Lecture 25: Trees

CS 0445: Data Structures

Constantinos Costa

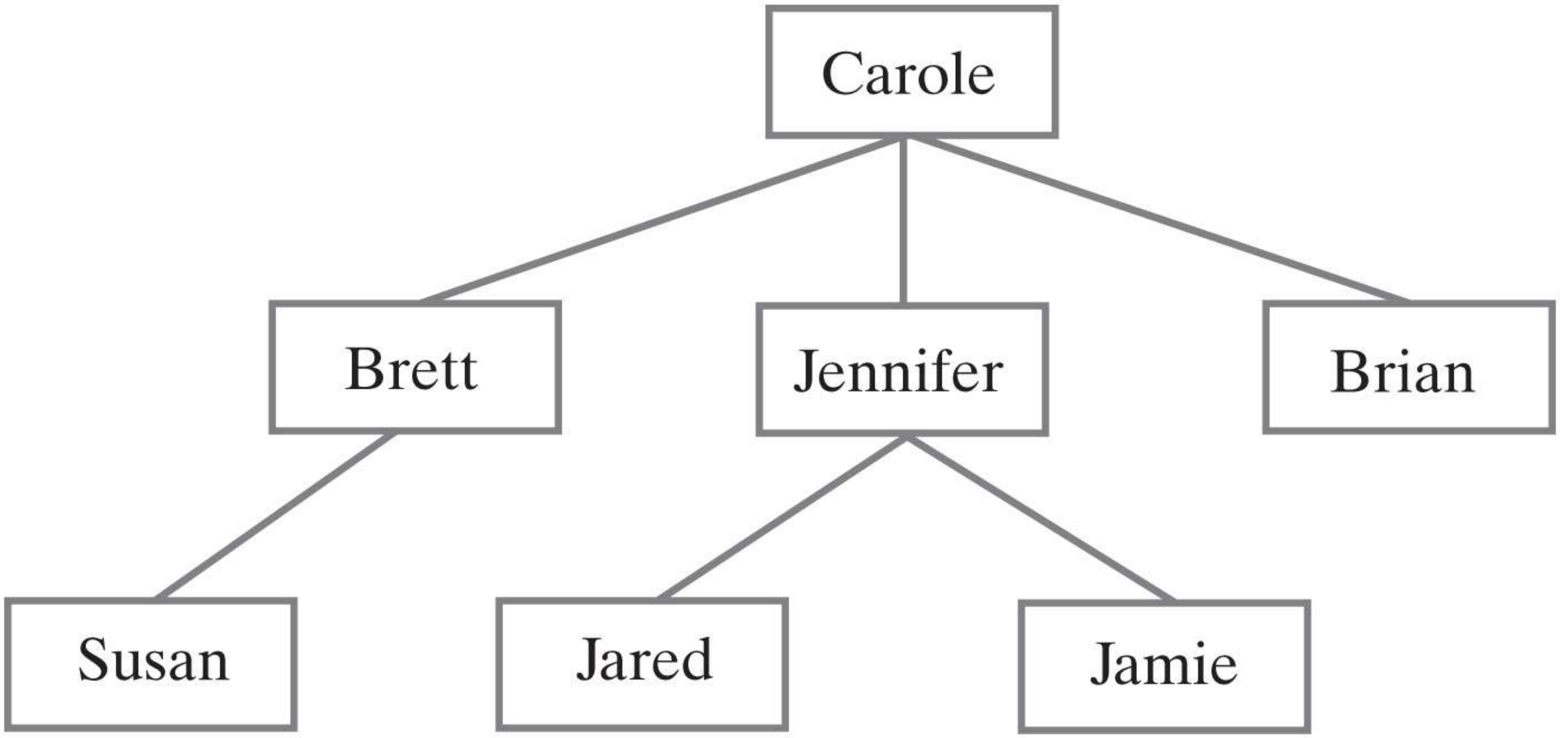
http://db.cs.pitt.edu/courses/cs0445/current.term/

Nov 07, 2019, 8:00-9:15

University of Pittsburgh, Pittsburgh, PA

Hierarchical Organizations

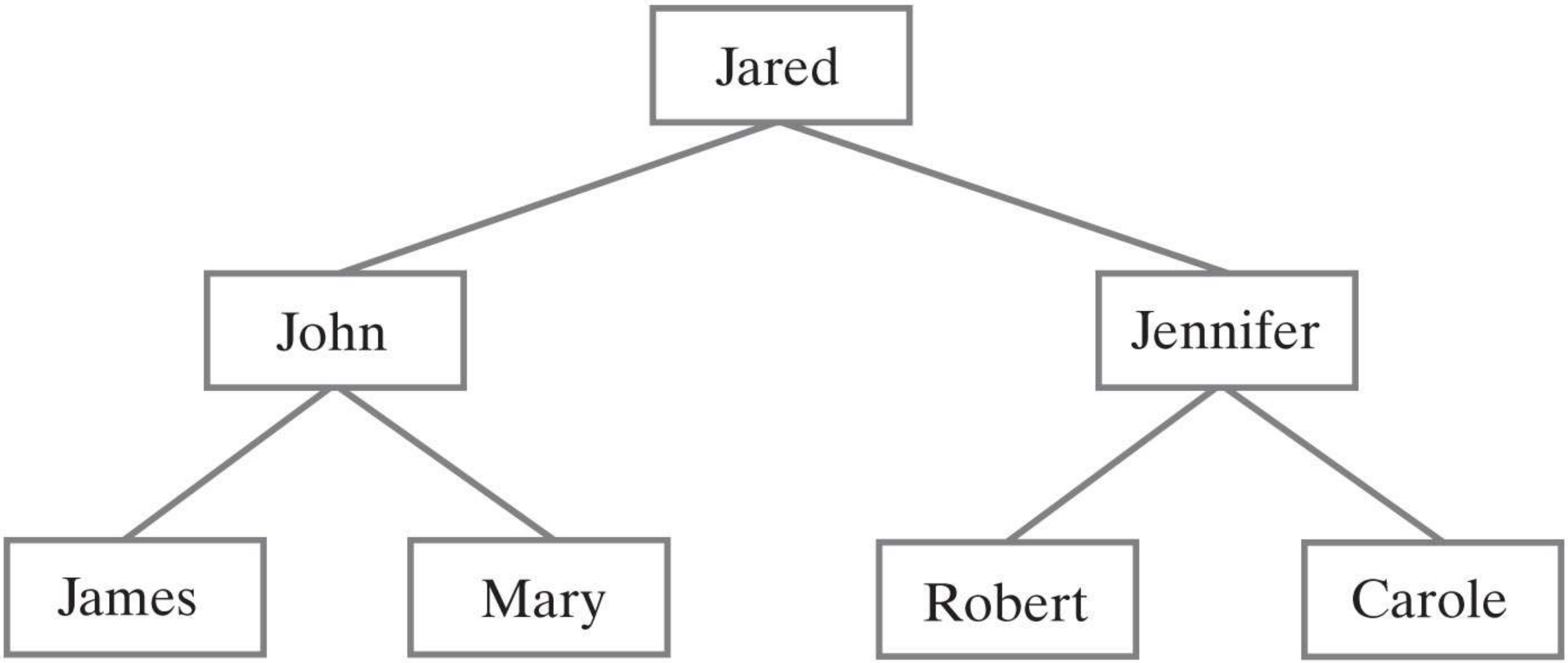
- Carole’s children and grandchildren



© 2019 Pearson Education, Inc.

Hierarchical Organizations

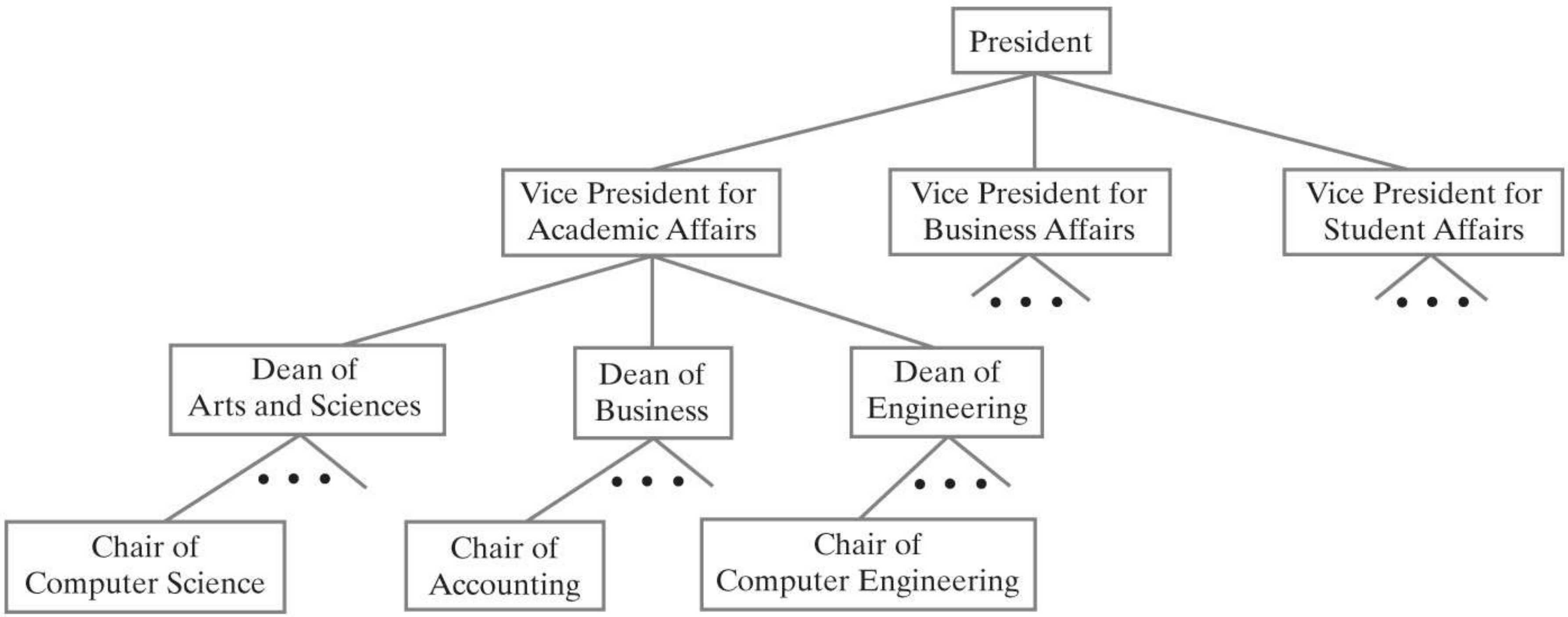
- Jared's parents and grandparents



© 2019 Pearson Education, Inc.

Hierarchical Organizations

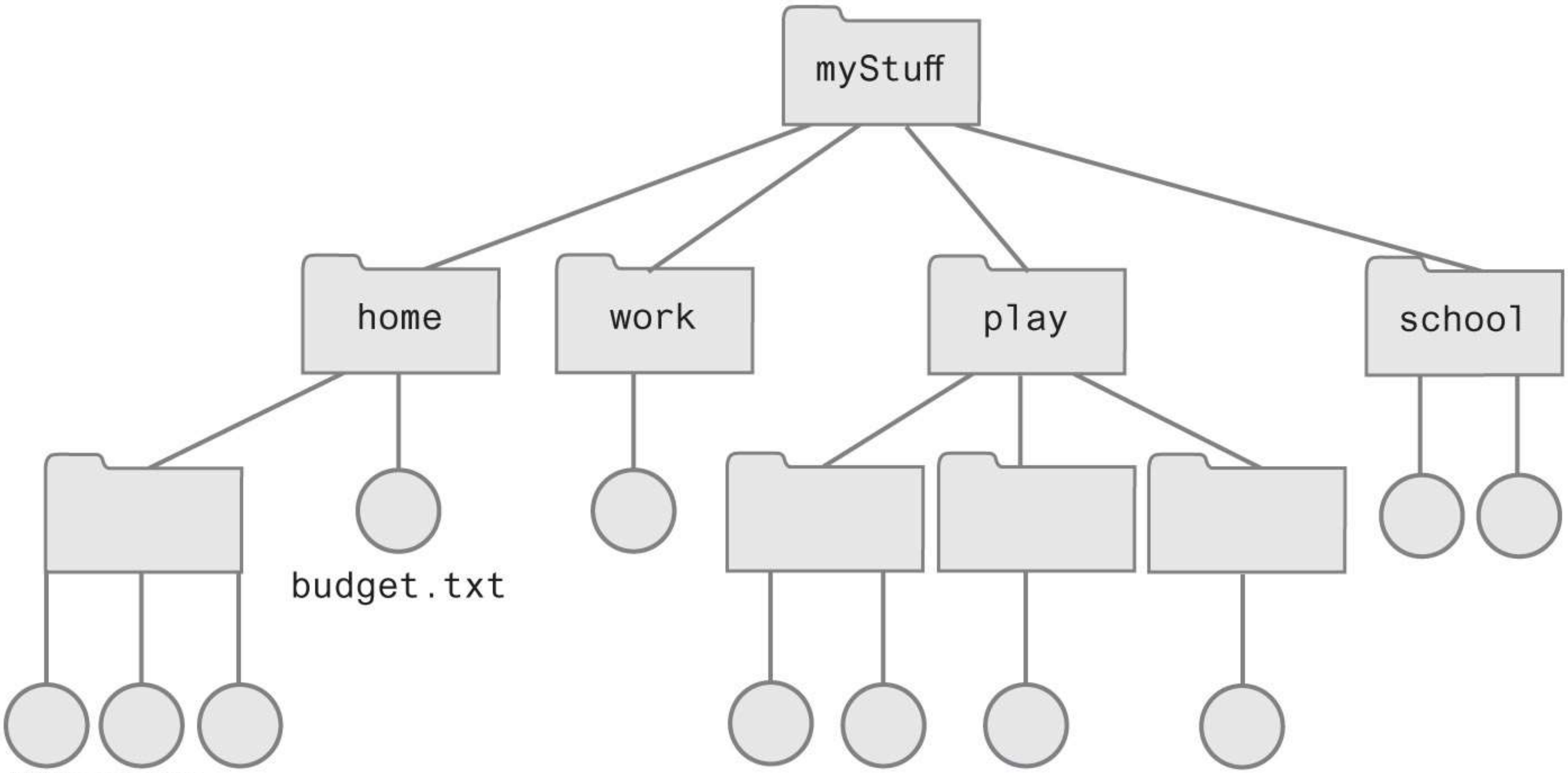
- A portion of a university's administrative structure



© 2019 Pearson Education, Inc.

Hierarchical Organizations

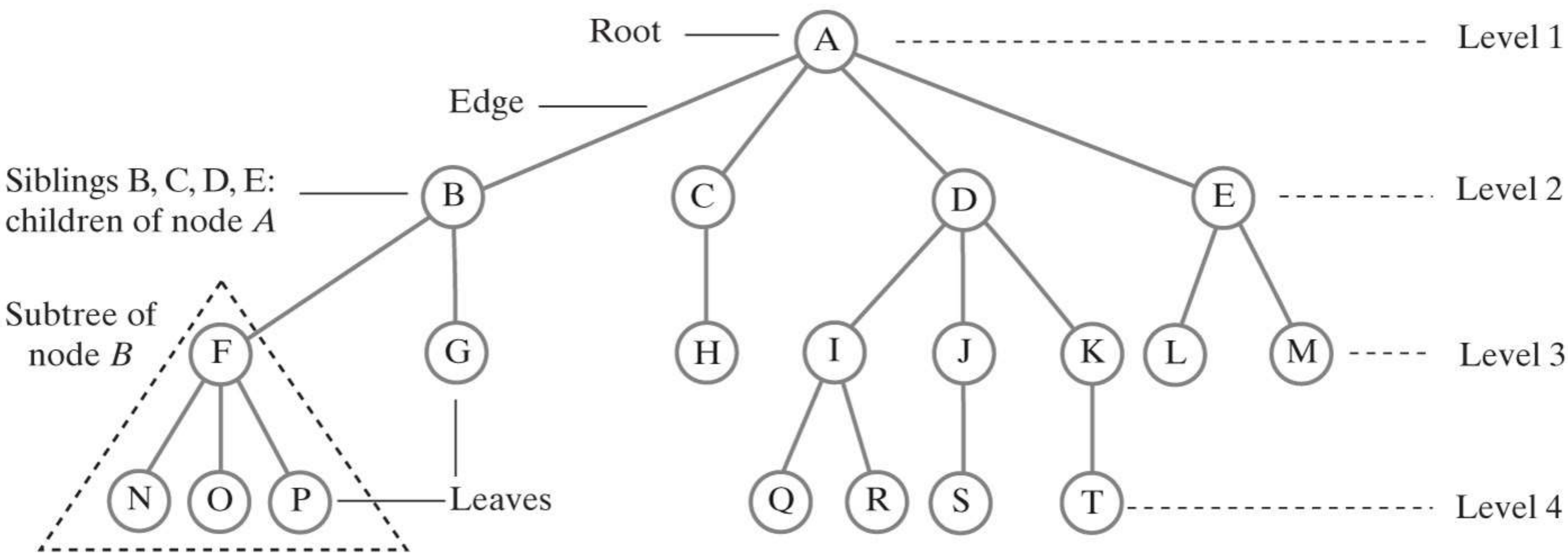
- Computer files organized into folders



© 2019 Pearson Education, Inc.

Tree Terminology

- A tree equivalent to the tree in previous figure

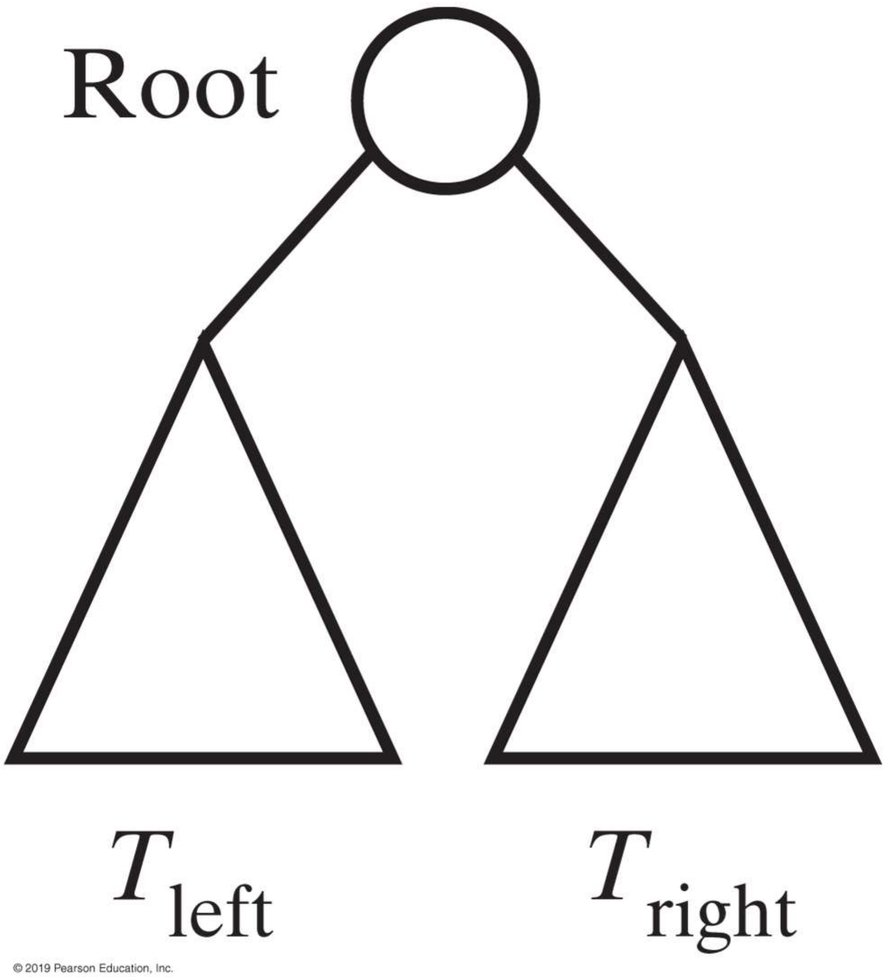


© 2019 Pearson Education, Inc.

Tree Terminology

- Contrast plants with root at bottom
  - ADT tree with root at top
  - Root is only node with no parent
- A tree can be empty
- Any node and its descendants form a subtree of the original tree
- The height of a tree is the number of levels in the tree

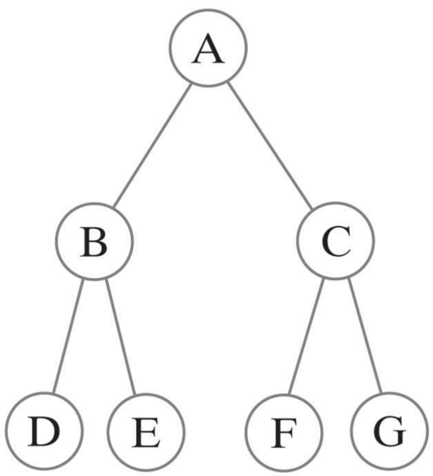
Binary trees



Binary Trees

- Three binary trees

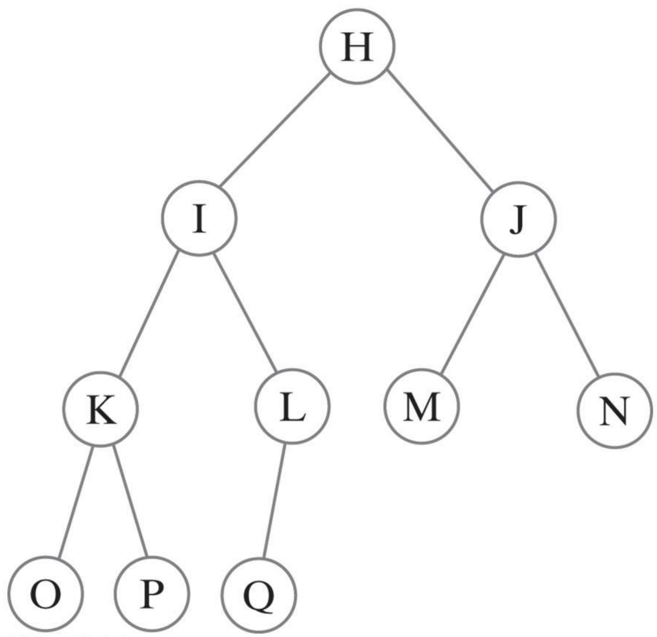
(a) Full tree



Left children: B, D, F
Right children: C, E, G

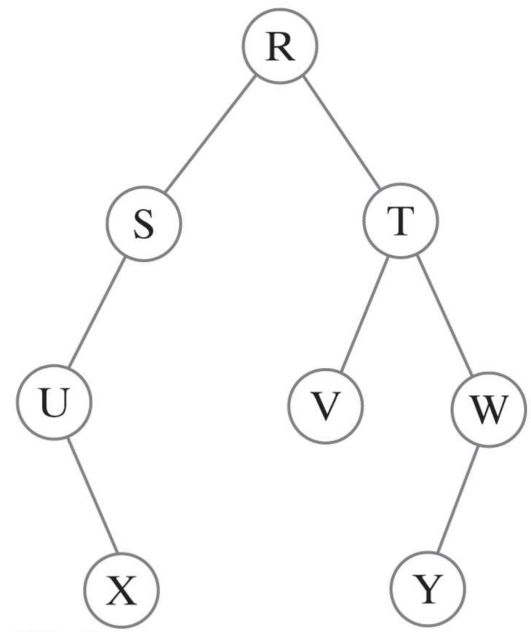
© 2019 Pearson Education, Inc.

(b) Complete tree



© 2019 Pearson Education, Inc.

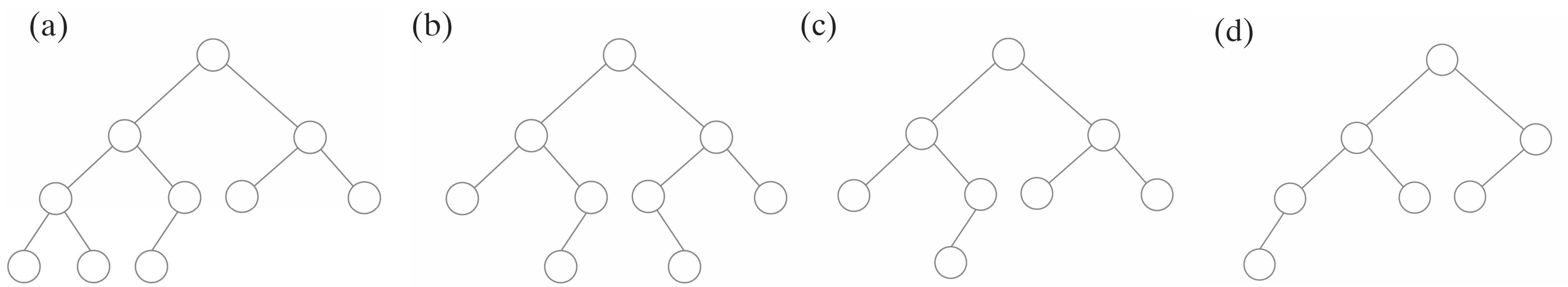
(c) Tree that is not full and not complete



© 2019 Pearson Education, Inc.

Binary Trees

- Some binary trees that are height balanced



Balanced and complete

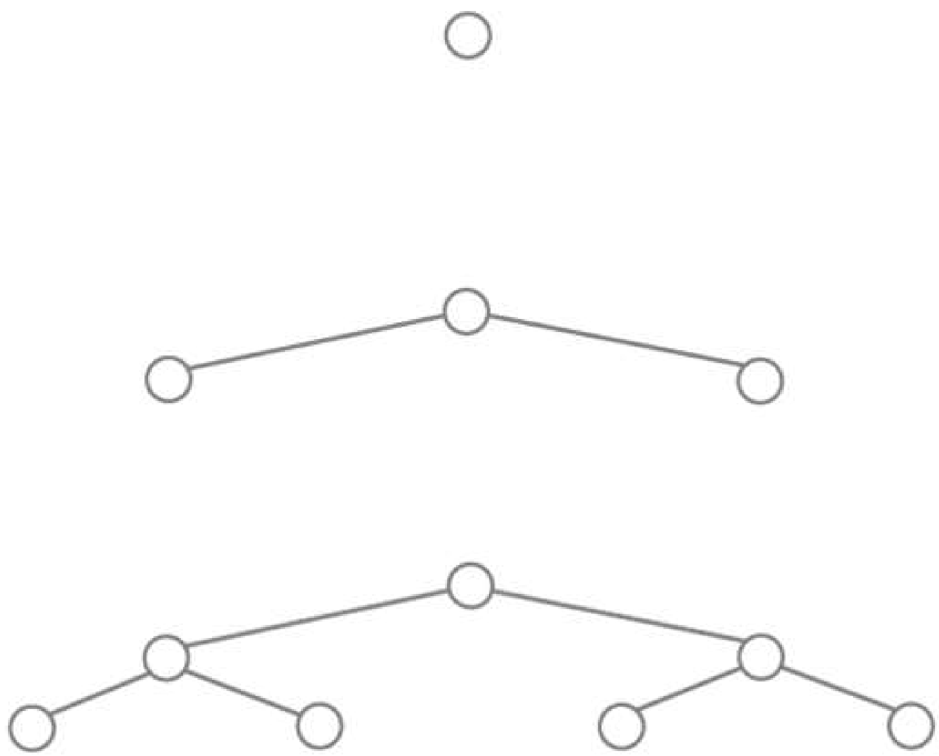
© 2019 Pearson Education, Inc.

Balanced, but not complete

Binary Tree Height (Part 1)

- The number of nodes in a full binary tree as a function of the tree's height

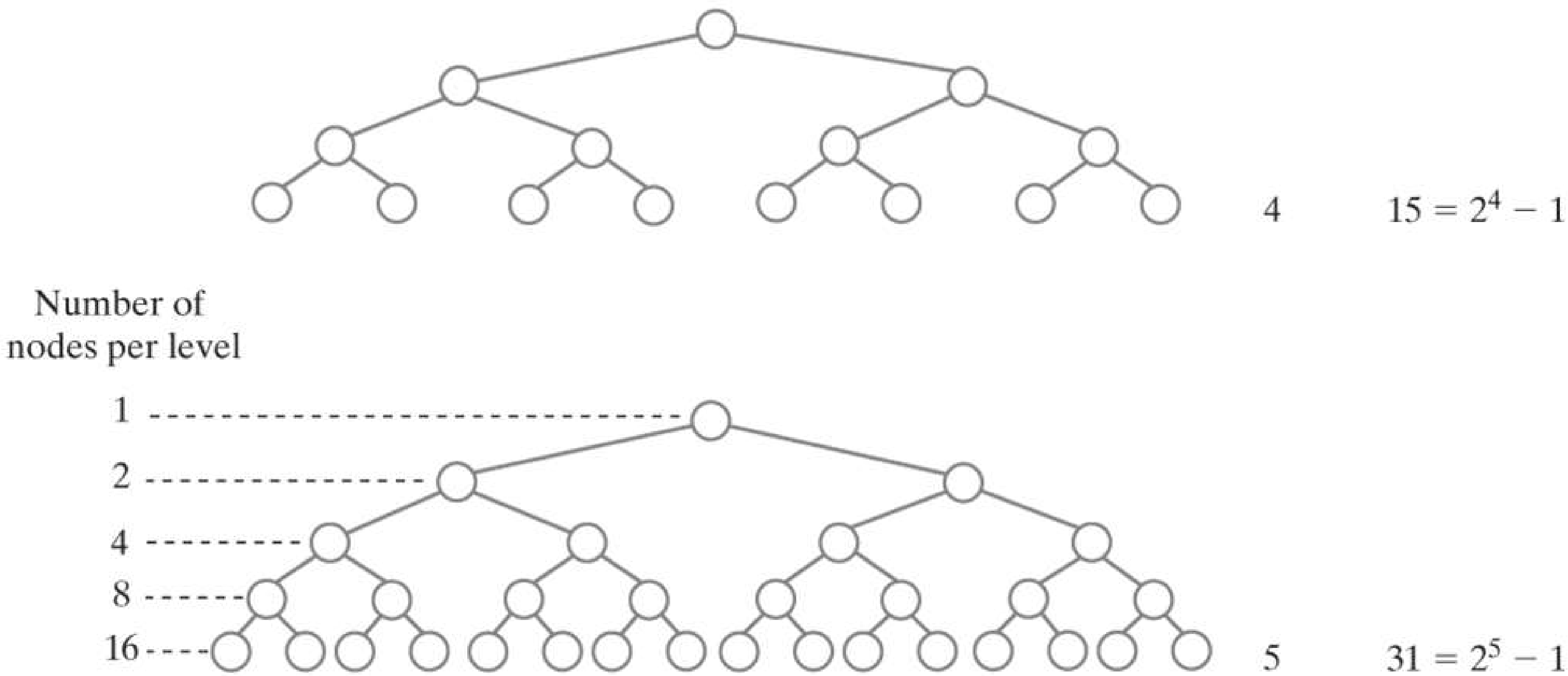
Full Tree



Height	Number of Nodes
1	$1 = 2^1 - 1$
2	$3 = 2^2 - 1$
3	$7 = 2^3 - 1$

Binary Tree Height (Part 2)

- The number of nodes in a full binary tree as a function of the tree's height



© 2019 Pearson Education, Inc.

Traversals of A Tree

- Traversal:
  - Visit, or process, each data item exactly once
- We will say that traversal can pass through a node without visiting it at that moment.
- Order in which we visit items is not unique
- Traversals of a binary tree are somewhat easy to understand

Traversals of a Binary Tree

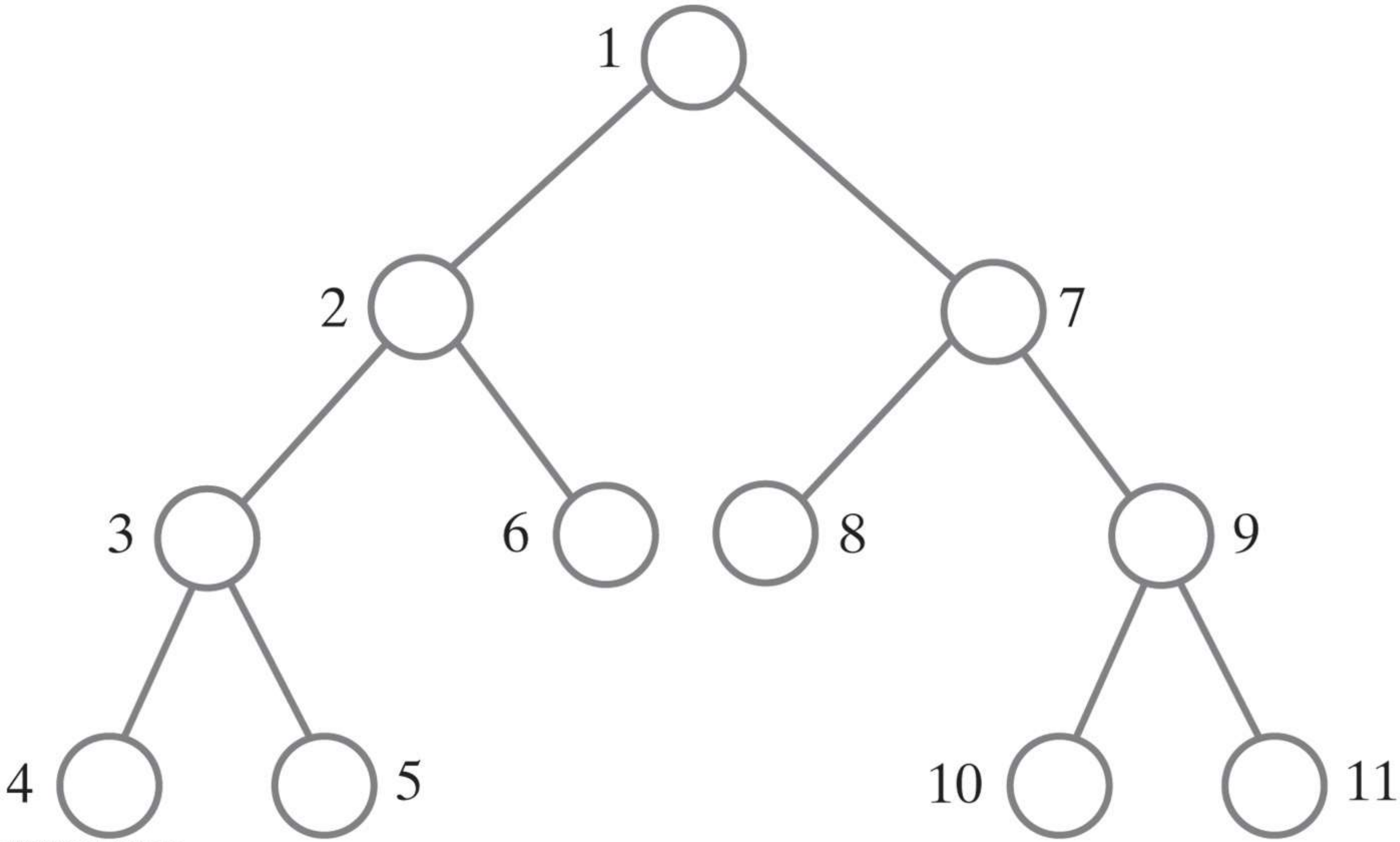
- We use recursion
- To visit all the nodes in a binary tree, we must
  - Visit the root
  - Visit all the nodes in the root's left subtree
  - Visit all the nodes in the root's right subtree

Traversals of a Binary Tree

- **Preorder traversal**
  - Visit root before we visit root’s subtrees
- **Inorder traversal**
  - Visit root of a binary tree between visiting nodes in root’s subtrees.
- **Postorder traversal**
  - Visit root of a binary tree after visiting nodes in root’s subtrees
- **Level-order traversal**
  - Begin at root and visit nodes one level at a time

Traversals of a Binary Tree

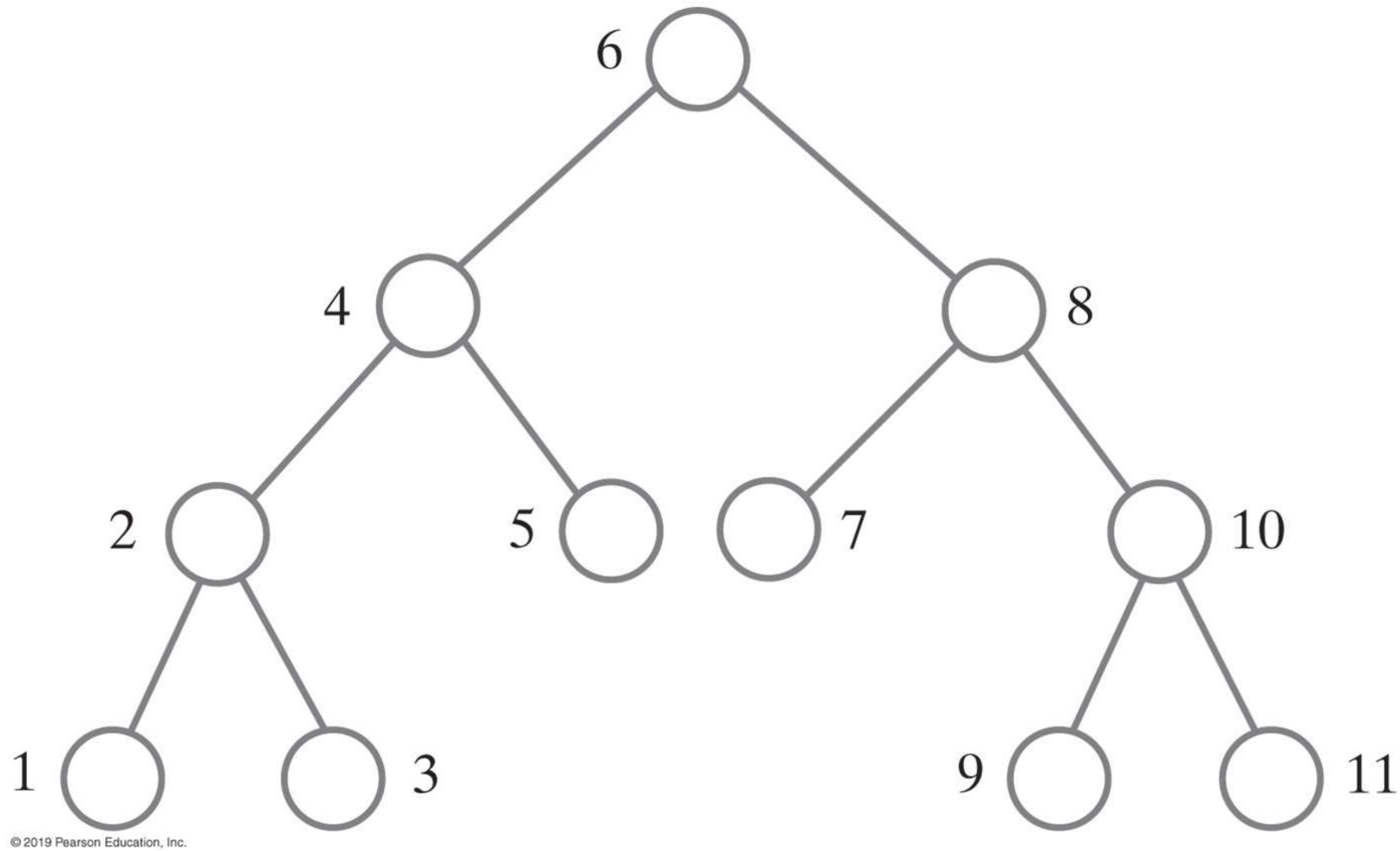
- The visitation order of a preorder traversal



© 2019 Pearson Education, Inc.

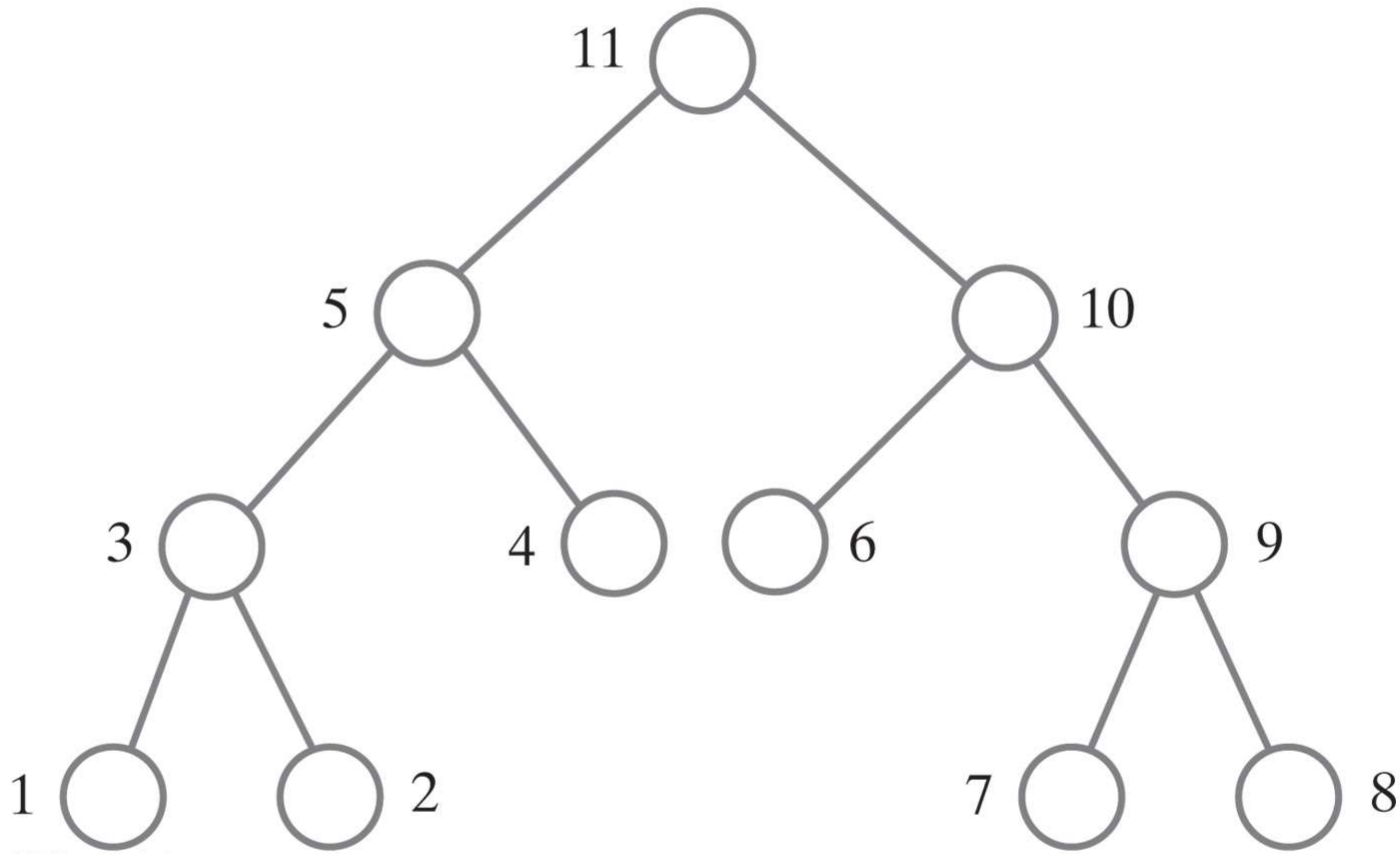
Traversals of a Binary Tree

- The visitation order of an in-order traversal



Traversals of a Binary Tree

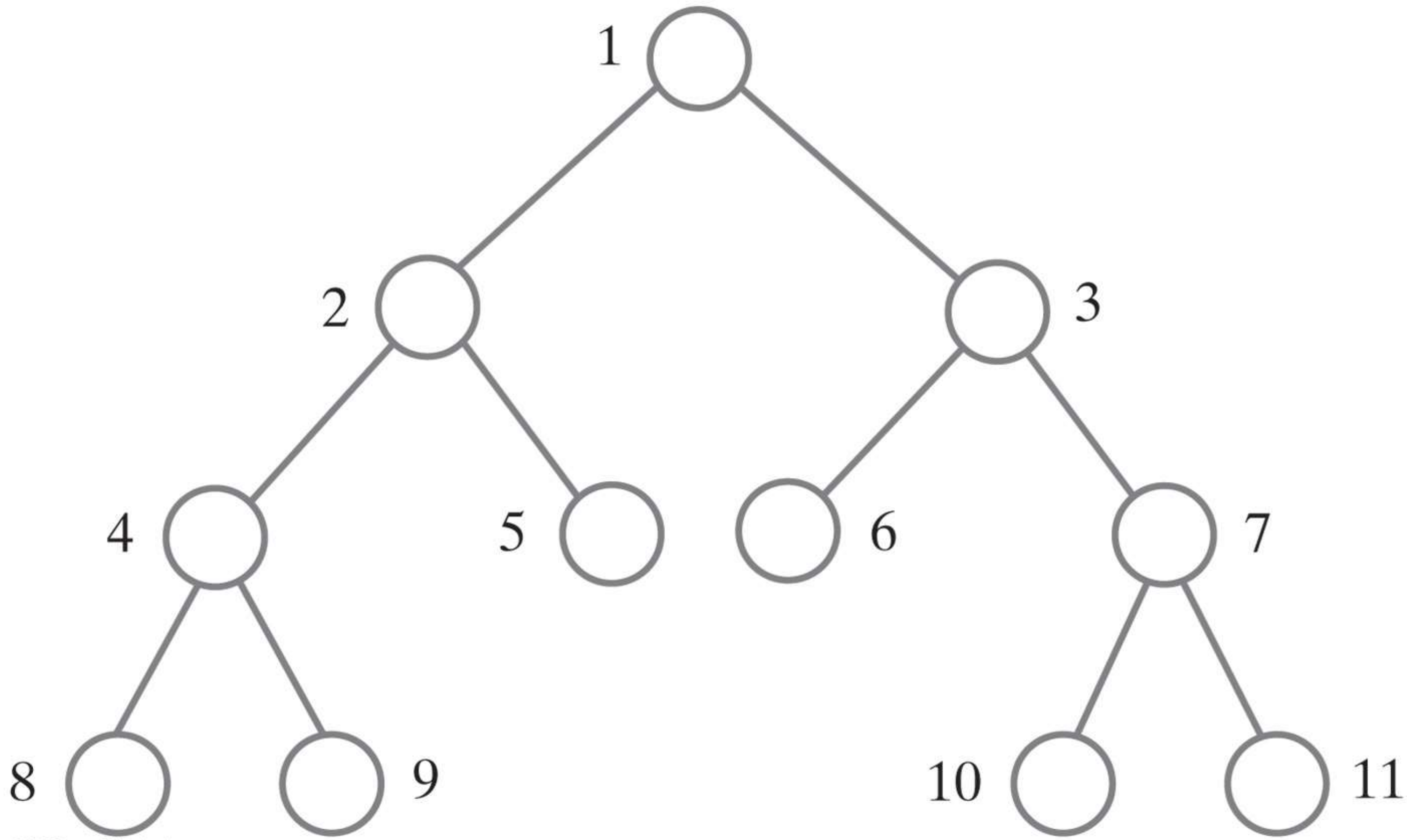
- The visitation order of a postorder traversal



© 2019 Pearson Education, Inc.

Traversals of a Binary Tree

- The visitation order of a level-order traversal



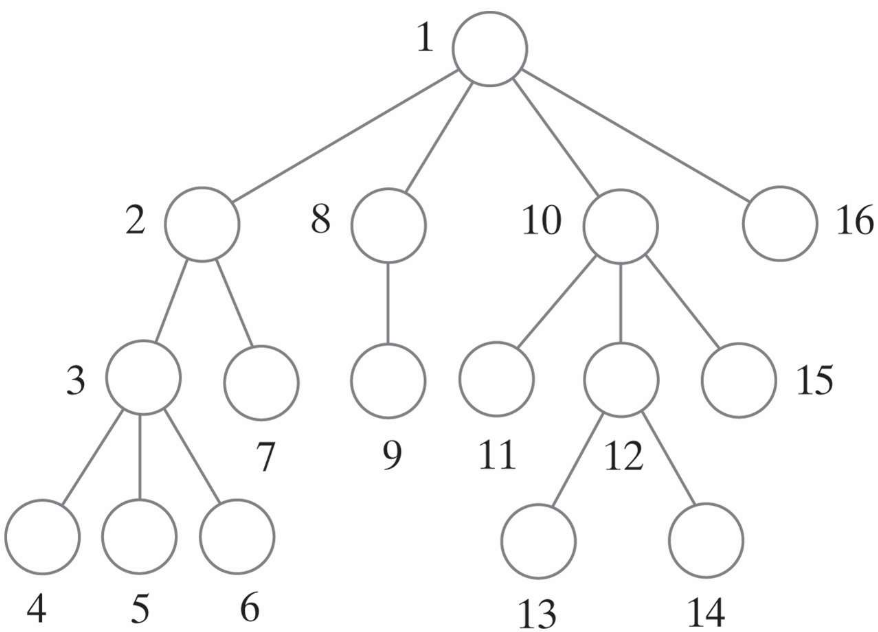
© 2019 Pearson Education, Inc.

Traversals of a General Tree

- Types of traversals for general tree
  - Level order
  - Preorder
  - Postorder
- Not suited for general tree traversal
  - Inorder

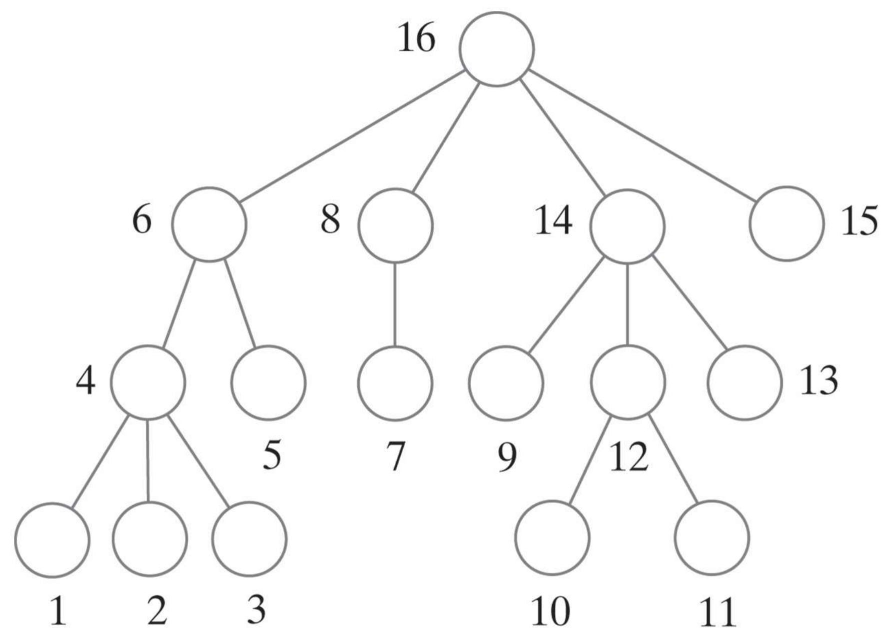
Traversals of a General Tree

- The visitation order of two traversals of a general tree



(a) Preorder traversal

© 2019 Pearson Education, Inc.



(b) Postorder traversal

© 2019 Pearson Education, Inc.

Interfaces for All Trees

- An interface of methods common to all trees

```java
package TreePackage;
/** An interface of basic methods for the ADT tree.  */
public interface TreeInterface<T>
{
   public T getRootData();
   public int getHeight();
   public int getNumberOfNodes();
   public boolean isEmpty();
   public void clear();
} // end TreeInterface
```

Traversals

- An interface of traversal methods for a tree

```
package TreePackage;
import java.util.Iterator;
/** An interface of iterators for the ADT tree. */
public interface TreeIteratorInterface<T>
{
   public Iterator<T> getPreorderIterator();
   public Iterator<T> getPostorderIterator();
   public Iterator<T> getInorderIterator();
   public Iterator<T> getLevelOrderIterator();
} // end TreeIteratorInterface
```

Interface for Binary Trees

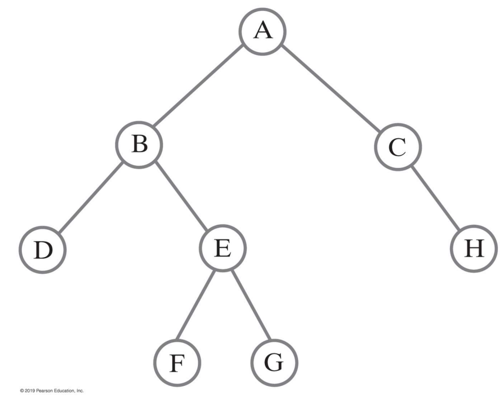
- An interface for a binary tree

```
package TreePackage;
/*   An interface for the ADT binary tree. */
public interface BinaryTreeInterface<T> extends TreeInterface<T>,
                                  TreeIteratorInterface<T>
{
  /** Sets the data in the root of this binary tree.
     @param rootData  The object that is the data for the tree's root.
  */
  public void setRootData(T rootData);

  /** Sets this binary tree to a new binary tree.
    @param rootData   The object that is the data for the new tree's root.
    @param leftTree   The left subtree of the new tree.
    @param rightTree  The right subtree of the new tree. */
  public void setTree(T rootData, BinaryTreeInterface<T> leftTree,
                      BinaryTreeInterface<T> rightTree);
} // end BinaryTreeInterface
```

Building a Binary Tree

- Java statements that build a tree

```
BinaryTreeInterface<String> dTree = new BinaryTree<>();
dTree.setTree("D", null, null);

BinaryTreeInterface<String> fTree = new BinaryTree<>();
fTree.setTree("F", null, null);

BinaryTreeInterface<String> gTree = new BinaryTree<>();
gTree.setTree("G", null, null);

BinaryTreeInterface<String> hTree = new BinaryTree<>();
hTree.setTree("H", null, null);

BinaryTreeInterface<String> emptyTree = new BinaryTree<>();

// Form larger subtrees
BinaryTreeInterface<String> eTree = new BinaryTree<>();
eTree.setTree("E", fTree, gTree); // Subtree rooted at E

BinaryTreeInterface<String> bTree = new BinaryTree<>();
bTree.setTree("B", dTree, eTree); // Subtree rooted at B

BinaryTreeInterface<String> cTree = new BinaryTree<>();
cTree.setTree("C", emptyTree, hTree); // Subtree rooted at C

BinaryTreeInterface<String> aTree = new BinaryTree<>();
aTree.setTree("A", bTree, cTree); // Desired tree rooted at A
```



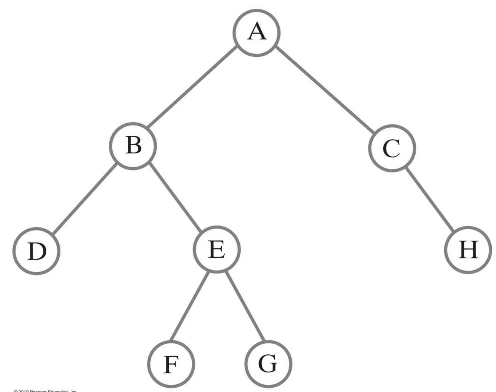
A binary tree whose nodes contain one-letter strings

Building a Binary Tree

- Java statements that build a tree and then display some of its characteristics:

```java
// Display root, height, number of nodes
System.out.println("Root of tree contains " + aTree.getRootData());
System.out.println("Height of tree is " + aTree.getHeight());
System.out.println("Tree has " + aTree.getNumberOfNodes() + " nodes");

// Display nodes in preorder
System.out.println("A preorder traversal visits nodes in this order:");
Iterator<String> preorder = aTree.getPreorderIterator();
while (preorder.hasNext())
  System.out.print(preorder.next() + " ");
System.out.println();
```

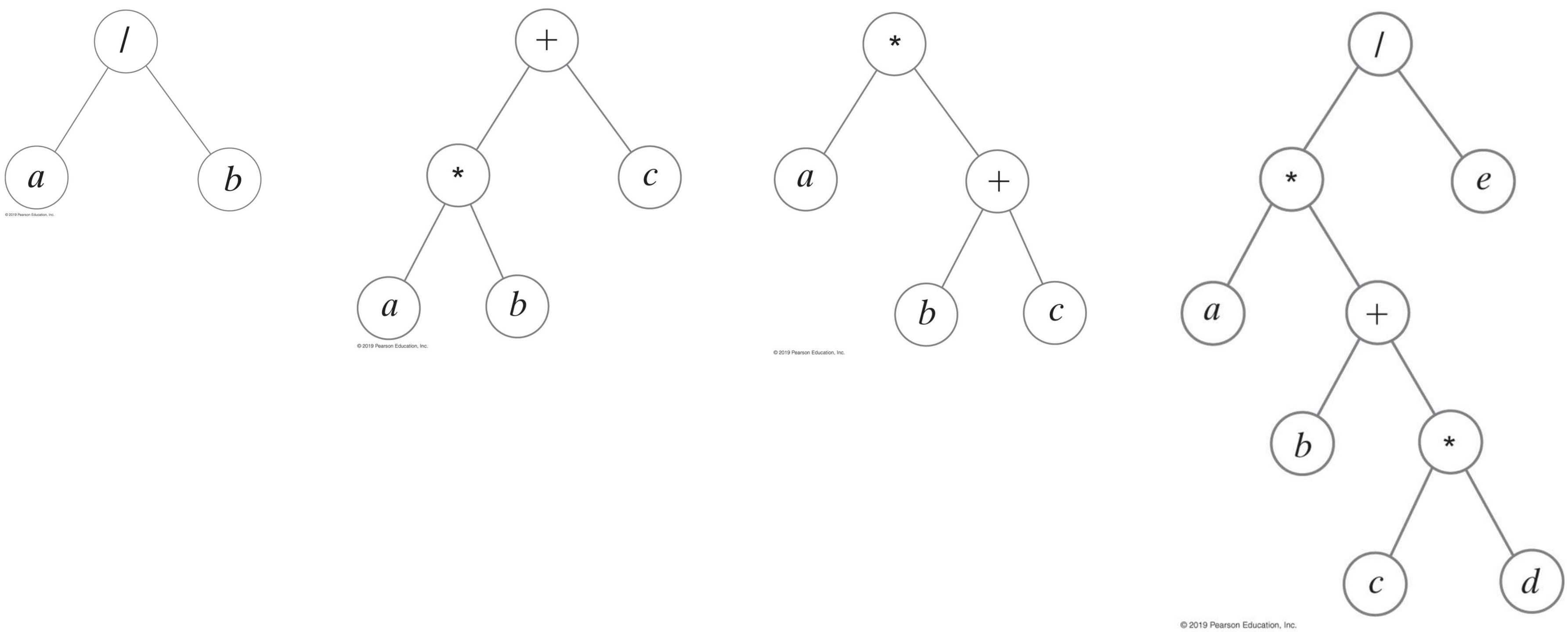
A binary tree whose nodes contain one-letter strings



Expression Trees

- Expression trees for four algebraic expressions

(a) a / b (b) $a * b + c$ (c) $a * (b + c)$ (d) $a * (b + c * d) / e$



Expression Trees

- Algorithm for postorder traversal of an expression tree.

Algorithm **evaluate(expressionTree)**

if (expressionTree *is empty*)

return 0

else

{

firstOperand = evaluate(*left subtree of* **expressionTree)**

secondOperand = evaluate(*right subtree of* **expressionTree)**

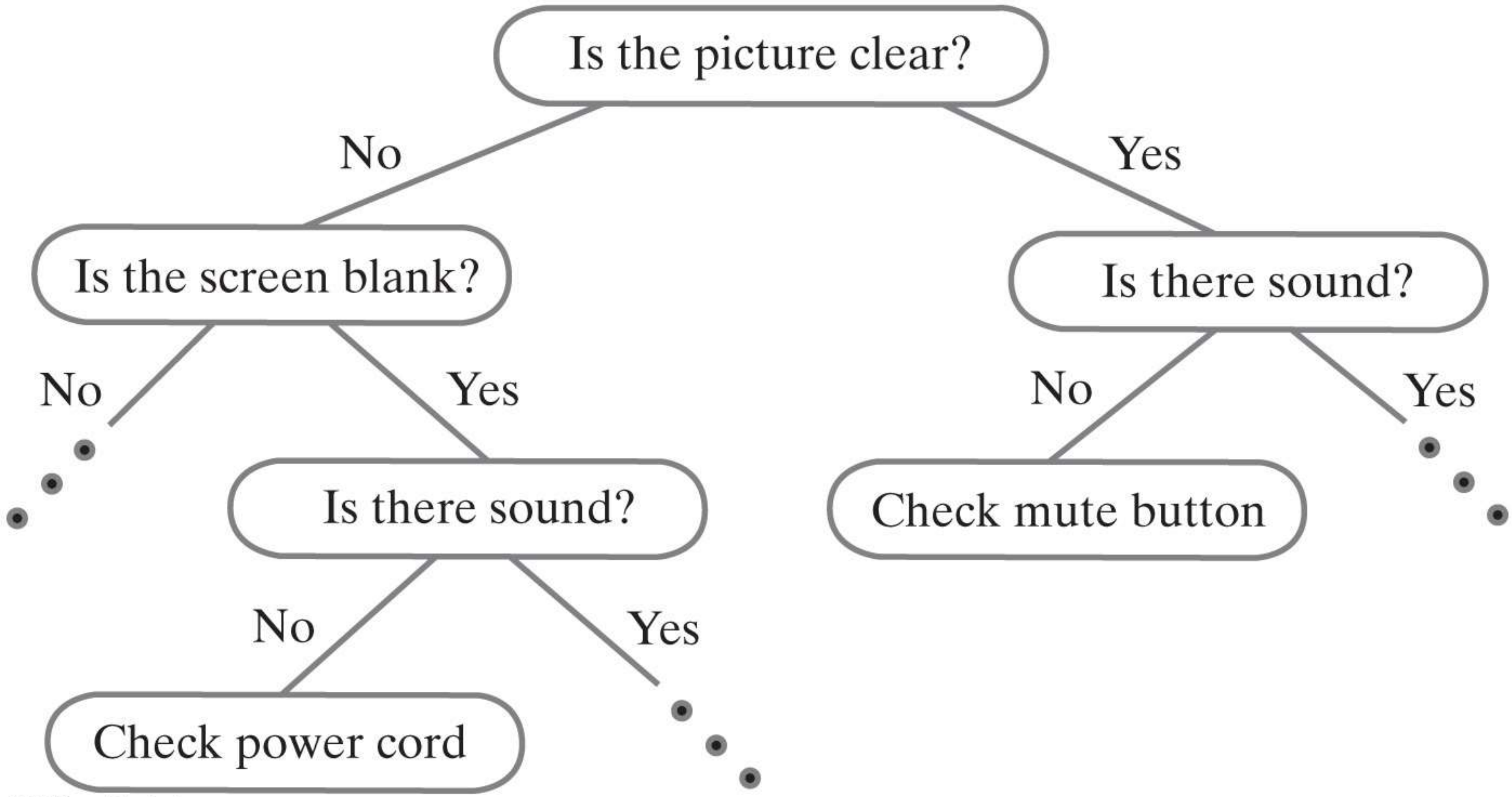
operator = *the root of* **expressionTree**

return *the result of the operation* **operator** *and its operands* **firstOperand** *and secondOperand*

}

Expert System Using A Decision Tree

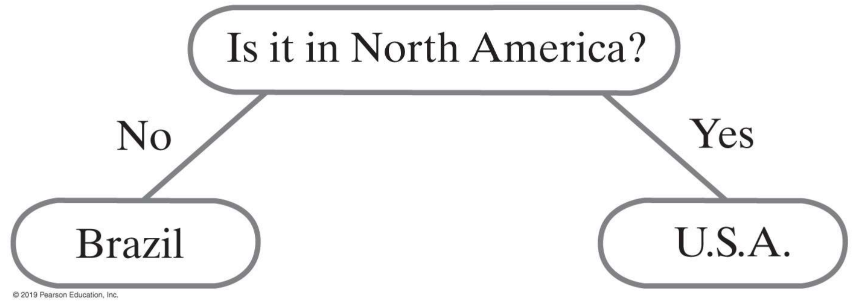
- A portion of a binary decision tree

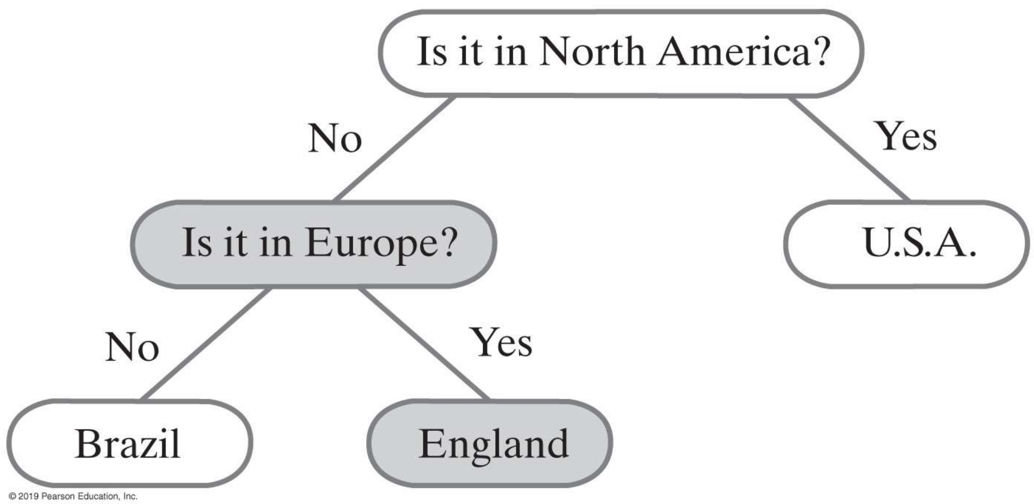


© 2019 Pearson Education, Inc.

Expert System Using A Decision Tree

- An initial decision tree for a guessing game





The decision tree for a guessing game after acquiring another fact

Expert System Using A Decision Tree (Part 1)

- An interface for a binary decision tree

```
package TreePackage;
/** An interface for a decision tree. */
public interface DecisionTreeInterface<T> extends BinaryTreeInterface<T>
{
  /** Gets the data in the current node.
     @return  The data object in the current node, or
             null if the current node is null. */
  public T getCurrentData();

  /** Sets the data in the current node.
     Precondition: The current node is not null.
     @param newData  The new data object. */
  public void setCurrentData(T newData);

  /** Sets the data in the children of the current node,
     creating them if they do not exist.
     Precondition: The current node is not null.
     @param responseForNo   The new data object for the left child.
     @param responseForYes  The new data object for the right child. */
  public void setResponses(T responseForNo, T responseForYes);
```

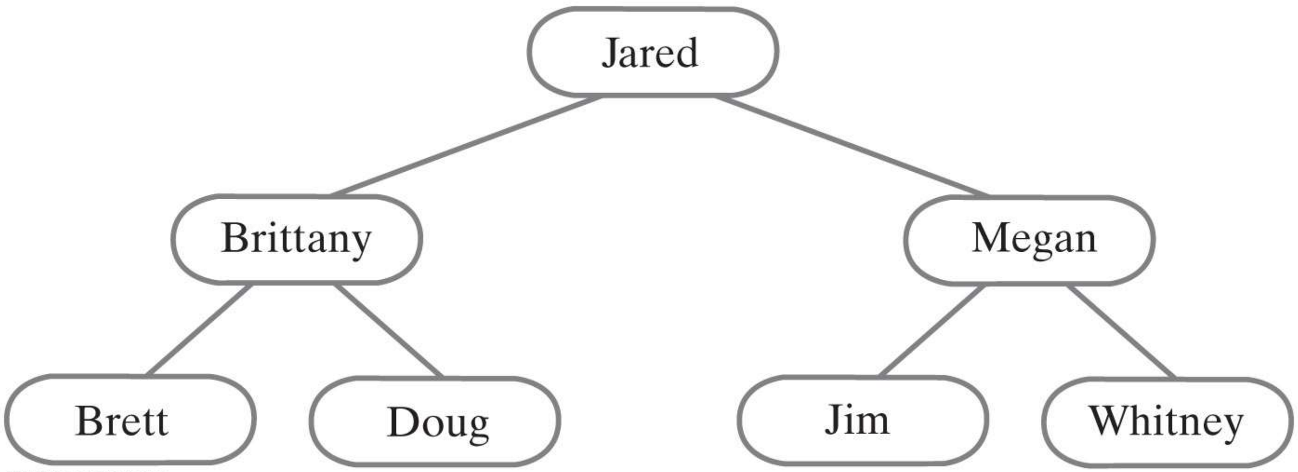
Expert System Using A Decision Tree (Part 2)

- An interface for a binary decision tree

```java
   /** Sees whether the current node contains an answer.
     @return  True if the current node is a leaf, or
              false if it is a nonleaf. */
   public boolean isAnswer();

   /** Sets the current node to its left child.
      If the child does not exist, sets the current node to null. */
   public void advanceToNo();

   /** Sets the current node to its right child.
      If the child does not exist, sets the current node to null. */
   public void advanceToYes();

   /** Sets the current node to the root of the tree. */
   public void resetCurrentNode();
} // end DecisionTreeInterface
```

Binary Search Tree

- For each node in a binary search tree
  - Node's data is greater than all data in node's left subtree
  - Node's data is less than all data in node's right subtree
- Every node in a binary search tree is the root of a binary search tree



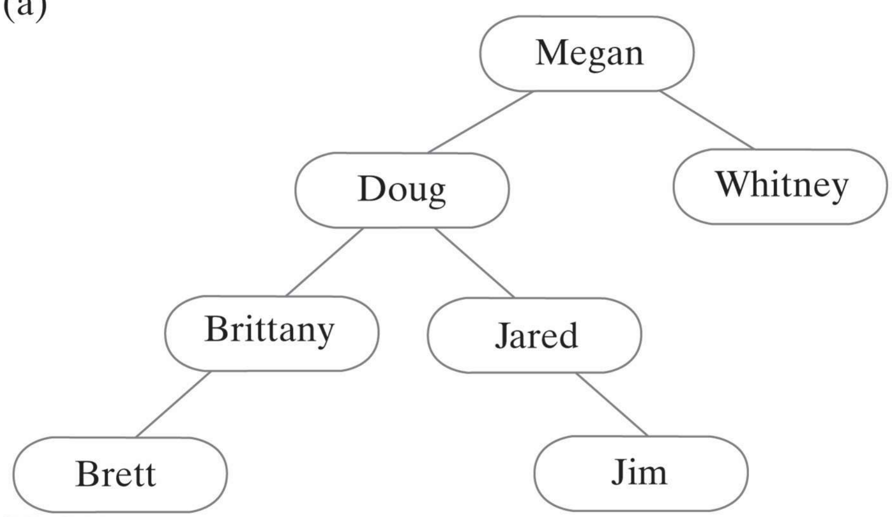
© 2019 Pearson Education, Inc.

A binary search tree of names

Binary Search Tree

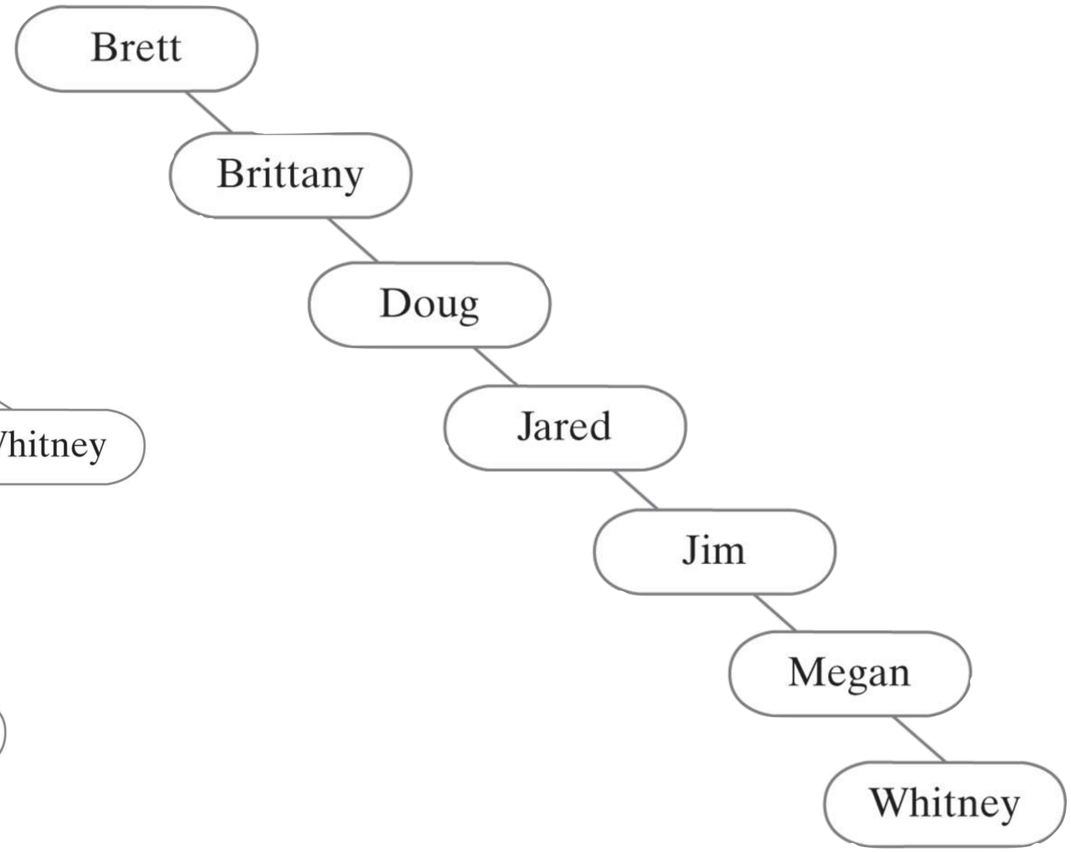
- Two binary search trees containing the same data as the tree in previous figure

(a)



© 2019 Pearson Education, Inc.

(b)



© 2019 Pearson Education, Inc.

Binary Search Tree

- Pseudocode for recursive search algorithm

Algorithm **bstSearch(binarySearchTree, desiredObject)**

// Searches a binary search tree for a given object.

//Returns true if the object is found.

if (binarySearchTree *is empty***)**

  return false

else if (desiredObject == *object in the root of* **binarySearchTree)**

  return true

else if (desiredObject < *object in the root of* **binarySearchTree)**

  return bstSearch(*left subtree of* **binarySearchTree, desiredObject)**

else

  return bstSearch(*right subtree of* **binarySearchTree, desiredObject)**

Binary Search Tree

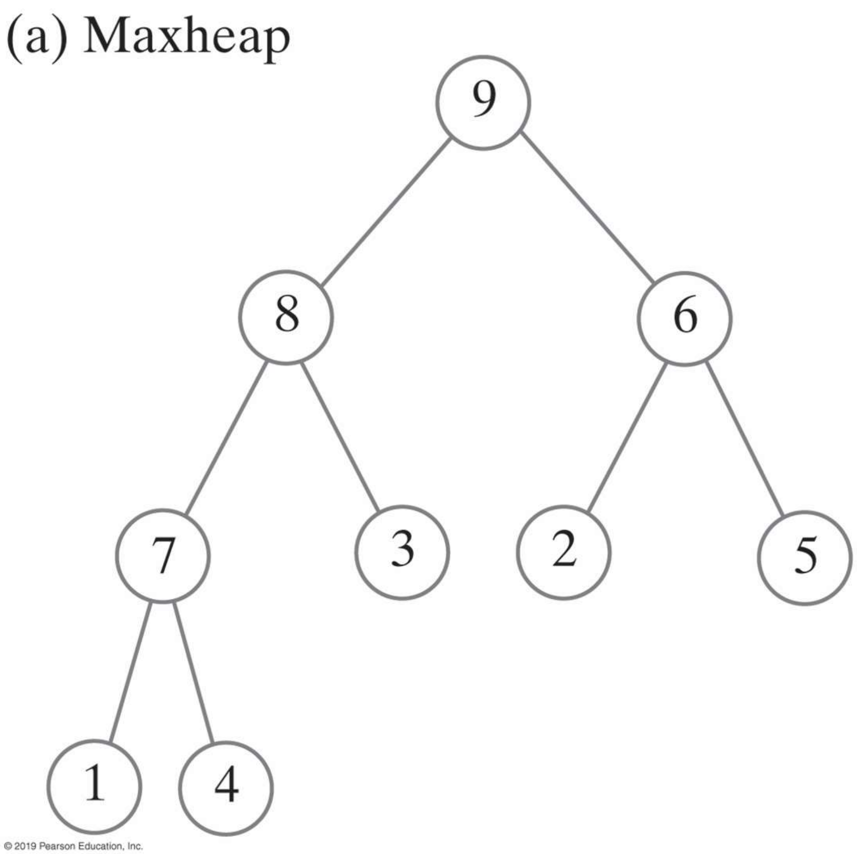
- Efficiency of a search
  - Searching a binary search tree of height h is $O(h)$
- To make searching a binary search tree efficient:
  - Tree must be as short as possible.

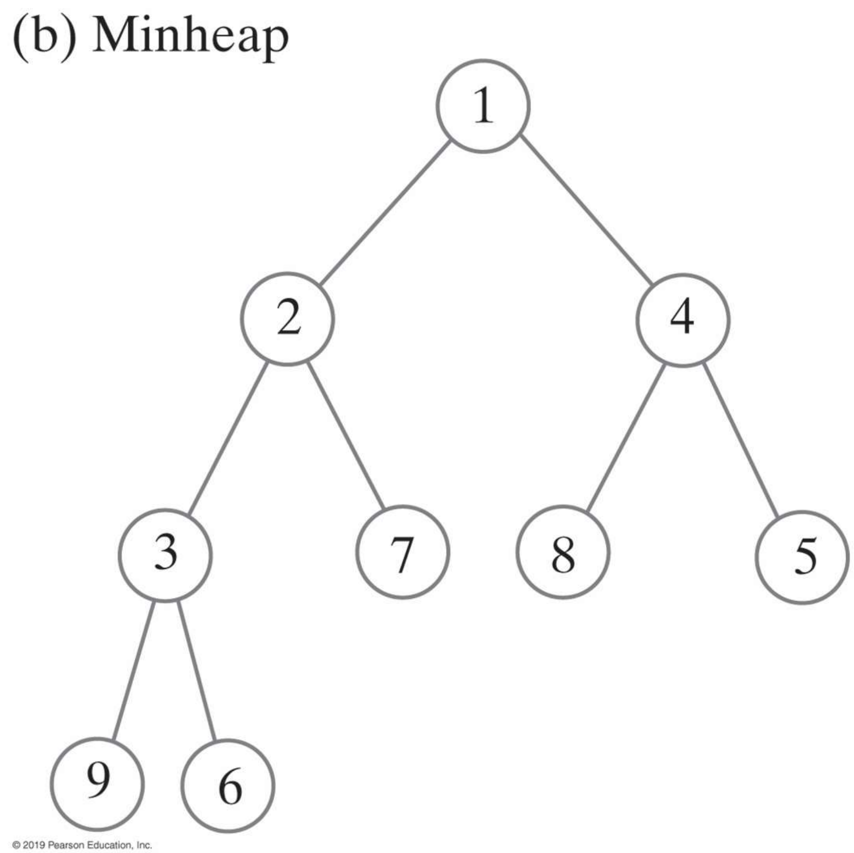
Heaps

- Complete binary tree whose nodes contain `Comparable` objects and are organized as follows:
  - Each node contains an object no smaller/larger than objects in its descendants
  - ***Maxheap***: object in node greater than or equal to its descendant objects
  - ***Minheap***: object in node less than or equal to its descendant objects

Heaps

- Two heaps that contain the same values





Heaps

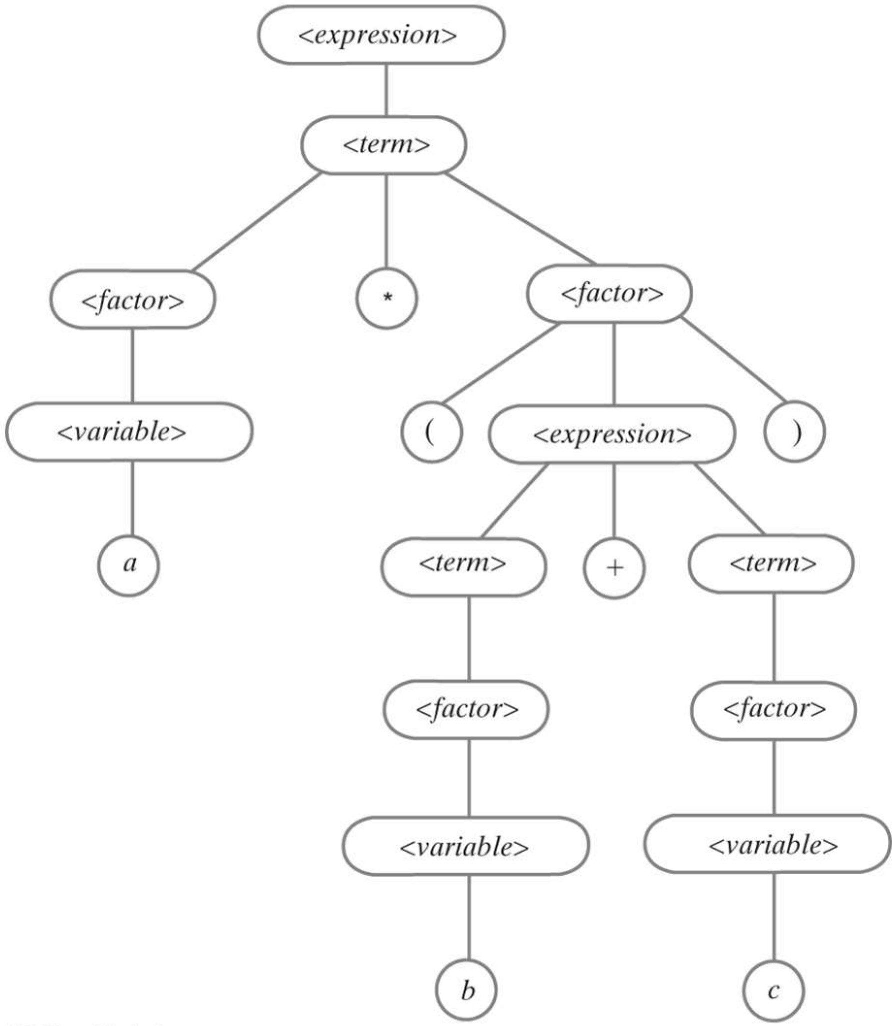
```
/**  An interface for the ADT maxheap. */
public interface MaxHeapInterface<T extends Comparable<? super T>>
{
  /** Adds a new entry to this heap.
     @param newEntry  An object to be added. */
  public void add(T newEntry);

  /** Removes and returns the largest item in this heap.
     @return  Either the largest object in the heap or,
              if the heap is empty before the operation, null. */
  public T removeMax();

  /** Retrieves the largest item in this heap.
     @return  Either the largest object in the heap or,
              if the heap is empty, null. */
  public T getMax();

  /** Detects whether this heap is empty.
     @return  True if the heap is empty, or false otherwise. */
  public boolean isEmpty();

  /** Gets the size of this heap.
     @return  The number of entries currently in the heap. */
  public int getSize();

  /** Removes all entries from this heap. */
  public void clear();
} // end MaxHeapInterface
```

More on General Trees

- Parse tree
  - Check syntax of a string for valid algebraic expression
  - If valid can be expressed as a parse tree
- Parse tree must be a general tree
  - So it can accommodate any expression
- Compilers use parse trees
  - Check syntax, produce code

Parse Tree for an Equation

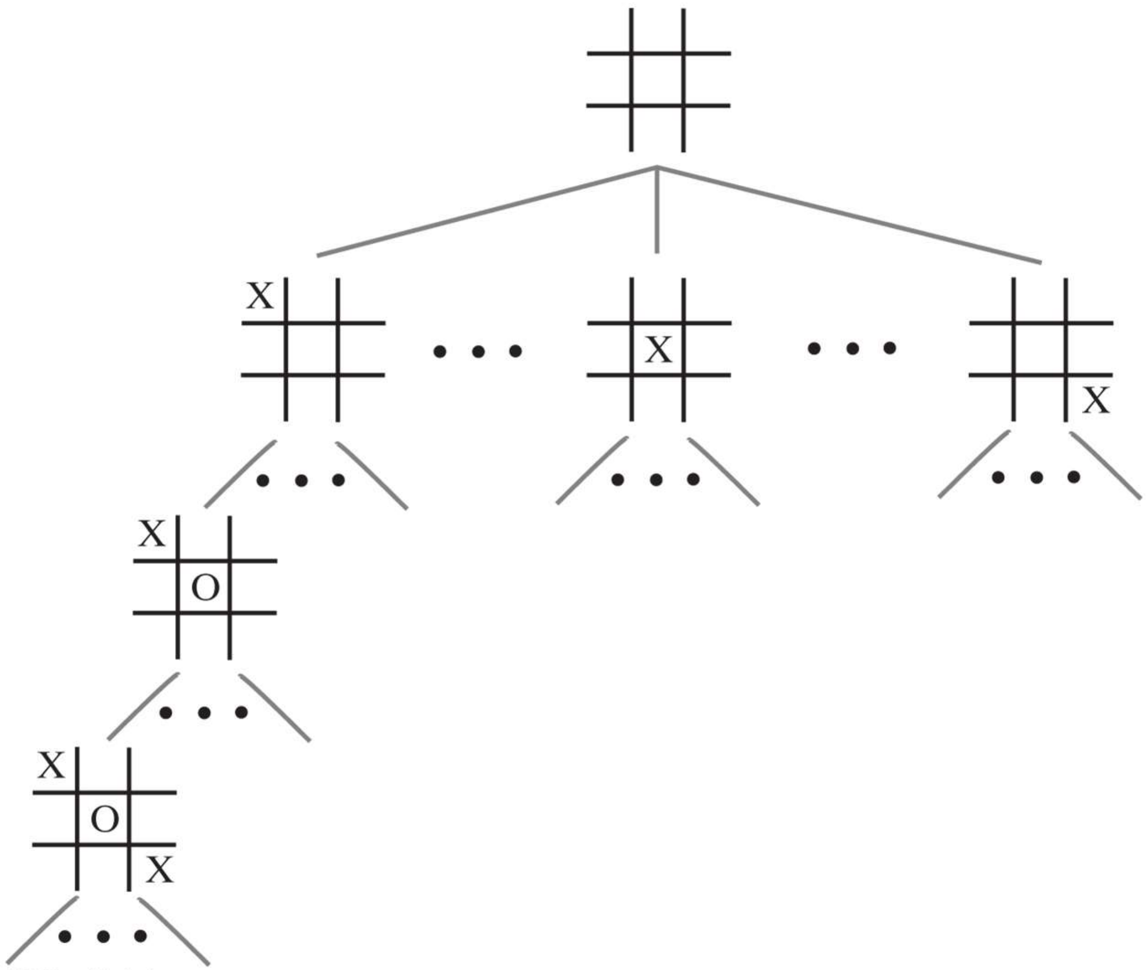
- A parse tree for the algebraic expression `a * (b + c)`



© 2019 Pearson Education, Inc.

Parse Tree for a Game

- A portion of a game tree for tic-tac-toe



© 2019 Pearson Education, Inc.