# CSC115 Lecture 16

LillAnne Jackson

## Trees

1. *Binary Tree implementation*
2. *Tree Traversal*

## The ADT Binary Tree: Basic Operations of the ADT Binary Tree

The operations available for a particular ADT binary tree depend on the type of binary tree being implemented

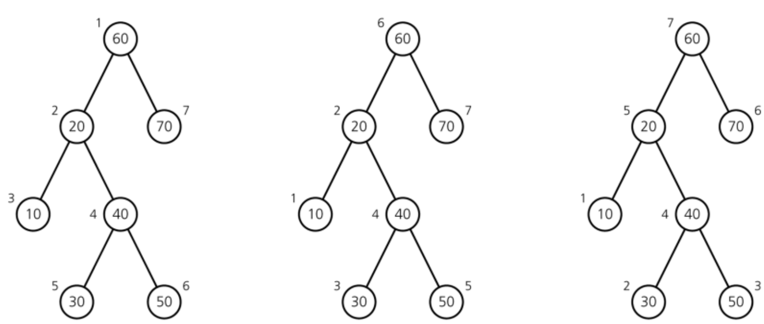
```
public void setRootItem(newItem);

public void attachLeft(newItem)
     throws TreeException;
public void attachRight(newItem)
    throws TreeException;

public void attachLeftSubtree(leftTree)
    throws TreeException;
public void attachRightSubtree(rightTree)
    throws TreeException;

public void detachLeftSubtree(fromHere)
    throws TreeException;
public void detachRightSubtree(fromHere)
    throws TreeException;
```

# Traversals of a Binary Tree

- A traversal algorithm for a binary tree visits each node in the tree
- Recursive traversal algorithms
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal
- Traversal is O(n)

 11 A-3

# Traversal of a Binary Tree




(a) Preorder: 60, 20, 10, 40, 30, 50, 70 (b) Inorder: 10, 20, 30, 40, 50, 60, 70 (c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

Figure 11-10

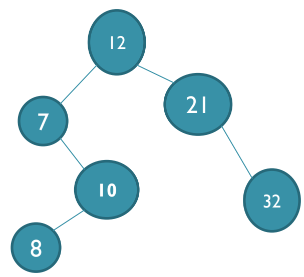
Traversals of a binary tree: a) preorder; b) inorder; c) postorder

 11 A-4

# Binary Search Tree

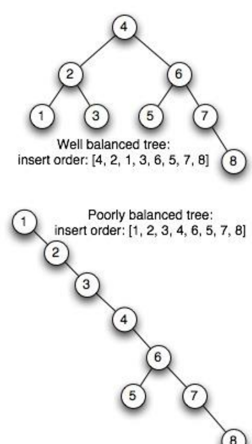
A binary tree that has the following properties for each node, n

- n's value is greater than all values in its left subtree $T_L$
- n's value is less than all values in its right subtree $T_R$
- Both $T_L$ and $T_R$ are binary search trees



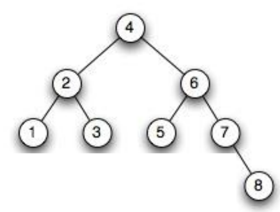

# BST Creation Example

- Create a BST with the following Data:
  1. 4, 2, 1, 3, 6, 5, 7, 8
  2. 1, 2, 3, 4, 6, 5, 7, 8



# The ADT Binary Search Tree

- Operations of the ADT binary search tree
  - Insert a new item into a binary search tree
  - Delete the item with a given search key from a binary search tree
  - Retrieve the item with a given search key from a binary search tree
  - Traverse the items in a binary search tree in preorder, inorder, or postorder




# Algorithms for the Operations of the ADT Binary Search Tree

- Since the binary search tree is recursive in nature, it is natural to formulate recursive algorithms for its operations
- A search algorithm
  - `search(bst, searchKey)`
    - Searches the binary search tree `bst` for the item whose search key is `searchKey`

# Algorithms for the Operations of the ADT Binary Search Tree: Insertion

- `insertItem(treeNode, newItem)`
  - Inserts `newItem` into the binary search tree of which `treeNode` is the root

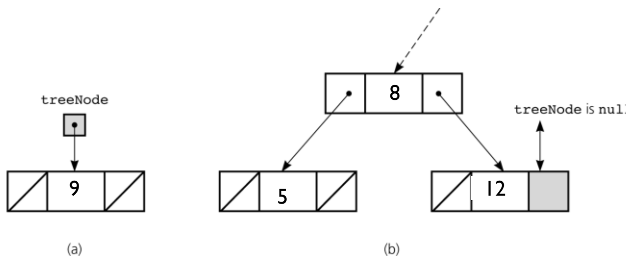



Figure 11-23a and 11-23b

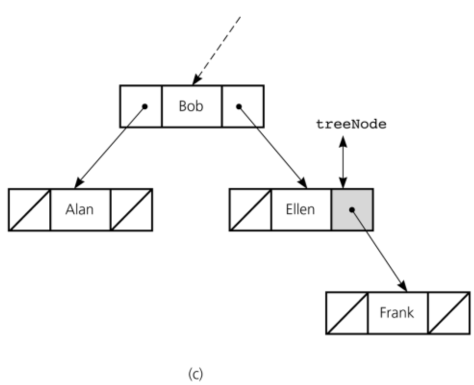
a) Insertion into an empty tree; b) search terminates at a leaf

 11 B-9

# Algorithms for the Operations of the ADT Binary Search Tree: Insertion

Figure 11-23c

c) insertion at a leaf




 11 B-10

# Algorithms for the Operations of the ADT Binary Search Tree: Deletion

- Steps for deletion
  - Use the search algorithm to locate the item with the specified key
  - If the item is found, remove the item from the tree
- Three possible cases for node N containing the item to be deleted
  - N is a leaf
  - N has only one child
  - N has two children

# Algorithms for the Operations of the ADT Binary Search Tree: Deletion

- Strategies for deleting node N
  - If N is a leaf
    - Set the reference in N's parent to `null`
  - If N has only one child
    - Let N's parent adopt N's child
  - If N has two children
    - Locate another node M that is easier to remove from the tree than the node N
    - Copy the item that is in M to N
    - Remove the node M from the tree

## Algorithms for the Operations of the ADT Binary Search Tree: Retrieval

- Retrieval operation can be implemented by refining the `search` algorithm
  - Return the item with the desired search key if it exists
  - Otherwise, return a `null` reference




## Algorithms for the Operations of the ADT Binary Search Tree: Traversal

- Traversals for a binary search tree are the same as the traversals for a binary tree
- Theorem 11-1

  The inorder traversal of a binary search tree T will visit its nodes in sorted search-key order

# The Efficiency of Binary Search Tree Operations

- The maximum number of comparisons for a retrieval, insertion, or deletion is the height of the tree
- The maximum and minimum heights of a binary search tree
  - n is the maximum height of a binary tree with n nodes

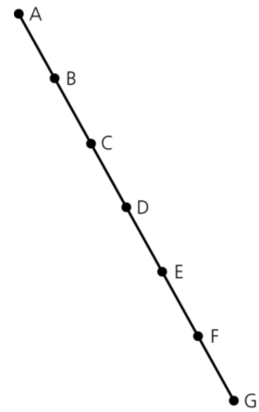



Figure 11-30
A maximum-height binary tree with seven nodes





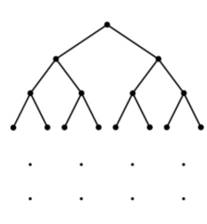
# The Efficiency of Binary Search Tree Operations

- Theorem 11-2

  A full binary tree of height $h \geq 0$ has $2^h - 1$ nodes
- Theorem 11-3

  The maximum number of nodes that a binary tree of height h can have is $2^h - 1$

Figure 11-32
Counting the nodes in a full binary tree of height *h*



| Level | Number of nodes at this level | Number of nodes at this and previous levels |
|---|---|---|
| 1 | $1 = 2^0$ | $1 = 2^1 - 1$ |
| 2 | $2 = 2^1$ | $3 = 2^2 - 1$ |
| 3 | $4 = 2^2$ | $7 = 2^3 - 1$ |
| 4 | $8 = 2^3$ | $15 = 2^4 - 1$ |
| . | . | . |
| . | . | . |
| . | . | . |
| h | $2^{h-1}$ | $2^h - 1$ |

# The Efficiency of Binary Search Tree Operations

- Theorem 11-4
  The minimum height of a binary tree with n nodes is $\lceil \log_2(n+1) \rceil$
- The height of a particular binary search tree depends on the order in which insertion and deletion operations are performed

| Operation | Average case | Worst case |
|---|---|---|
| Retrieval | O(log n) | O(n) |
| Insertion | O(log n) | O(n) |
| Deletion | O(log n) | O(n) |
| Traversal | O(n) | O(n) |

**Figure 11-34**
The order of the retrieval, insertion, deletion, and traversal operations for the reference-based implementation of the ADT binary search tree