

**CSIS 3103**

Ch 6:  
Binary Search Trees

## Binary Search Tree

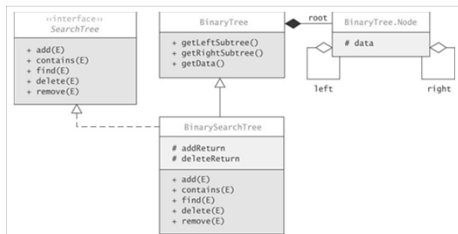
- Never has to be sorted because its elements always satisfy the required order relations
- When new elements are inserted (or removed) properly, the binary search tree maintains its order

## SearchTree Interface

Method	Behavior
boolean add(E item)	Inserts <i>item</i> where it belongs in the tree. Returns <b>true</b> if item is inserted; <b>false</b> if it isn't (already in tree).
boolean contains(E target)	Returns <b>true</b> if <i>target</i> is found in the tree.
E find(E target)	Returns a reference to the data in the node that is equal to <i>target</i> . If no such node is found, returns <b>null</b> .
E delete(E target)	Removes <i>target</i> (if found) from tree and returns it; otherwise, returns <b>null</b> .
boolean remove(E target)	Removes <i>target</i> (if found) from tree and returns <b>true</b> ; otherwise, returns <b>false</b> .

## BinarySearchTree Class

Data Field	Attribute
protected boolean addReturn	Stores a second return value from the recursive add method that indicates whether the item has been inserted.
protected E deleteReturn	Stores a second return value from the recursive delete method that references the item that was stored in the tree.



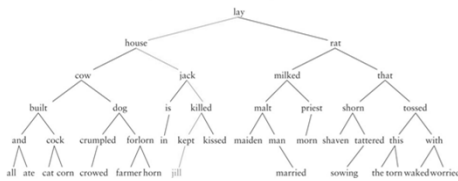
## Searching a Binary Search Tree

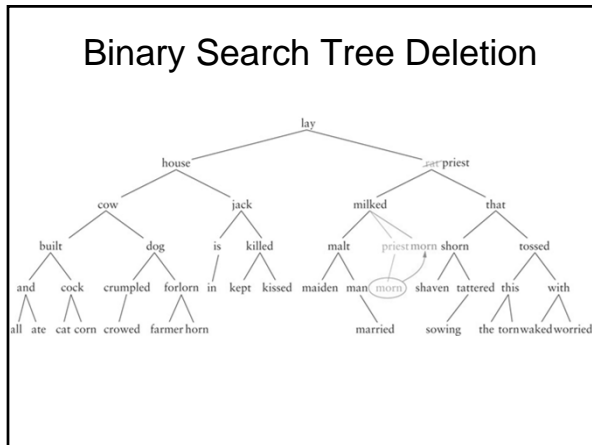
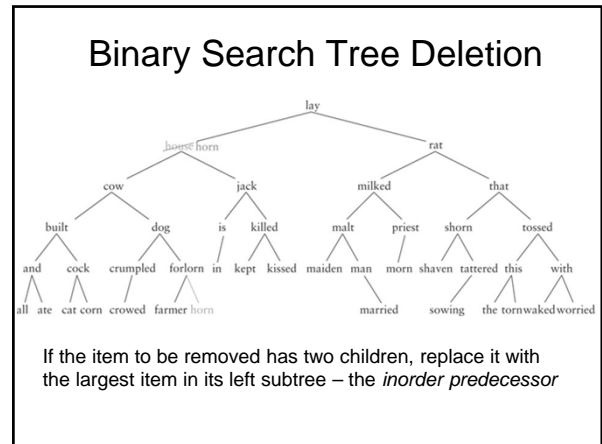
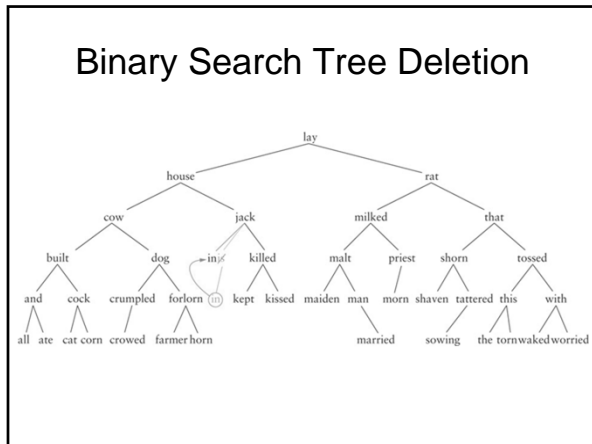
1. **if** the root is **null**
2.     return **null** (item is not in the tree)
3. Compare the value of **target** with **root.data**
4. **if** they are equal
5.     return the data at the root (target was found)
6. **else if** the target is less than **root.data**
7.     return the result of searching the left subtree
8. **else**
9.     return the result of searching the right subtree

## Binary Search Tree Insertion

Recursive Algorithm for Insertion in a Binary Search Tree

1. **if** the root is **null**
2.     Replace empty tree with a new tree with the item at the root and return **true**.
3. **else if** the item is equal to **root.data**
4.     The item is already in the tree, return **false**.
5. **else if** the item is less than **root.data**
6.     Recursively insert the item in the left subtree.
7. **else**
8.     Recursively insert the item in the right subtree.





### Removing from a Binary Search

Recursive Algorithm for Removal from a Binary Search Tree

1. if the root is null
2. The item is not in tree – return null.
3. Compare the item to the data at the local root.
4. if the item is less than the data at the local root
5. Return the result of deleting from the left subtree.
6. else if the item is greater than the local root
7. Return the result of deleting from the right subtree.
8. else // The item is in the local root
9. Store the data in the local root in deleted@return.
10. if the local root has no children
11. Set the parent of the local root to reference null.
12. else if the local root has one child
13. Set the parent of the local root to reference that child.
14. else // Find the inorder predecessor
15. if the left child has no right child it is the inorder predecessor
16. Set the parent of the local root to reference the left child.
17. else
18. Find the rightmost node in the right subtree of the left child.
19. Copy its data into the local root's data and remove it by setting its parent to reference its left child.

### Testing a Binary Search Tree

To test a binary search tree, verify that an inorder traversal will display the tree contents in ascending order after a series of insertions and deletions are performed