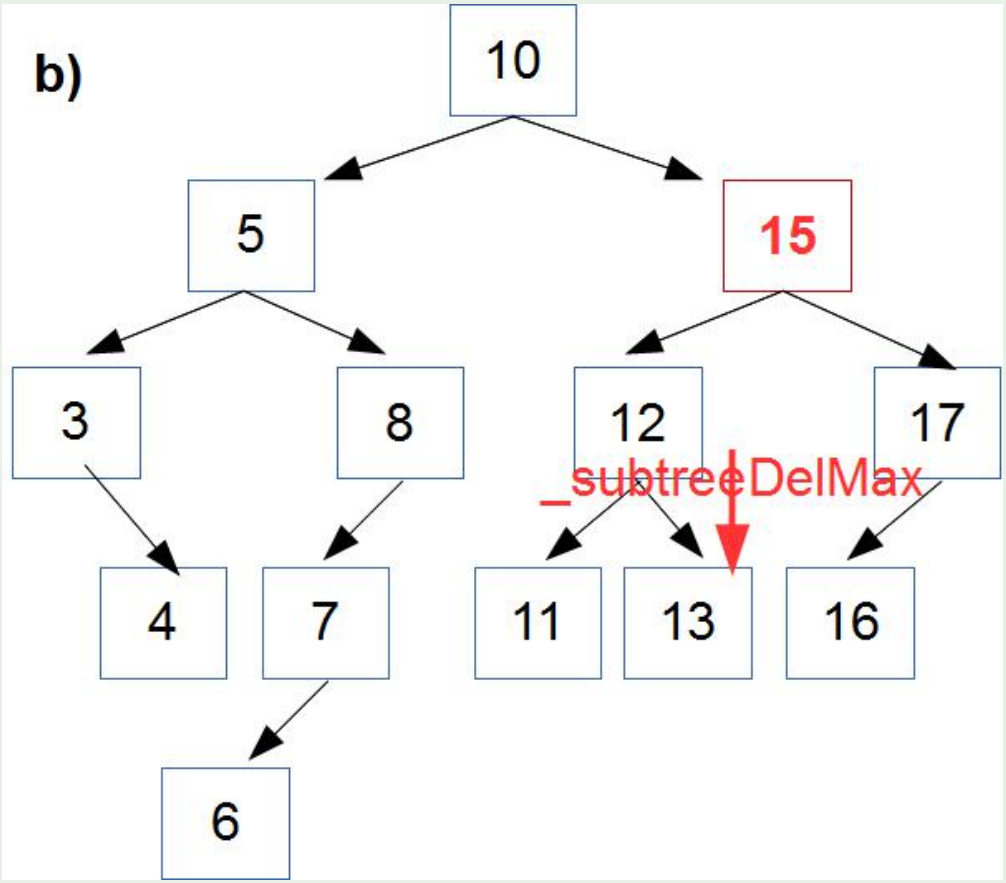
Deletion from BST

Example 3



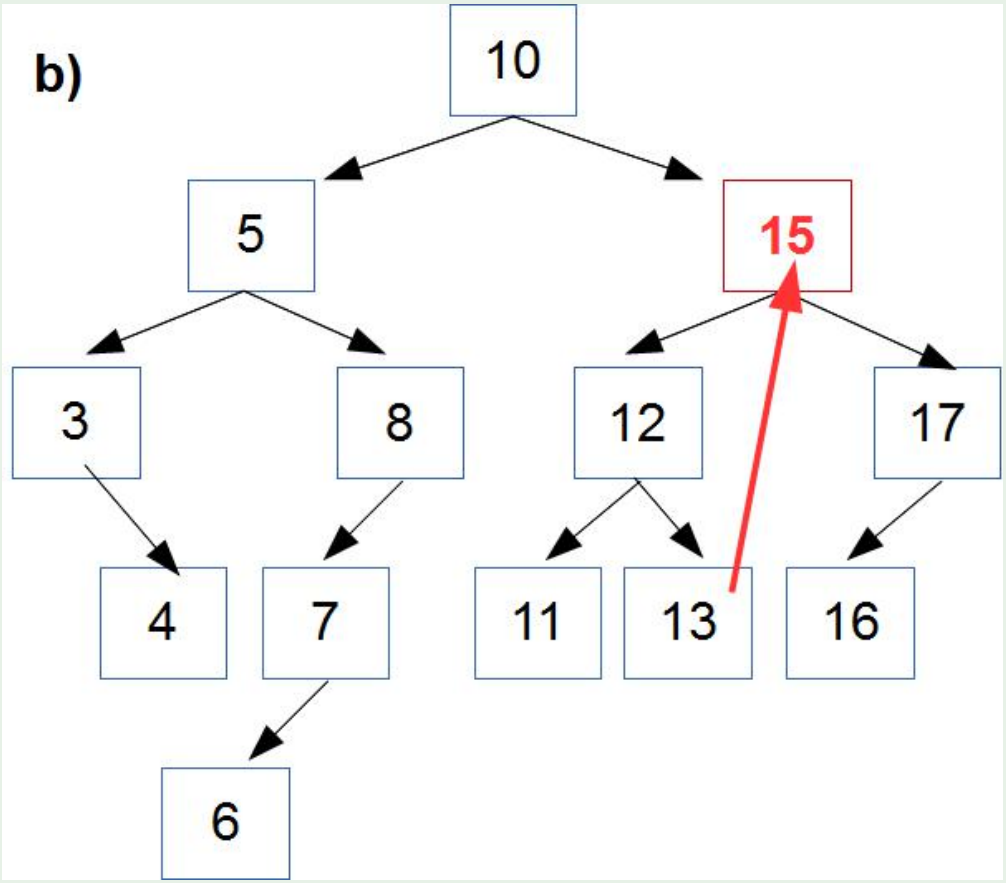
Deletion from BST

Example 3



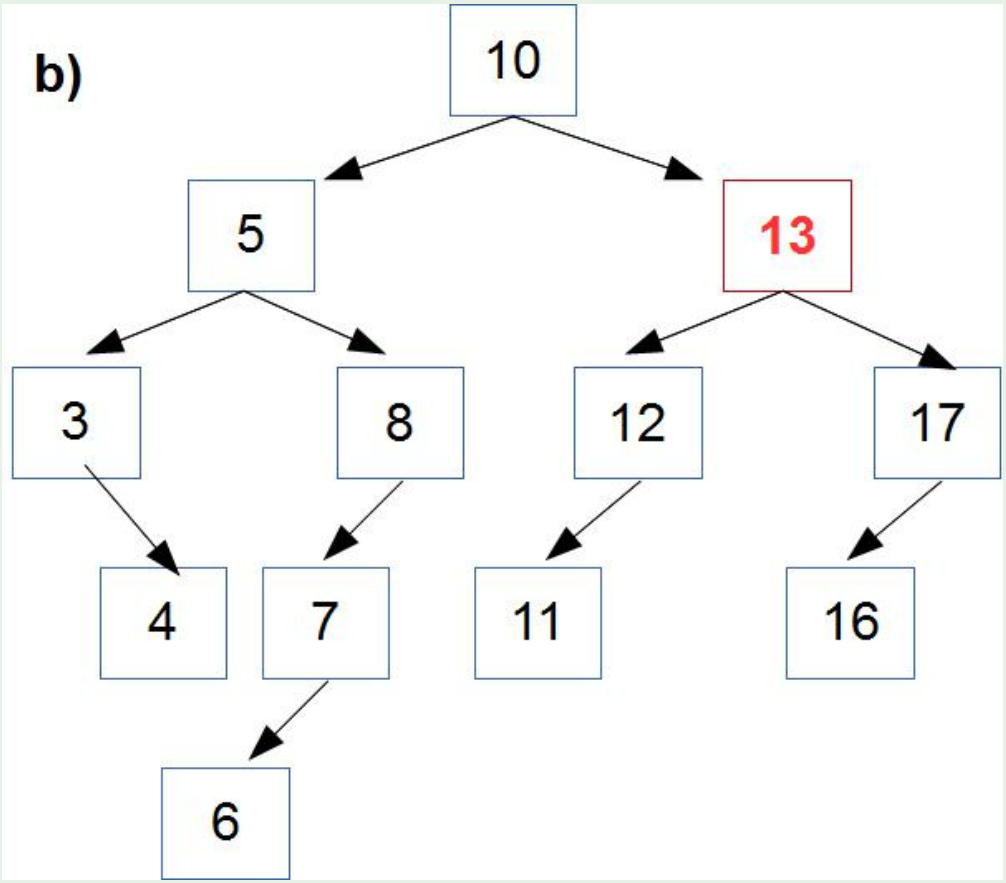
Deletion from BST

Example 3



Deletion from BST

Example 3



Deletion from BST

Conclusion

In conclusion, we can say that when deleting a value from BTS, which is represented by a node with two children, our book follows the following procedure:

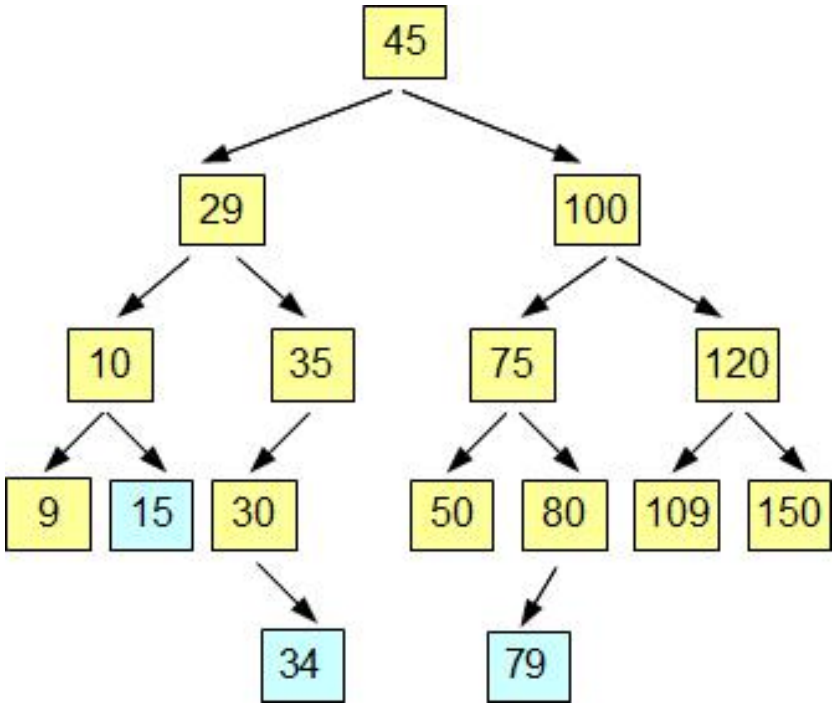
- step to the left sub-tree,
- locate the rightmost node(or a leaf) in it,
- and place its value in to the node with value to be deleted, making necessary adjustments of the references.

Deletion from BST

Removing nodes from BST

Let's practice more!

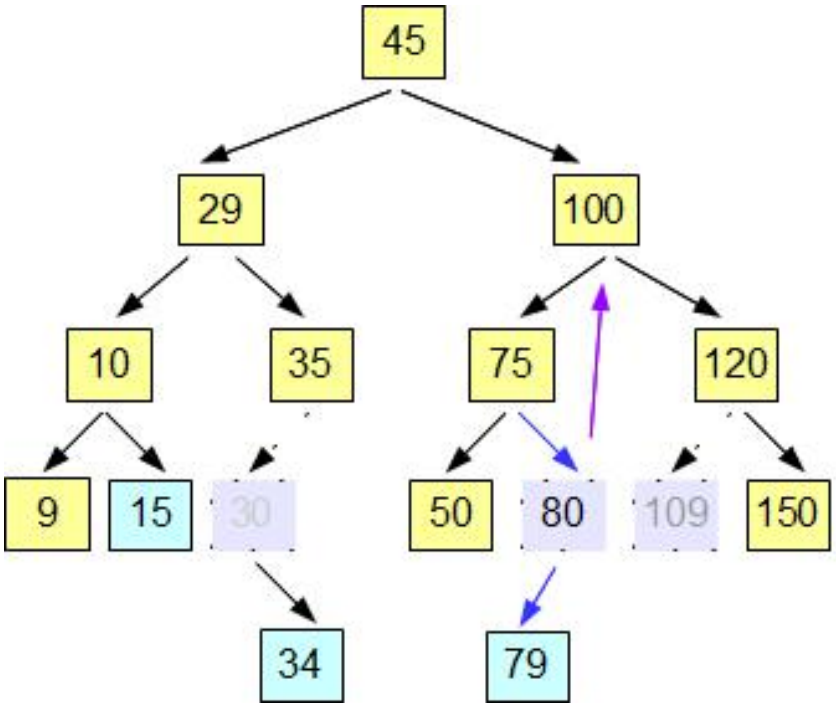
Delete 30, 100, and 109 from the following tree:

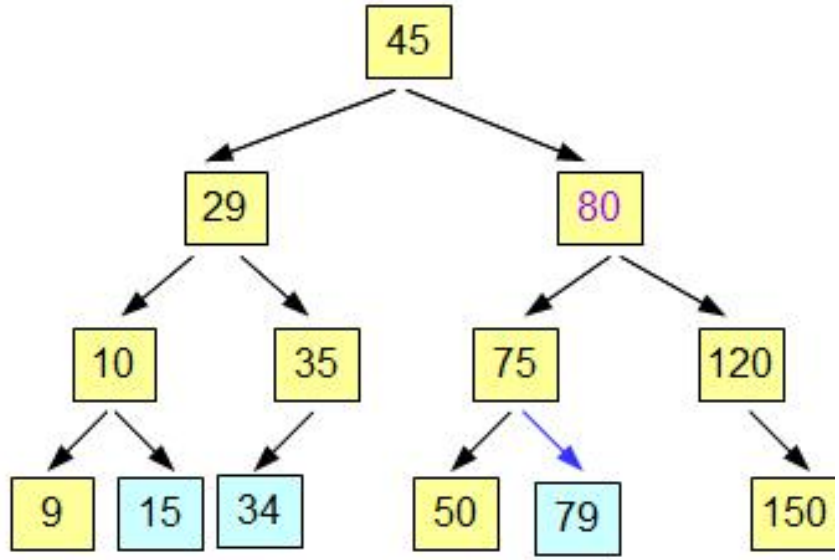


Deletion from BST

Removing nodes from BST

After deletion of 30, 100, and 109:





Printing BST

displaying BST as list

We would like to see what is happenning to our tree after deletion and insertion operations.

Let's generate an array representation for our tree.

Convention: empty nodes will be represented by zeros (for now).

Run-Time Analysis Of BST Methods

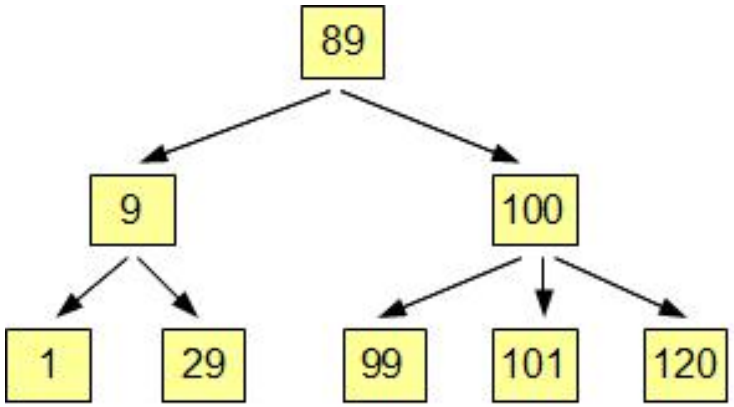
Methods

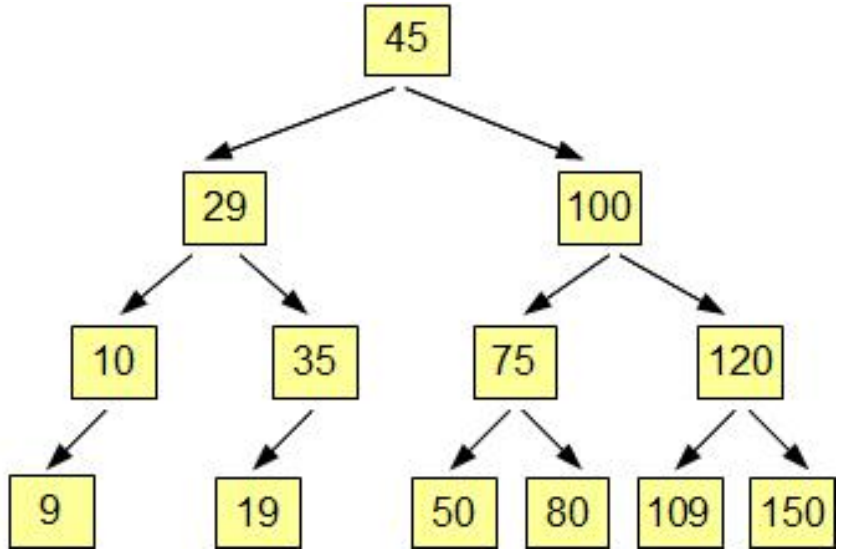
- asList is $\Theta(n)$.
- insert, delete, find have $\Theta(\log n)$ average behavior.
- insert, delete, find have $\Theta(n)$ worst-case behavior.

In-class work

In-class work - Part 1

For the following, state whether each of them is a binary tree, a binary search tree (BST), or just a tree.





In-class work

In-class work - on your own, part 2

Using class BST, insert the following numbers, one by one: 14, 10, 18, 25, 17, 7, 1, 12, 30.
Draw this BST as you think it should look. Then print the BST to see it in array representation.