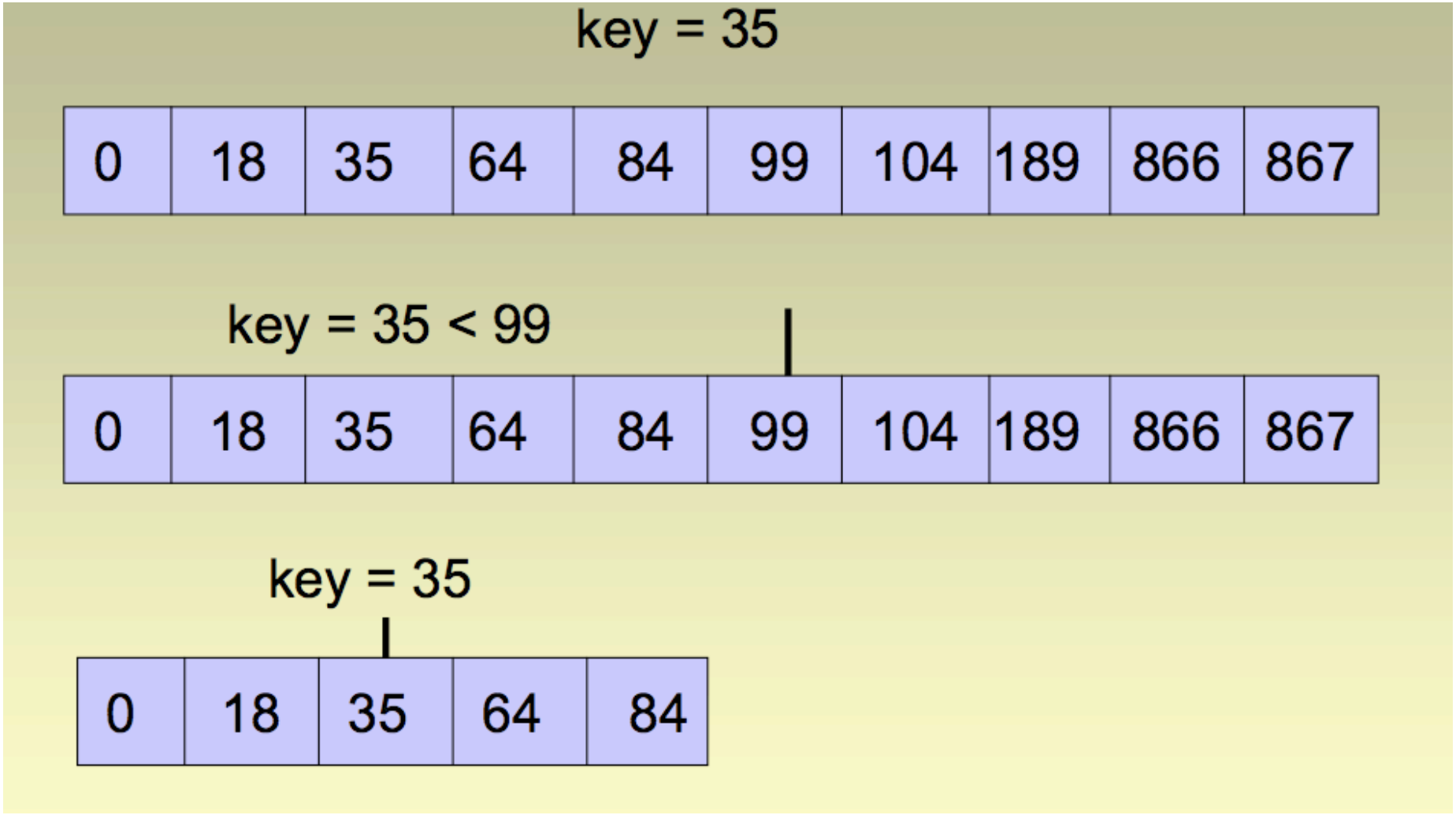
ECE 2574
Introduction to Data Structures and Algorithms

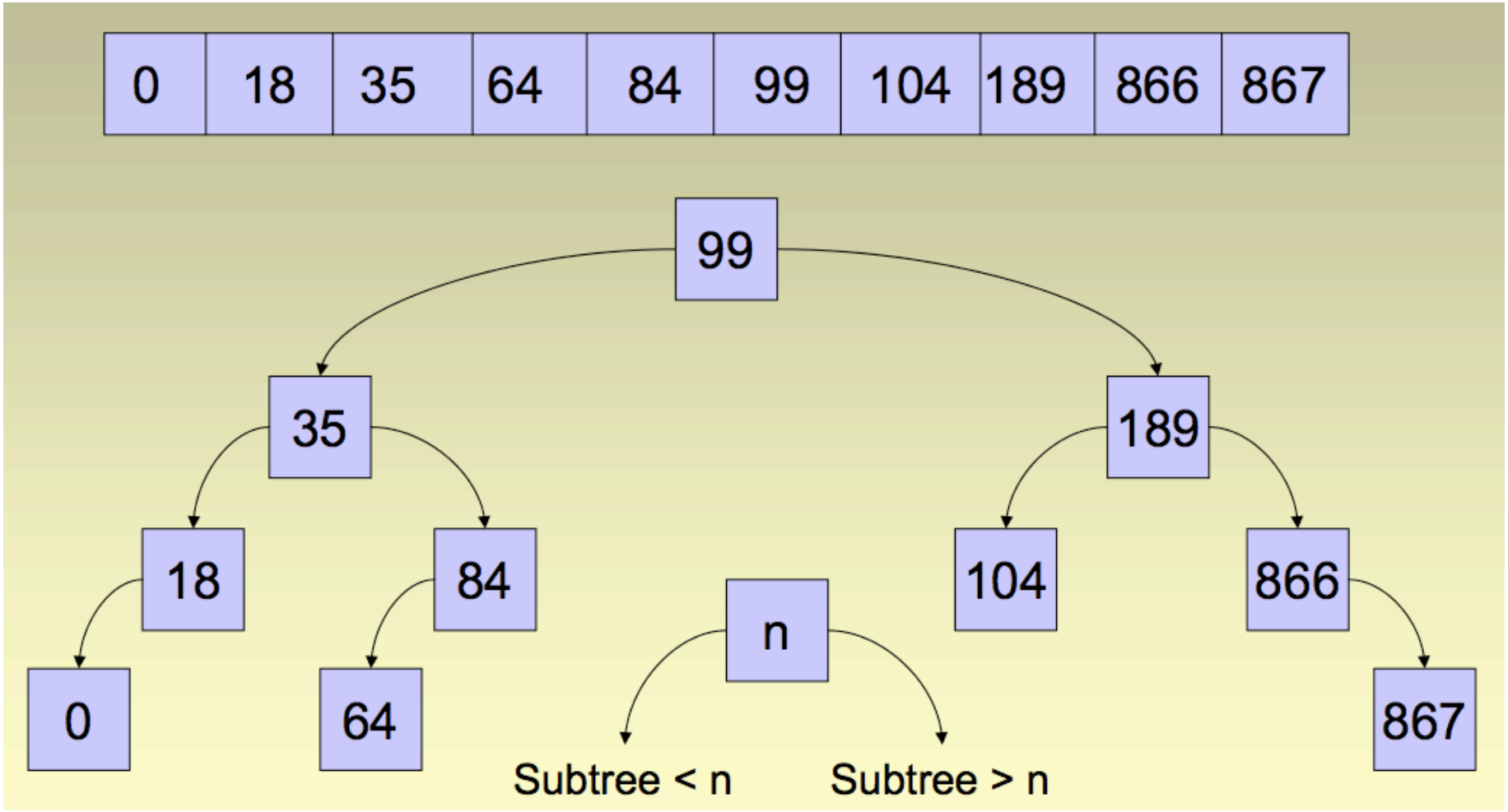
31: Binary Search Trees

Chris Wyatt
Electrical and Computer Engineering

Recall the binary search algorithm using a sorted list.

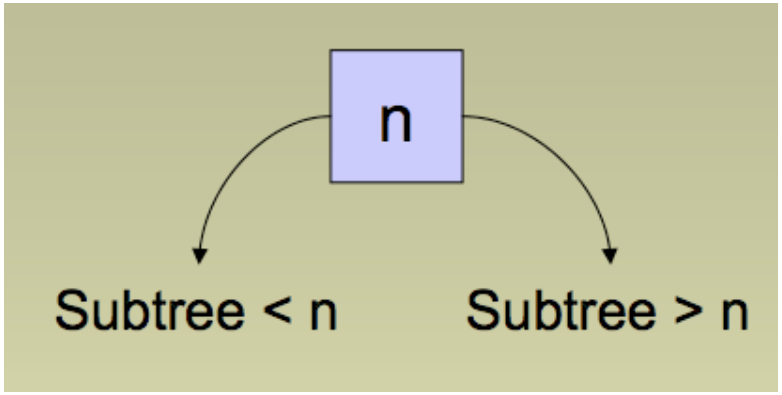


We can represent the sorted list using a binary tree with a specific relationship among the nodes.



This leads to the Binary Search Tree ADT.

We can map the sorted list operations onto the binary search tree.

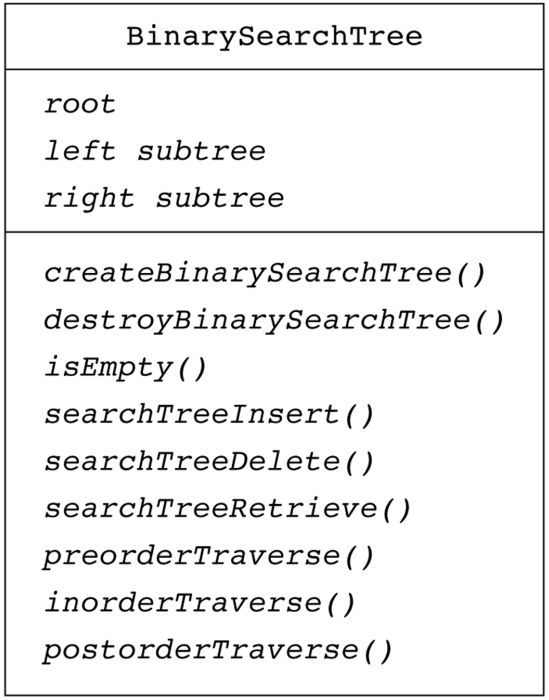


Because the insert and delete can use the binary structure, they are more efficient.

Better than binary search on a pointer-based (linked) list.

Binary Search Tree (BST) Operations

Consider the items of type TreeItemType to have an associated key of keyType.



// create an empty BST

+createBST()

// destroy a BST

+destroyBST()

// check if a BST is empty

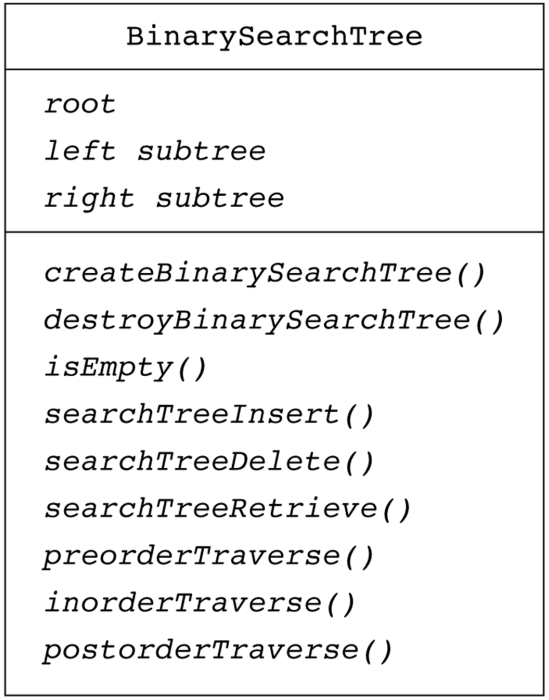
+isEmpty(): bool

Binary Search Tree (BST) Operations

```
// insert newItem into the BST based on its
// key value, fails if key exists or node
// cannot be created
+insert(in newItem:TreeItemType): bool

// delete item with searchKey from the BST
// fails if no such key exists
+delete(in searchKey:KeyItemType): bool

// get item corresponding to searchKey from
// the BST, fails if no such key exists
+retrieve(in searchKey:KeyItemType,
                out  treeItem:TreeItemType): bool
```

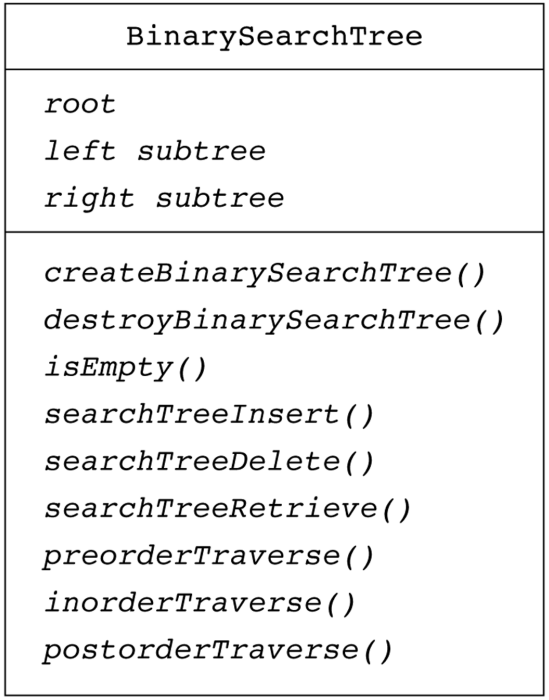


Binary Search Tree (BST) Operations

// call function visit passing each node data
//as the argument, using a preorder
//traversal
+preorderTraverse(in visit:FunctionType)

// call function visit passing each node data
// as the argument, using an inorder
//traversal
+inorderTraverse(in visit:FunctionType)

// call function visit passing each node dat a
// as the argument, using a postorder
// traversal
+postorderTraverse(in visit:FunctionType)



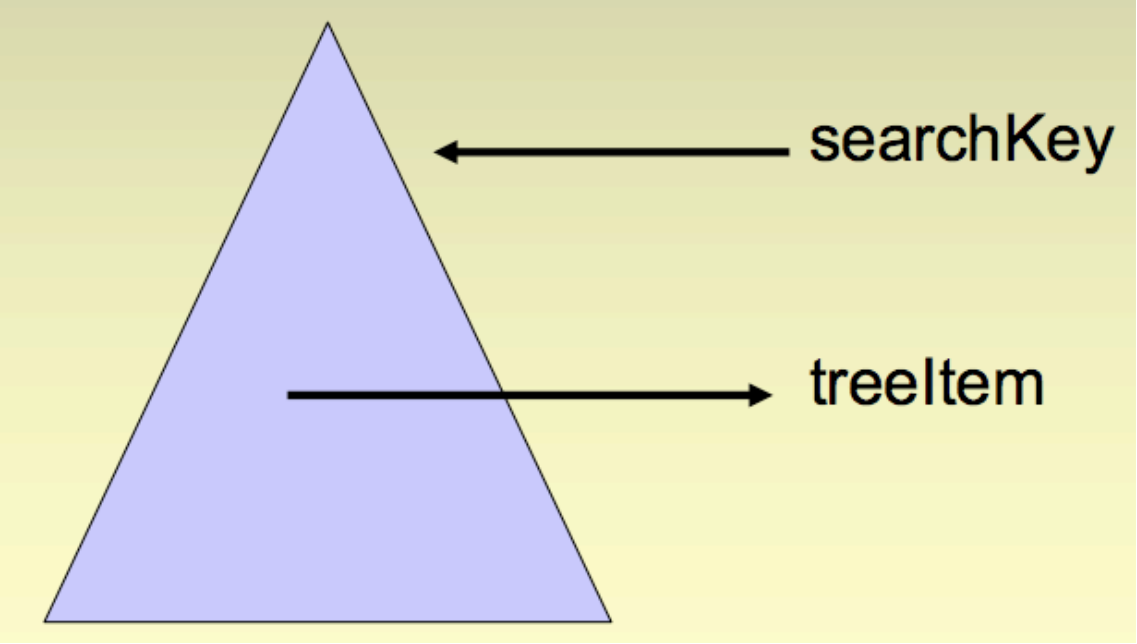
An interface for Binary Search Trees

Template over the key type and the value type.

See `abstract_bst.h`

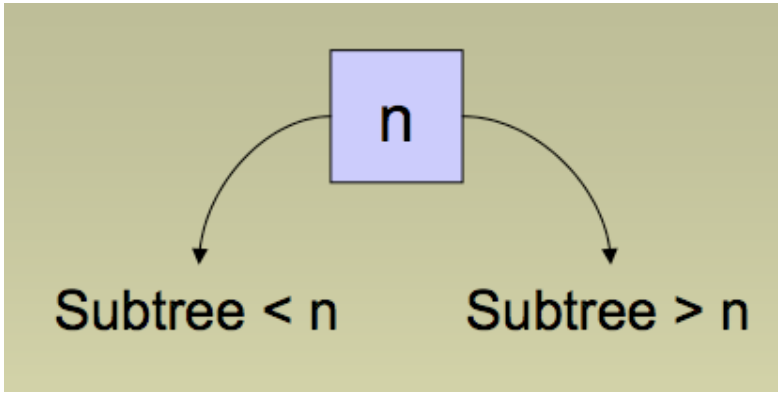
Binary Search Tree implementation.

```
// get item corresponding to searchKey from the BST
// fails if no such key exists
+retrieve(in searchKey:KeyItemType,
          out treeItem:TreeItemType): bool
```



Pseudo-code for search

```
// searches the BST tree for item corresponding to key
search(intree:BinarySearchTree, in key:KeyItemType)
if( tree.isEmpty() )
    no item found
if(key= key of the root)
    item found
else if (key < key of the root)
    search(leftsubtreeof tree, key)
else
    search(rightsubtreeof tree, key)
```



In class exercise

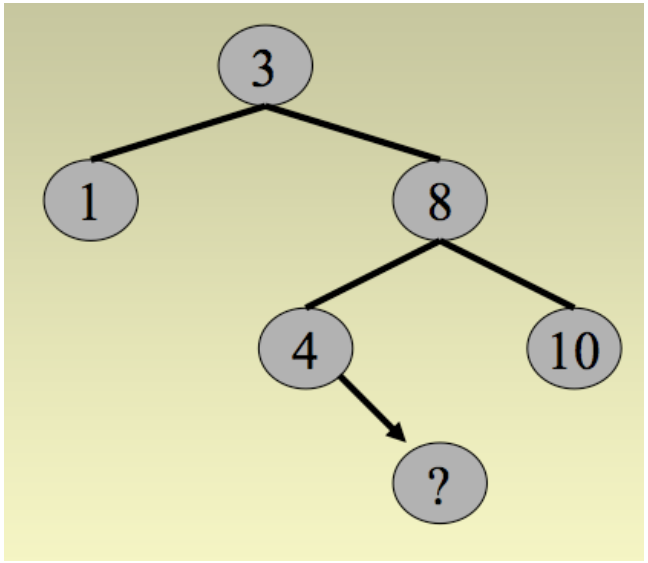
What is the complexity of search?

How to insert into the BST so as to maintain the ordering.

What if we try to search for key = 7

Search terminates at the right subtree of node 4.

What does that mean ?



If we insert 7 at ?, it is where it “belongs”

Pseudo-code for insert

```
insert(in key:KeyItemType, in item:TreeItemType)
if( search for key fails)
    if(key< last node searched)
        insert at left subtree of last node searched
    else
        insert at right subtree of last node searched
    endif
else
    insert fails
endif
```

In class exercise

What is the complexity of insert?

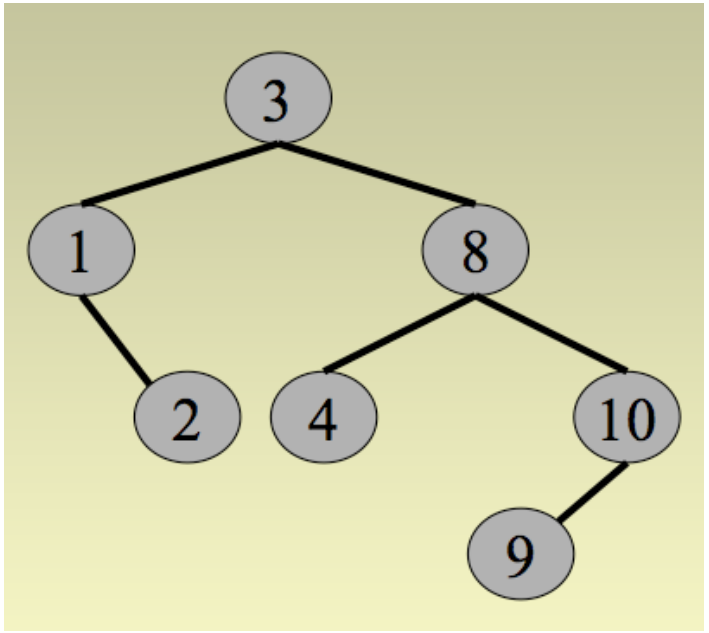
How to delete from the BST so as to maintain the ordering.

What if we try to delete key = 4?

What if we try to delete key = 1?

What if we try to delete key = 10?

What if we try to delete key = 8?

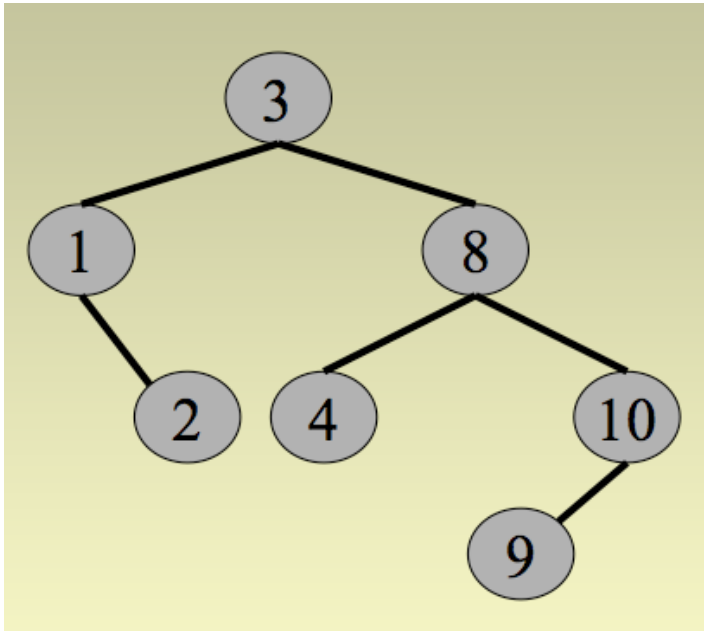


Case where the node to delete has 2 children.

Attempt to delete node 8.

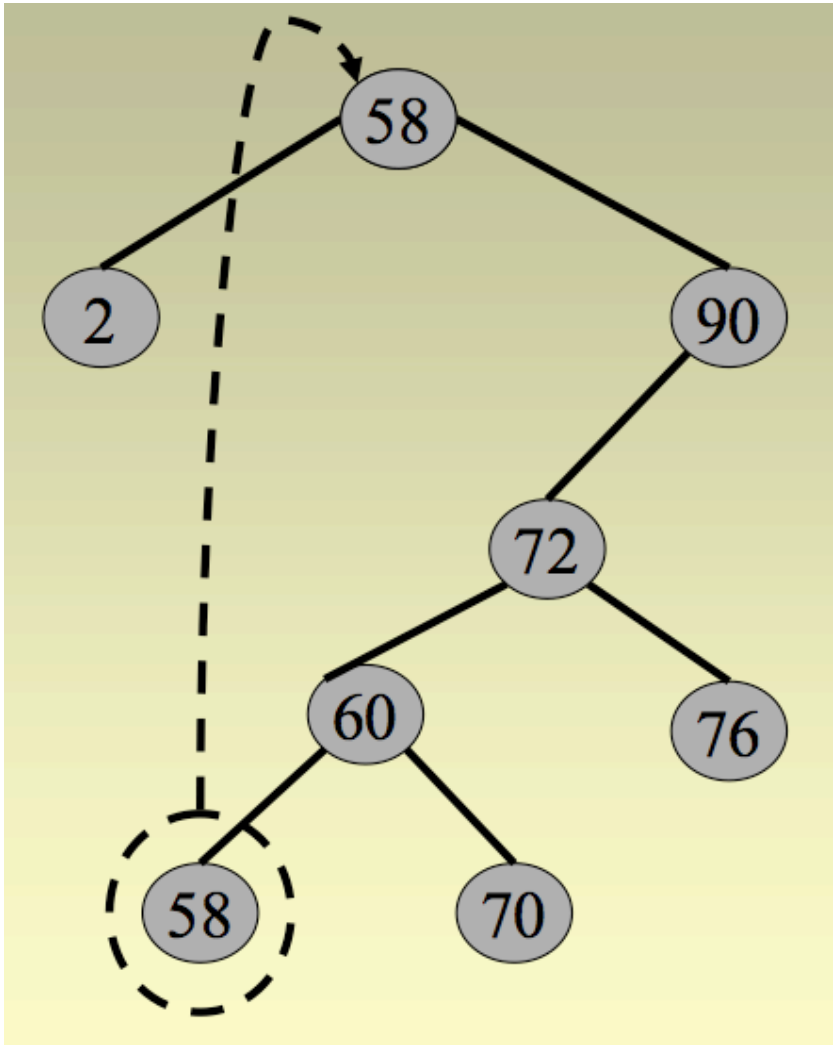
What if we find a node easier to delete and delete it instead.

If we choose the inorder successor, we can copy its contents (key and item) into current node (8), then delete it instead.



The *inorder successor* of a node rooted at R, is the leftmost node of the right subtree of R.

Which is the inorder successor of this subtree?



Pseudo-code for delete

```
delete(in key:KeyItemType)
if( search for key fails)
    delete fails
else
    if (found node is leaf)
            delete it
    if (found node has left/right child only)
            delete node, replace with left/right child,
    else
            find inorder successor, copy to found node
            delete inorder successor
    endif
endif
```

In class exercise

What is the complexity of delete?

Next Actions and Reminders

Read CH pp. 455-458 and Chapter 17 on heaps.

Program 4 is due 11/17.