10.5 Show the array that is obtained by using the natural mapping to store the binary tree shown in Example 10.2 on page 201.

10.6 Show the array that is obtained by using the natural mapping to store the binary tree shown in Problem 10.3 above.

10.7 If the nodes of a binary tree are numbered according to their natural mapping, and the visit operation prints the node's number, which traversal algorithm will print the numbers in order?

10.8 Write the expression tree for the expression $a \times (b + c) \times (d + e + f)$.

10.9 Write the prefix and the postfix representations for the expressions in Problem 10.8.

10.10 What are the bounds on the number $n$ of nodes in a binary tree of height 4?

10.11 What are the bounds on the height $h$ of a binary tree with 7 nodes?

10.12 What form does the highest binary tree have for a given number of nodes?

10.13 What form does the lowest binary tree (i.e., the least height) have for a given number of nodes?

10.14 Verify the recursive definition of binary trees (page 201) for the binary tree shown at the right.

10.15 Draw all 42 binary trees of size $n = 5$.

10.16 How many different binary trees of size $n = 6$ are there?

10.17 Derive a recurrence relation for the number $f(n)$ of binary trees of size $n$.

10.18 Show that, for all $n \leq 8$, the function $f(n)$ derived in Problem 10.17 produces the same sequence as the following explicit formula

$$f(n) = \frac{\binom{2n}{n}}{n+1} = \frac{(2n)!}{n!(n+1)!} \frac{(2n)(2n-1)(2n-2)\cdots(2n+3)(2n+2)}{(n)(n-1)(n-2)(n-3)\cdots(2)(1)}$$

For example,

$$f(4) = \frac{\binom{8}{4}}{5} = \frac{8!}{4!5!} = \frac{(8)(7)(6)}{(4)(3)(2)(1)} = \frac{(8)(7)}{4} = 14$$

10.19 Prove Corollary 10.3 on page 203.

10.20 Prove Theorem 10.2 on page 204.

10.21 Prove that every subtree of a complete binary tree is complete.

10.22 Prove that every subtree of a complete binary tree with the heap property is another complete binary tree with the heap property.

10.23 Draw the forest that is represented by the binary tree shown on the right.

10.24 Derive an explicit formula for the number $f(h)$ of complete binary trees of height $h$.

10.25 Derive an explicit formula for the number $f(h)$ of full binary trees of height $h$.

10.26 Implement the following member function for the `BinaryTree` class:

```cpp
bool empty() const;
// returns true iff this tree is empty
```

10.27 Implement the following member function for the `BinaryTree` class:

```cpp
int size() const;
// returns the number of elements in this tree
```
10.28 Implement the following member function for the BinaryTree class:
   ```cpp
ing int leaves() const;
// returns the number of leaves in this tree
```
10.29 Implement the following member function for the BinaryTree class:
   ```cpp
ing int height() const;
// returns the height of this tree
```
10.30 Implement the following member function for the BinaryTree class:
   ```cpp
int level(Iterator it);
// returns the level of the element *it in this tree
```
10.31 Implement the following member function for the BinaryTree class:
   ```cpp
void reflect();
// swaps the two children of each node in this tree
```
10.32 Implement the following member function for the BinaryTree class:
   ```cpp
void defoliate();
// removes all the leaves from this tree
```
10.33 Implement the following member function for the BinaryTree class:
   ```cpp
Type& root() const;
// provides read-write access to this tree’s root element
```
10.34 Implement the following static member function for the BinaryTree class:
   ```cpp
static bool isRoot(Iterator it);
// true iff *it has no parents
```
10.35 Implement the following static member function for the BinaryTree class:
   ```cpp
static bool isLeaf(Iterator it);
// true iff *it has no children
```
10.36 Implement the following static member function for the BinaryTree class:
   ```cpp
static Iterator parent(Iterator it);
// returns location of parent of *it
```
10.37 Implement the following static member function for the BinaryTree class:
   ```cpp
static Iterator leftChild(Iterator it);
// returns location of left child of *it
```
10.38 Implement the following static member function for the BinaryTree class:
   ```cpp
static Iterator rightChild(Iterator it);
// returns location of right child of *it
```
10.39 Implement the following member function for the BinaryTree class:
   ```cpp
Iterator find(Iterator begin, Iterator end, const Type& x);
// returns location of x in [begin,end[
```

**Answers to Review Questions**

10.1 The full binary tree of height 3 has \( l = 2^3 = 8 \) leaves.
10.2 The full binary tree of height 3 has \( m = 2^3 - 1 = 7 \) internal nodes.
10.3 The full binary tree of height 3 has \( n = 2^{3+1} - 1 = 2^4 - 1 = 16 - 1 = 15 \) nodes.
10.4 The full binary tree of height 9 has \( l = 2^9 = 512 \) leaves.
10.5 The full binary tree of height 9 has \( m = 2^9 - 1 = 512 - 1 = 511 \) internal nodes.
10.6 The full binary tree of height 9 has \( n = 2^{9+1} - 1 = 2^{10} - 1 = 1024 - 1 = 1023 \) nodes.
10.7 By Corollary 10.3, in any binary tree: \( \lceil \log n \rceil \leq h \leq n - 1 \). Thus in a binary tree with 100 nodes \( \lceil \log 100 \rceil \leq h \leq 100 - 1 = 99. Since \lceil \log 100 \rceil = \lceil (\log 100)/(\log 2) \rceil = \lceil 6.6 \rceil = 6 \), it follows that the height must be between 6 and 99, inclusive: \( 6 \leq h \leq 99 \).
10.8 The inorder traversal algorithm for binary trees recursively visits the root in between traversing the left and right subtrees. This presumes the existence of exactly two (possibly empty) subtrees at every (nonempty) node. In general trees, a node may have any number of subtrees, so there is no simple algorithmic way to generalize the inorder traversal.