Binary Tree Properties & Representation

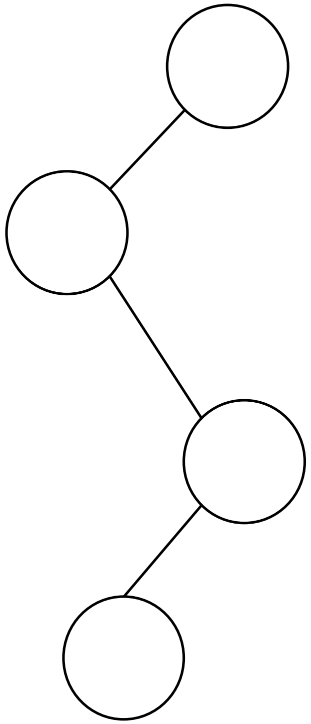
Binary trees with minimum number of nodes

Binary trees with minimum number of nodes

Given the height of a binary tree.

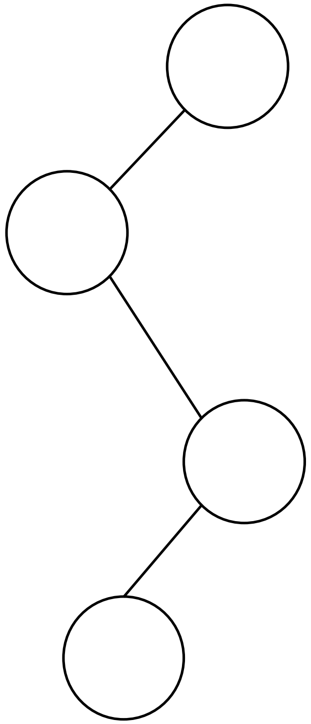
Binary trees with minimum number of nodes

Given the height of a binary tree.



Binary trees with minimum number of nodes

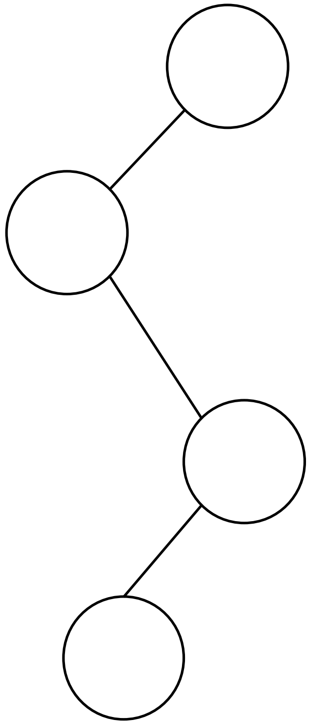
Given the height of a binary tree.



Binary trees with minimum number of nodes

Given the height of a binary tree.

h = 4: tree height
n = 4: number of nodes

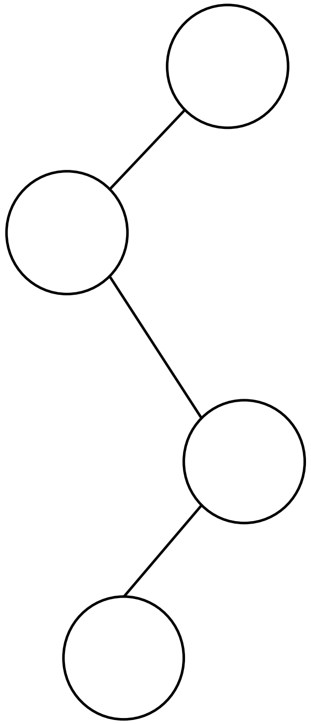


Binary trees with minimum number of nodes

- Each level has one node.
- The minimum number of nodes in binary trees is equal to the tree height (starting from 1).

$h = 4$: tree height

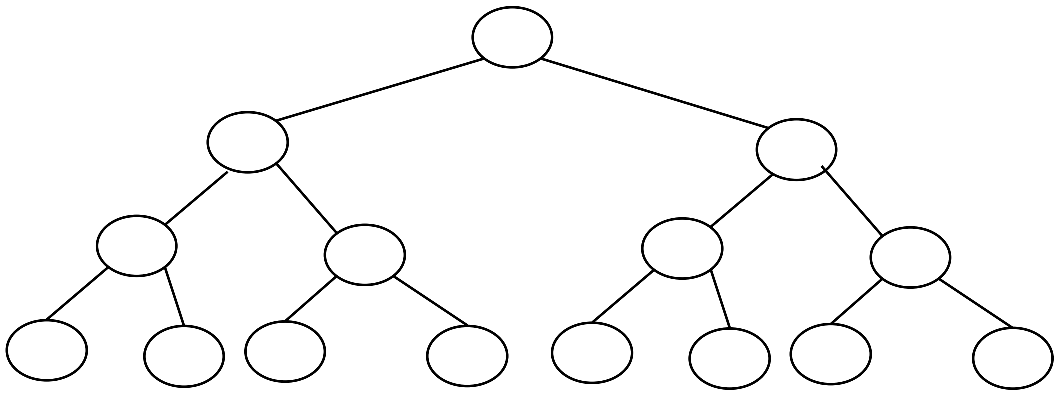
$n = 4$: number of nodes



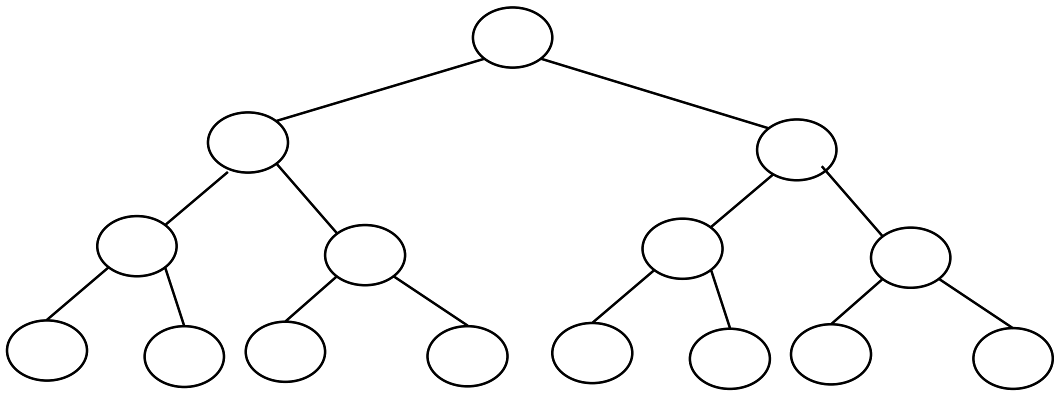
The binary tree with the maximum number of nodes

Given the height of a binary tree.

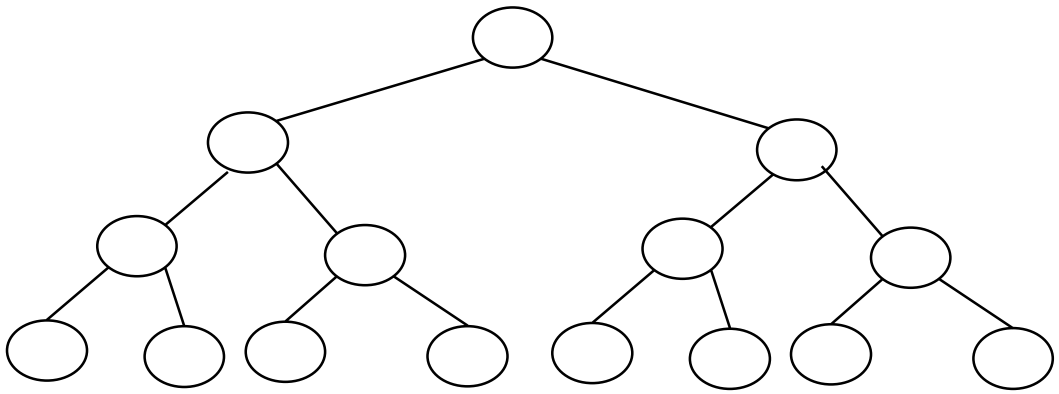
The binary tree with the maximum number of nodes



The binary tree with the maximum number of nodes



The binary tree with the maximum number of nodes

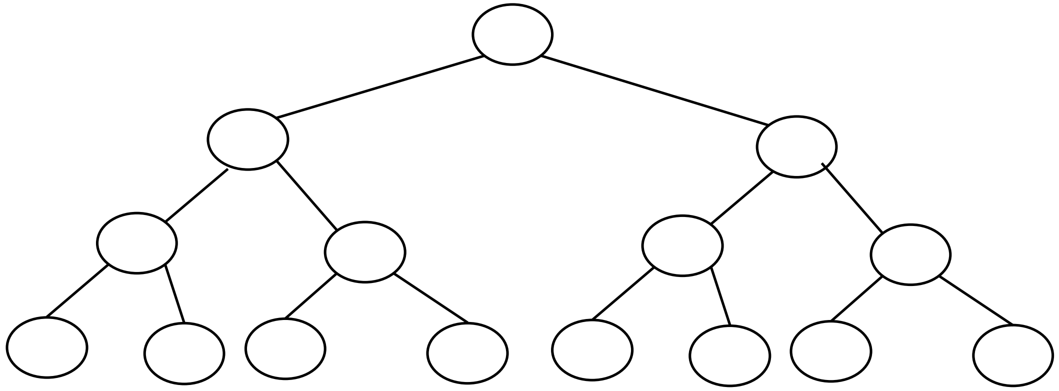


$$n = 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$$

$$= ?$$

The binary tree with the maximum number of nodes

- Nodes at each level are filled.
- Each internal node has two children.

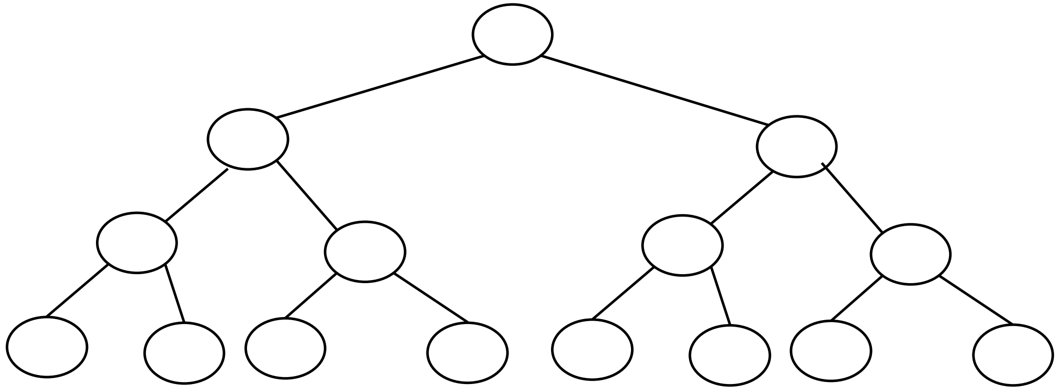


$$n = 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$$

$$= ?$$

The binary tree with the maximum number of nodes

- Nodes at each level are filled.
- Each internal node has two children.

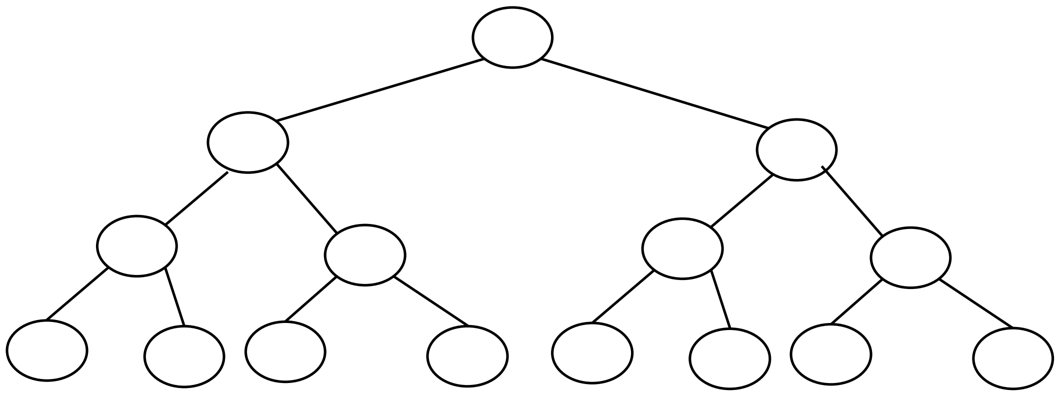


$$n = 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$$

$$= 2^h - 1$$

Perfect binary tree

- Nodes at each level are filled.
- Each internal node has two children.



height $h = 4$

n
$= 2^4 - 1$
$= 15$

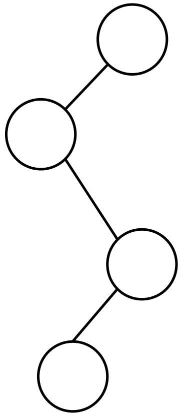
$$n = 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$$

$$= 2^h - 1$$

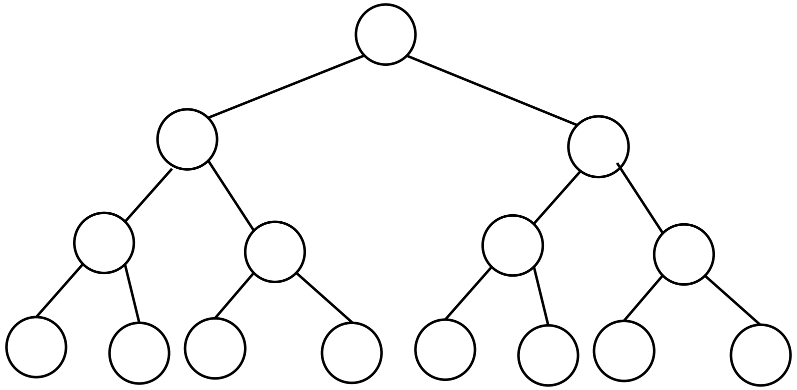
Relationship between number Of nodes and height

We have

- $h <= n <= 2^h - 1$
- $n+1 <= 2^h \;\; => \log_2(n+1) <= h$
- $\log_2(n+1) <= h <= n$



h = 4: tree height
n = 4: number of nodes

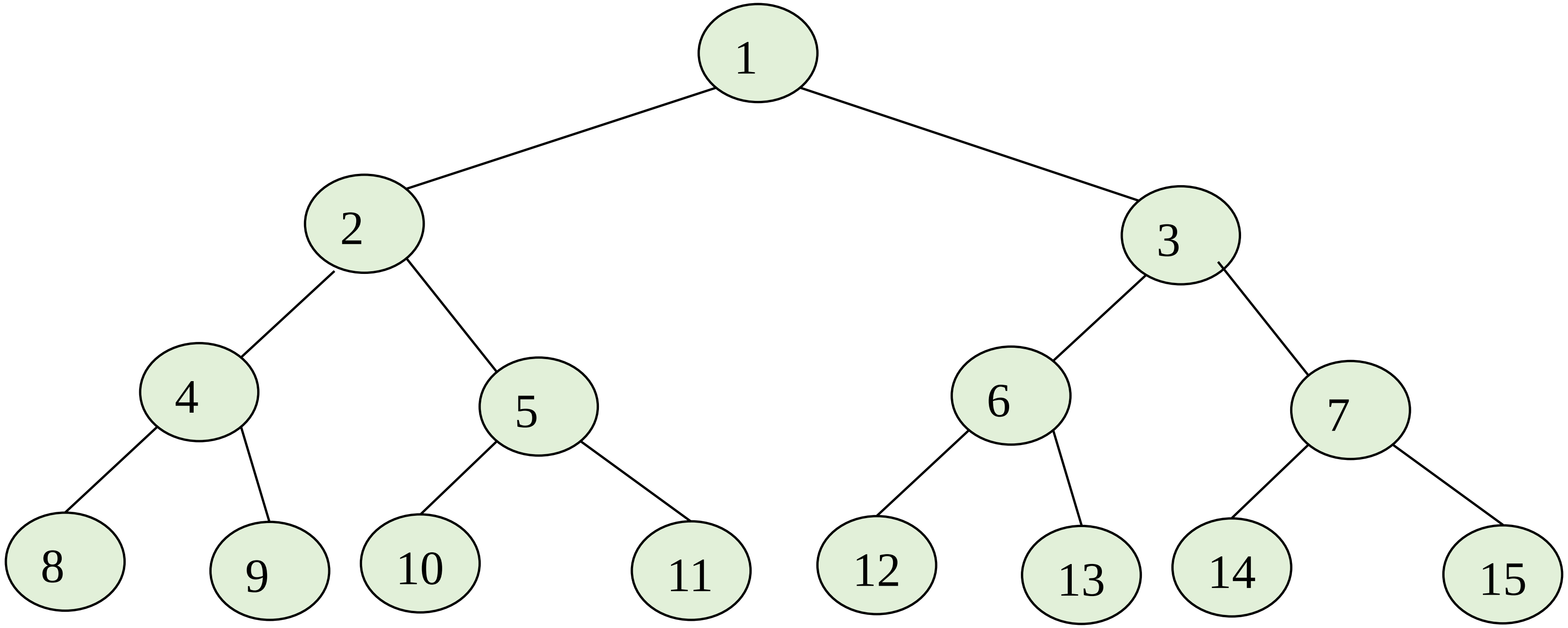


$n = 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

$= 2^h - 1$

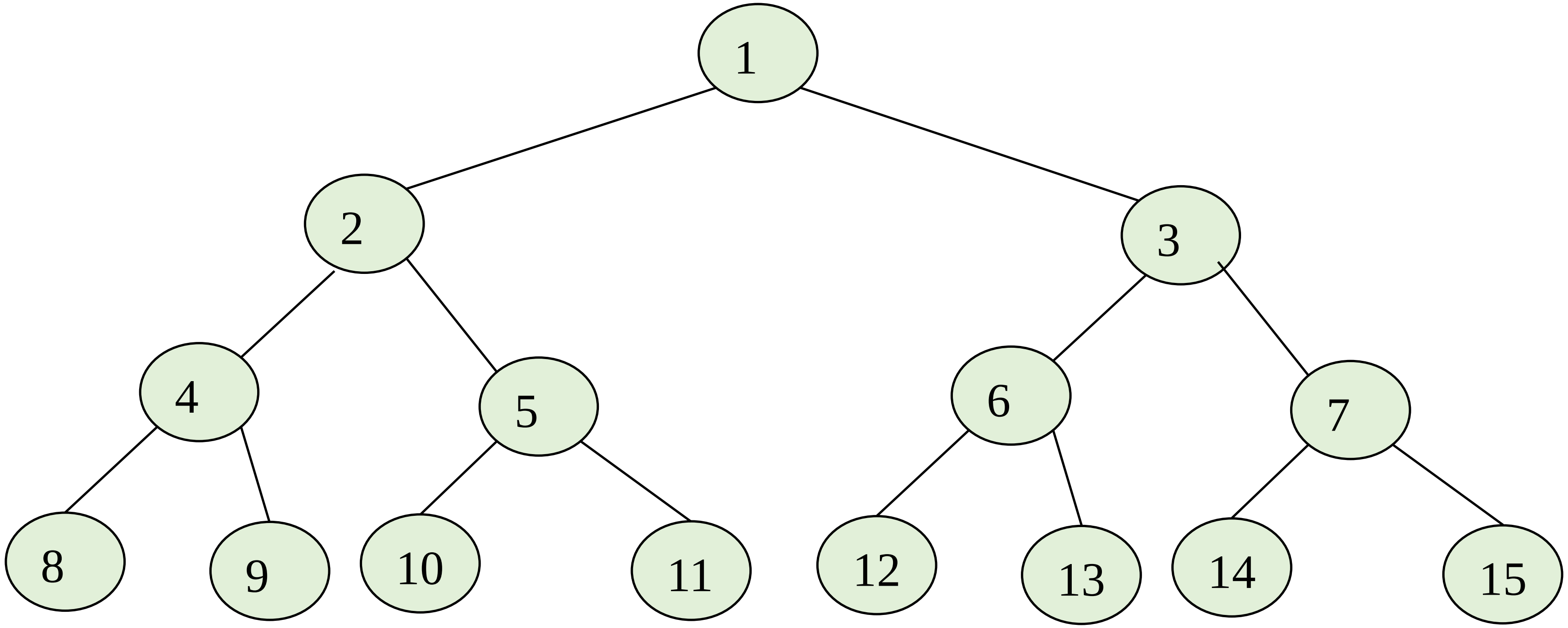
Perfect binary tree

- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.



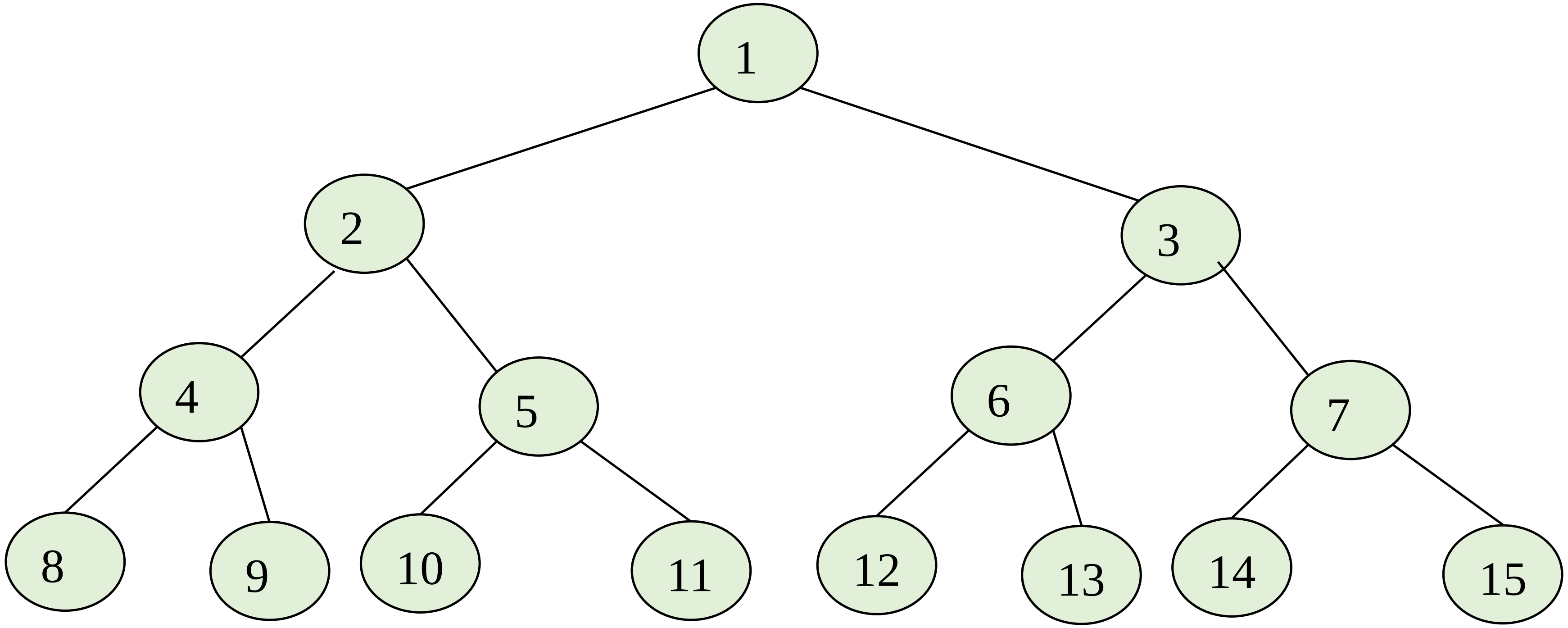
Perfect binary tree

- The root node has no parent.
- Parent of node i is node i/2   (integer division)
- e.g., 9/2 = 4; 8/2 = 4



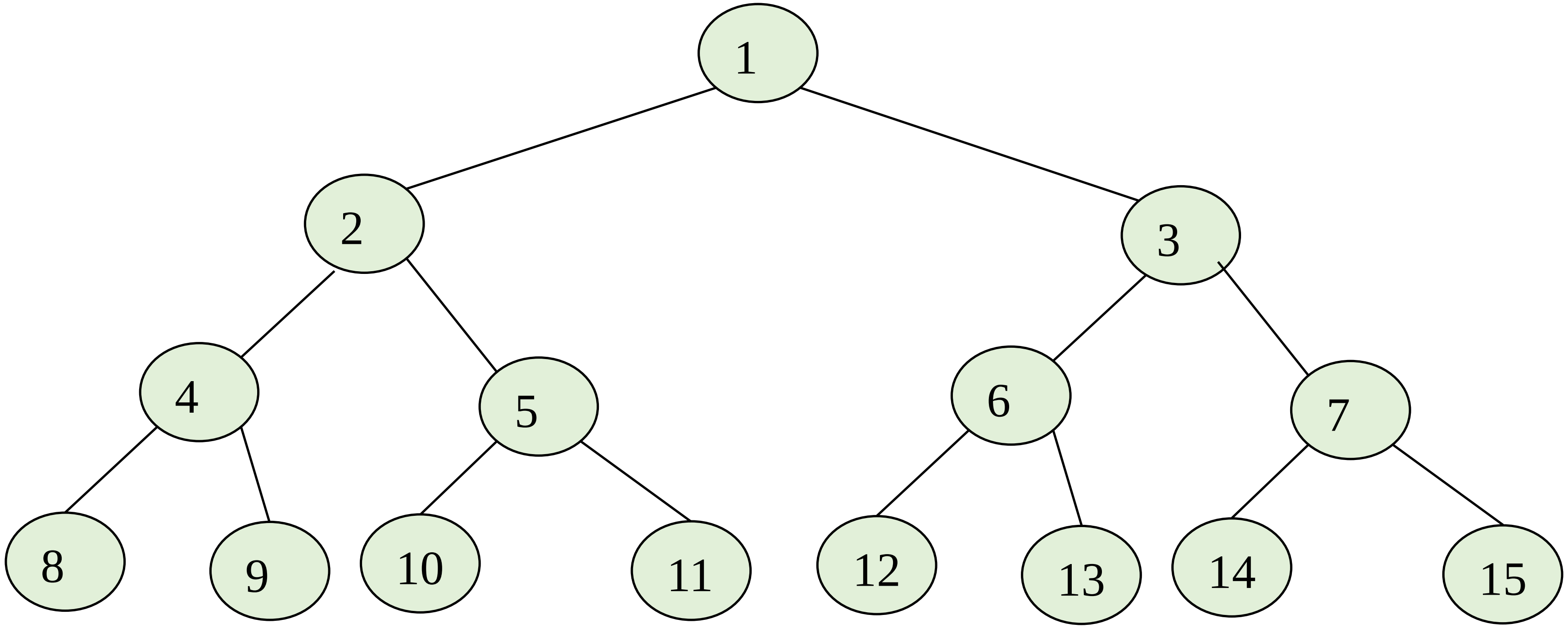
Perfect binary tree

- The root node has no parent.
- Parent of node i is node i/2   (integer division)
- e.g., 9/2 = 4; 8/2 = 4



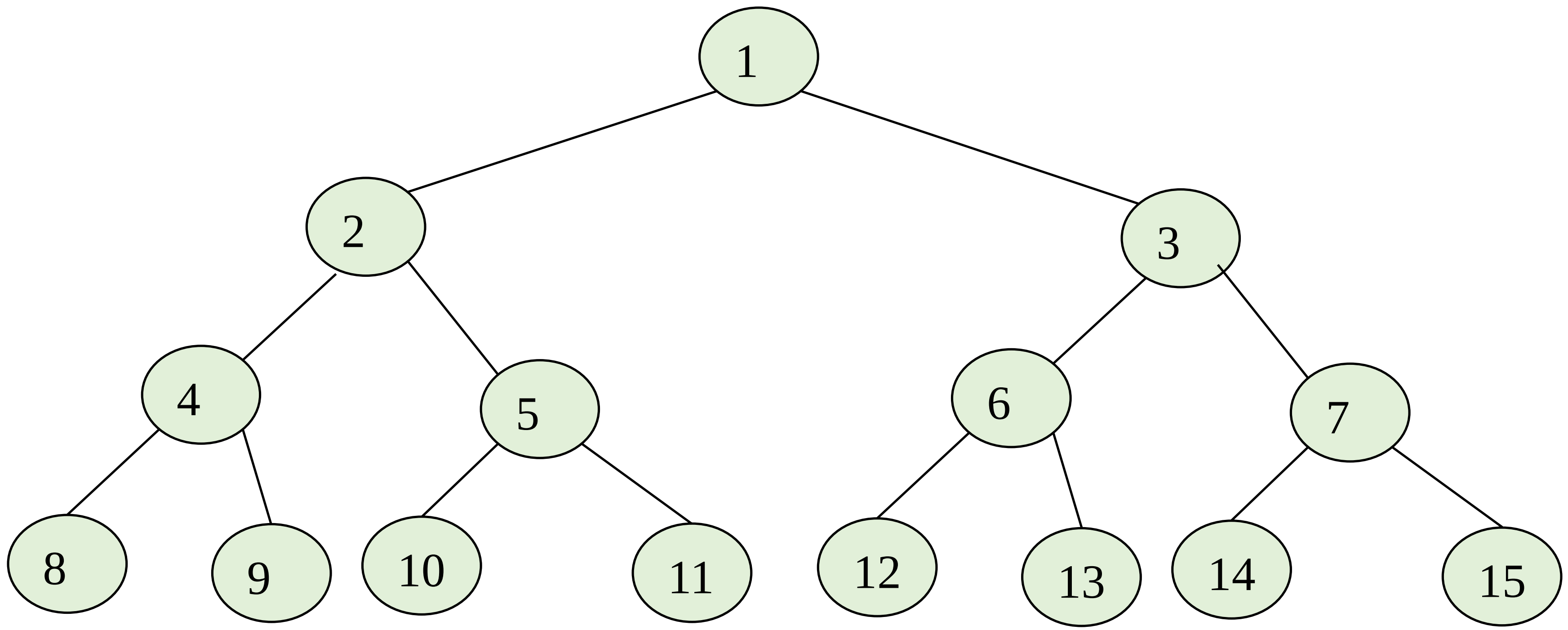
Perfect binary tree

- Left child of node i is node 2i
- If 2i > n, node i has no left child



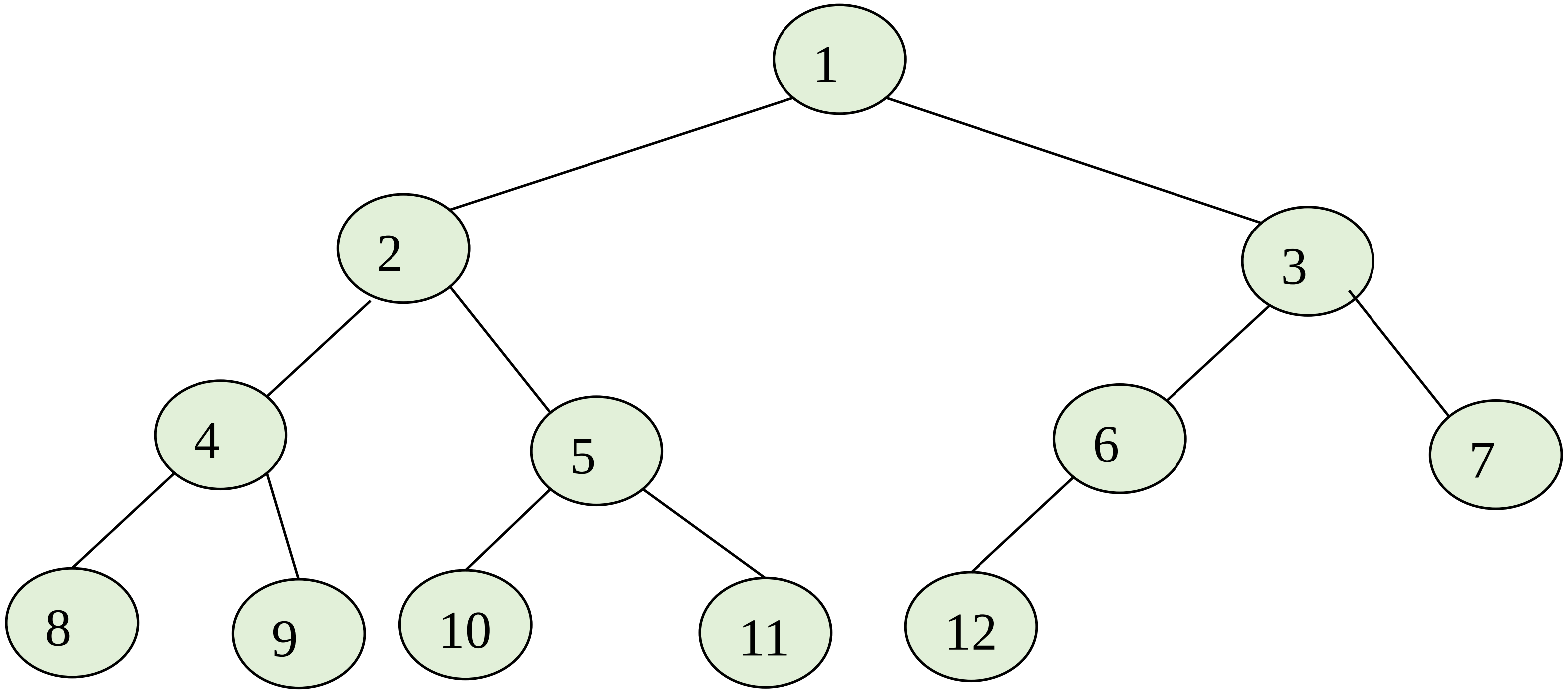
Perfect binary tree

- Right child of node i is node 2i+1
- If 2i + 1> n, node i has no right child



Complete binary tree with n nodes

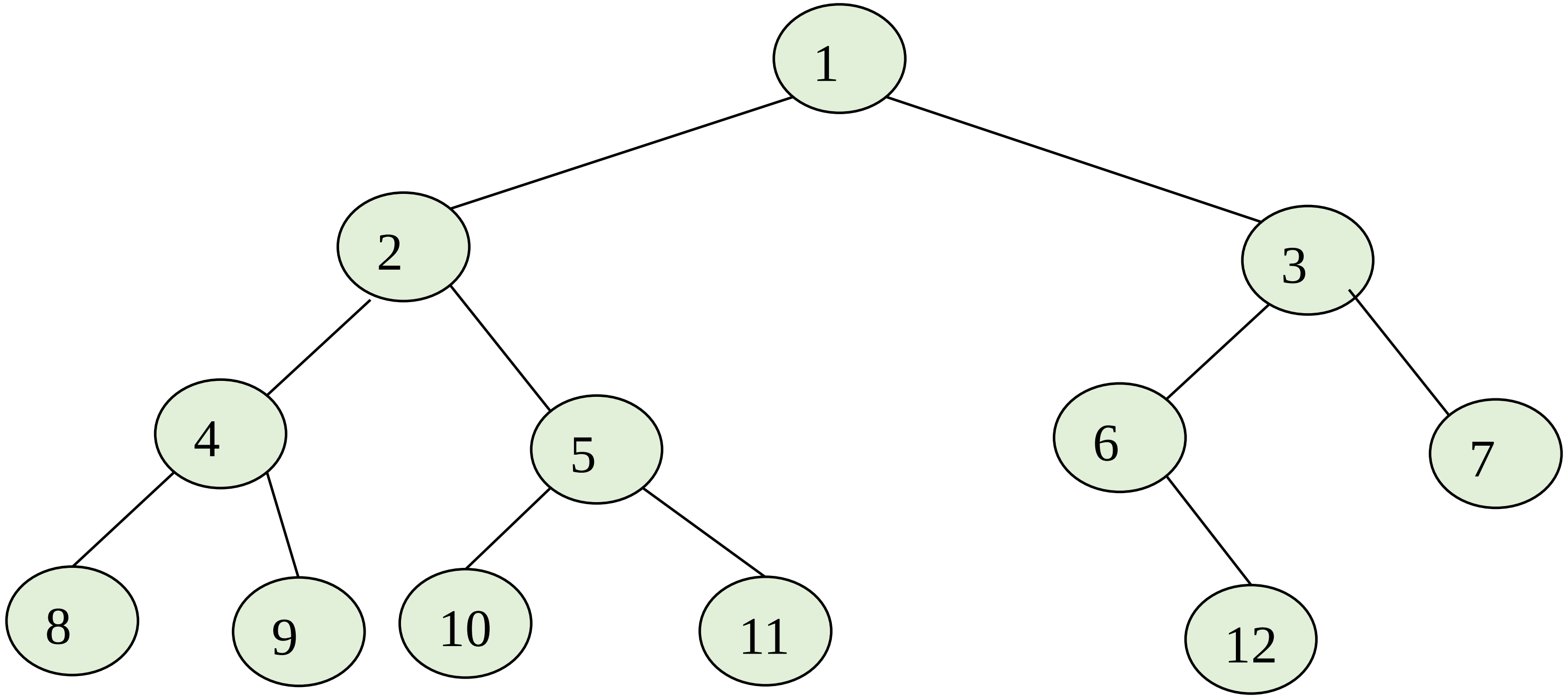
- Start with a perfect binary tree that has at least n nodes.
- Number the nodes from left to right at each level and top to bottom.
- Except for the last level, the leaf nodes must be filled from left to right.



A complete binary tree with 12 nodes.

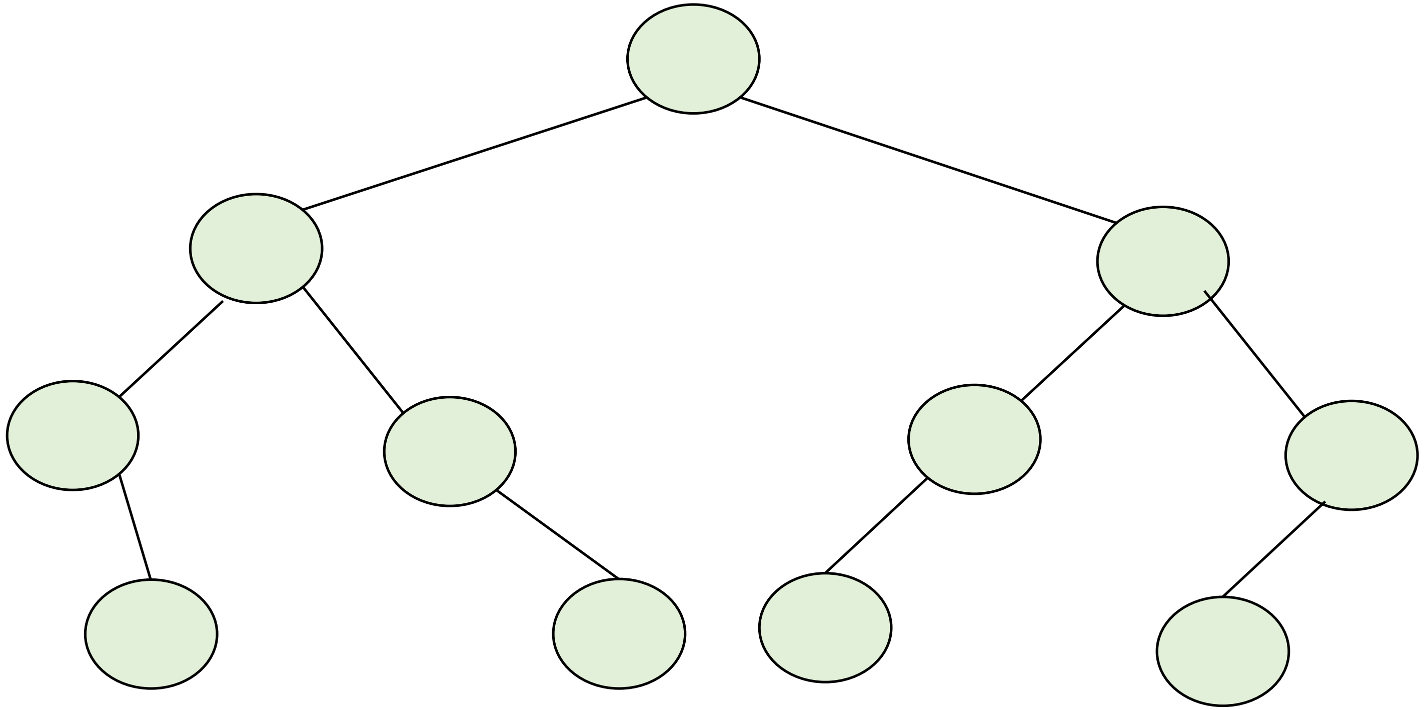
Example

- This is not a complete binary tree.



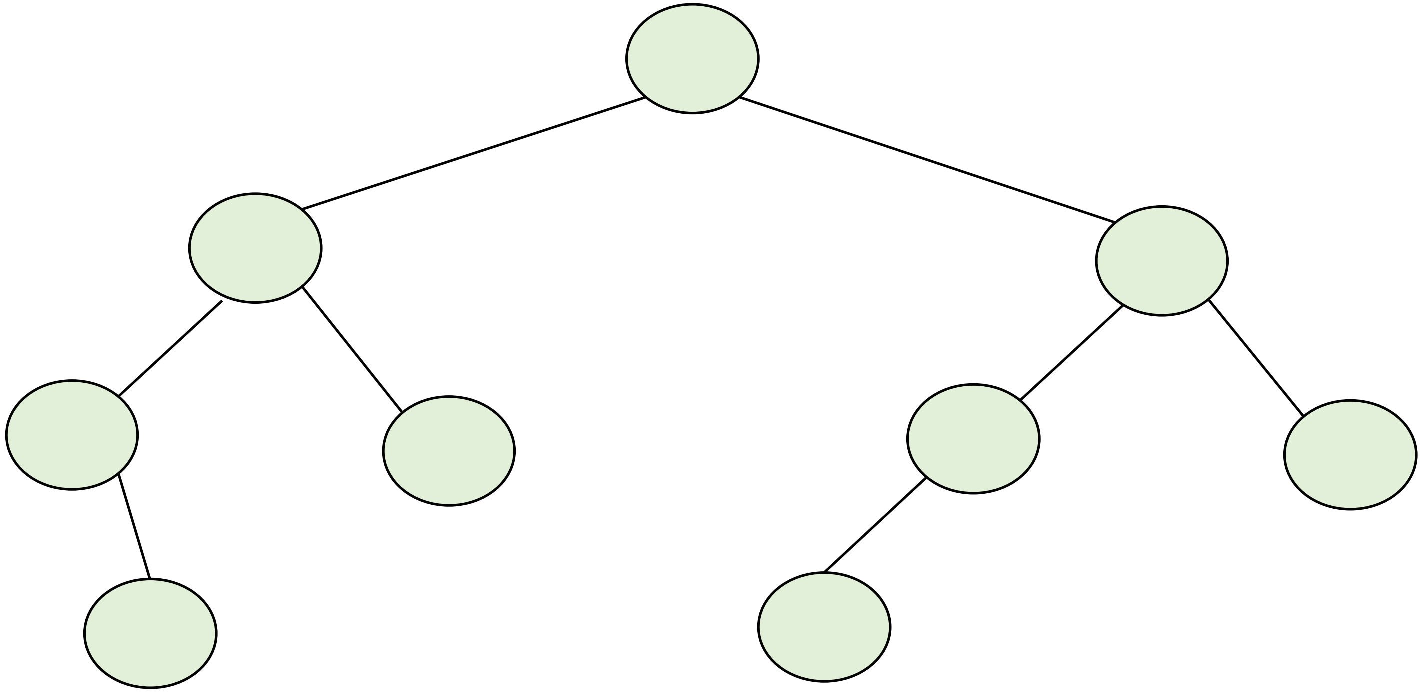
Perfectly height-balanced trees

- Perfectly height-balanced: if the left and right subtrees of any node are the same height.



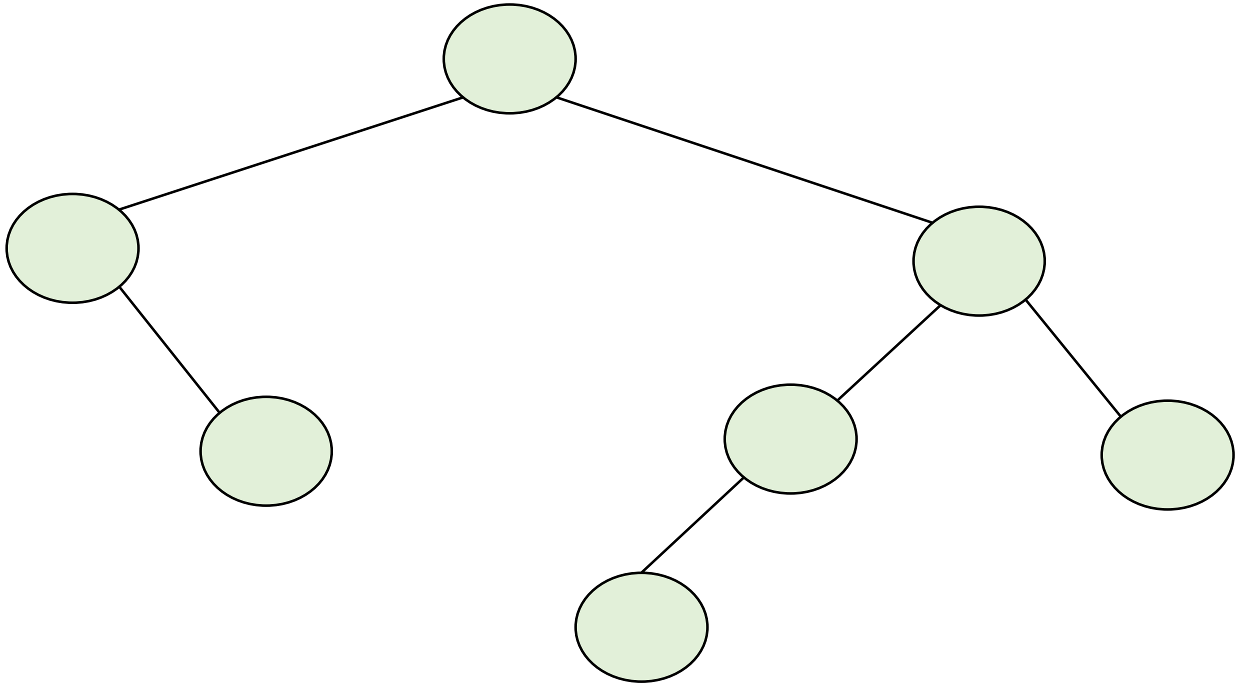
Height-balanced trees (1)

- Height-balanced: if the heights of the left and right subtrees of each node are within *one*.



Height-balanced trees (2)

- Height-balanced: if the heights of the left and right subtrees of each node are within *one*.

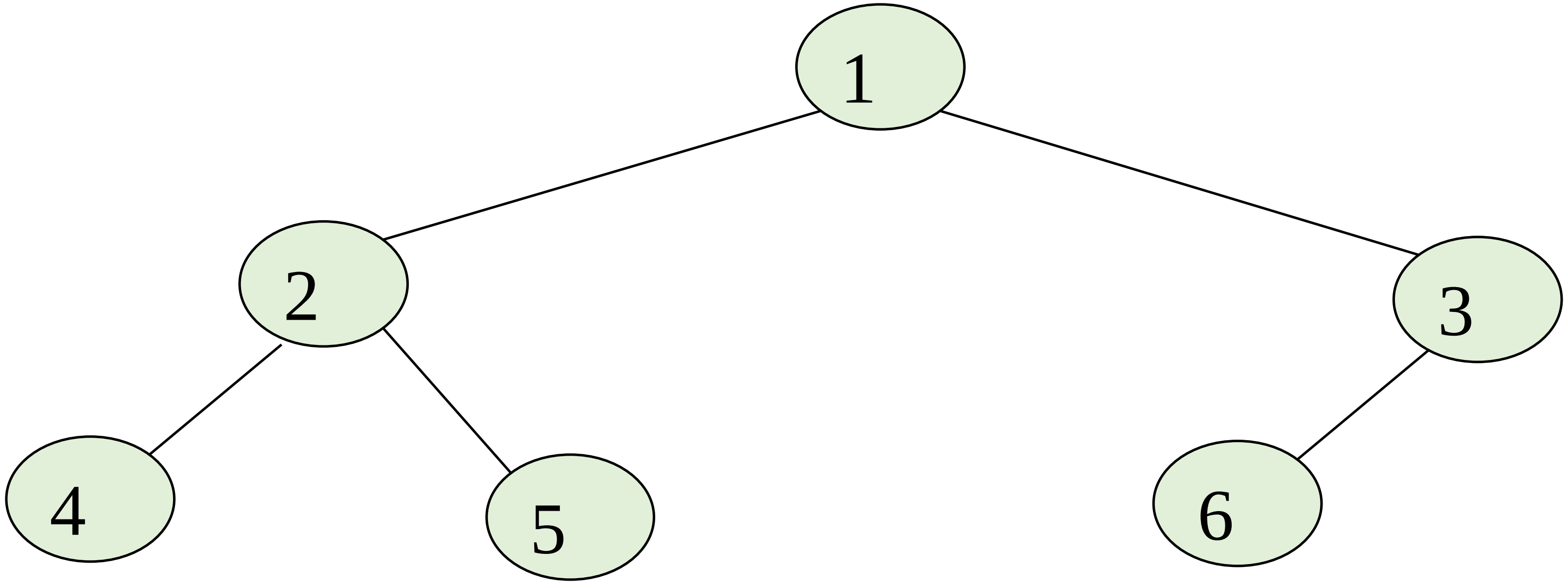


Binary Tree Representation

- Array representation.
- Linked representation.

Array Representation

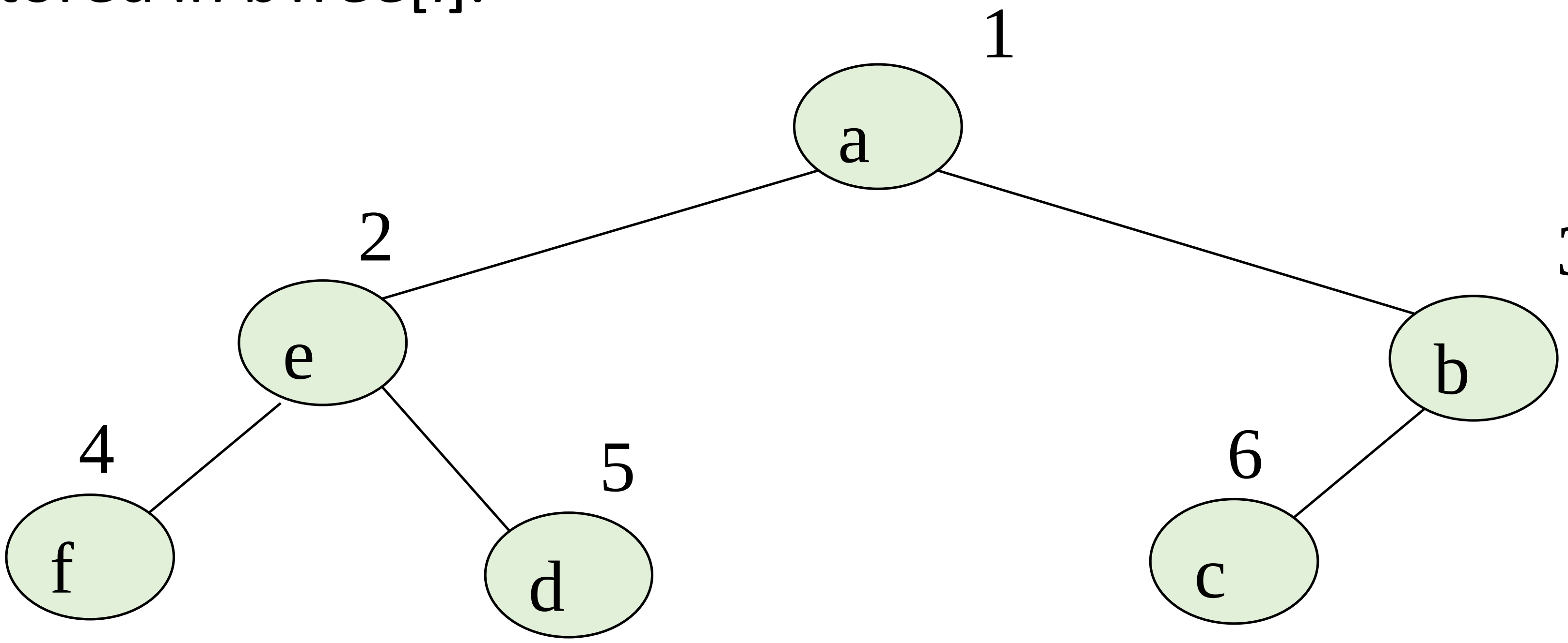
- Number the nodes using the numbering scheme for a perfect binary tree.
- Node i is stored in bTree[i].



bTree[] =

Array Representation

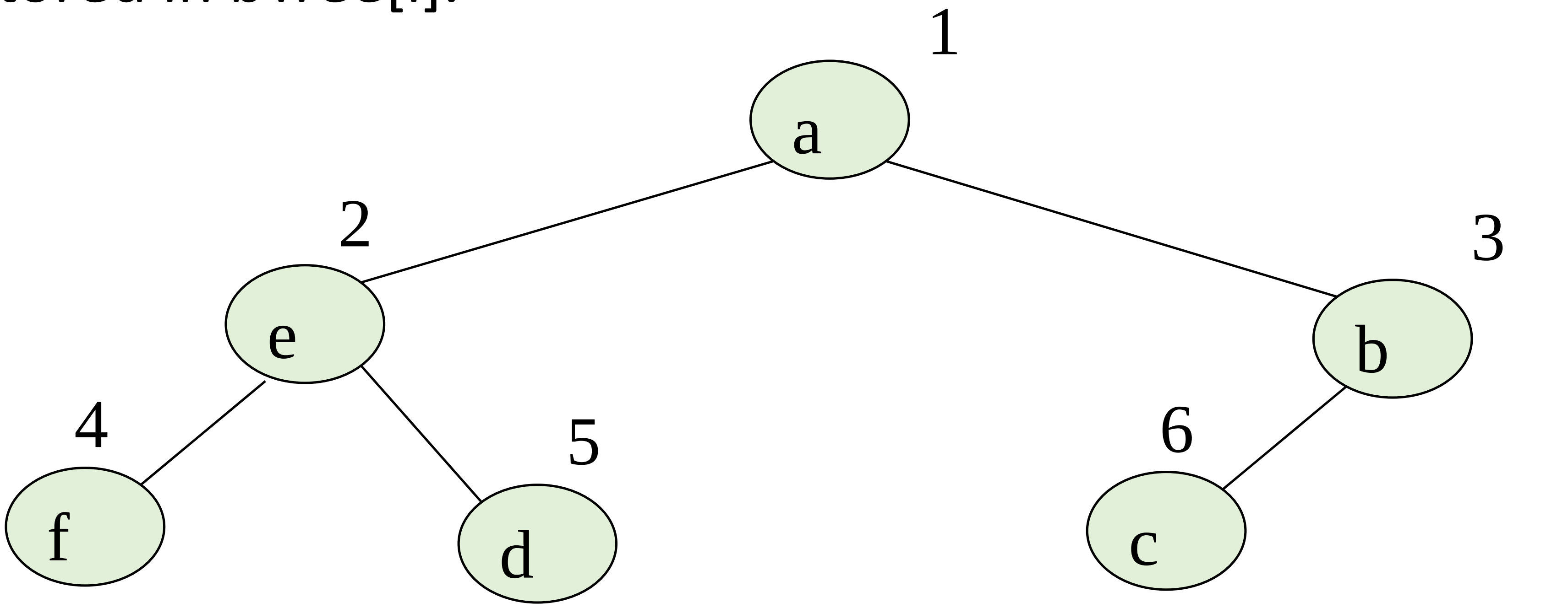
- Number the nodes using the numbering scheme for a perfect binary tree.
- Node i is stored in bTree[i].



bTree[] =

Array Representation

- Number the nodes using the numbering scheme for a perfect binary tree.
- Node i is stored in bTree[i].

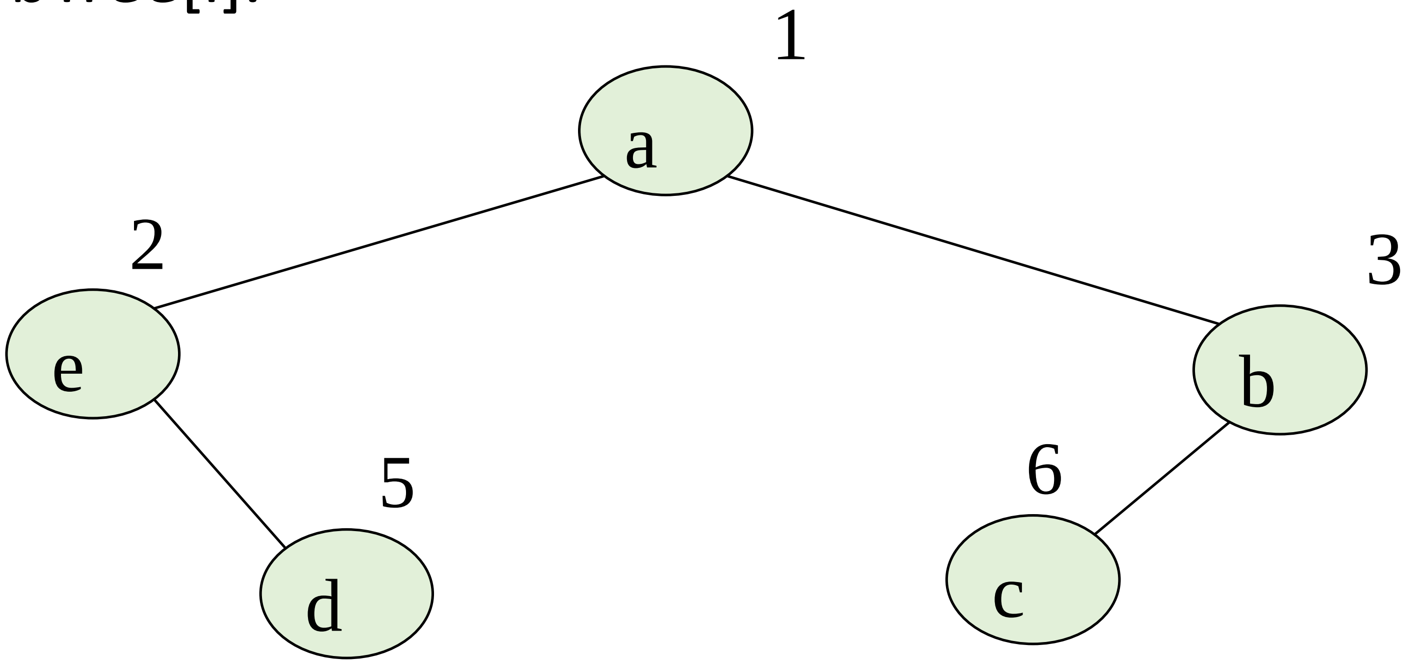


	0	1	2	3	4	5	6	7
bTree[] =	-	a	e	b	f	d	c	-

-: a special symbol indicates that the node does not exist.

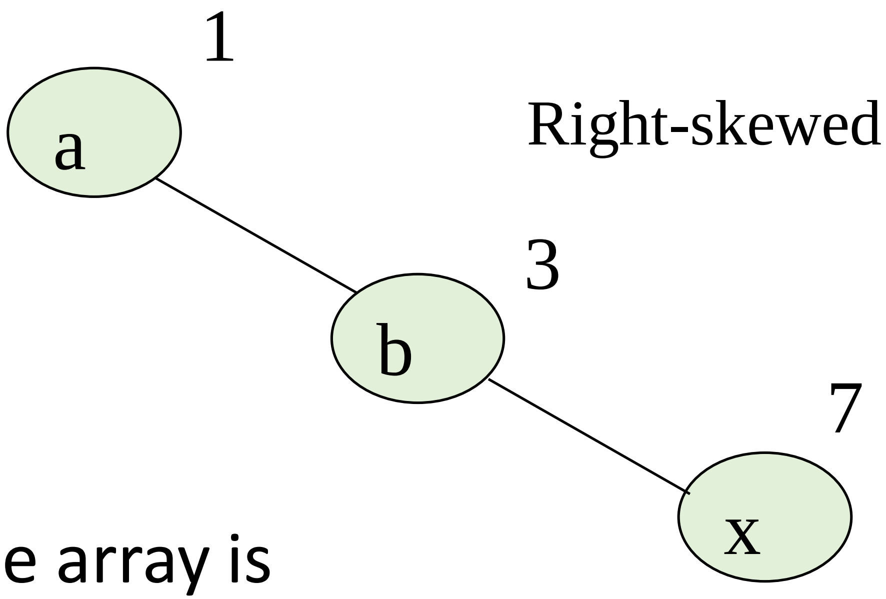
Array Representation

- Number the nodes using the numbering scheme for a perfect binary tree.
- Node i is stored in bTree[i].



	0	1	2	3	4	5	6	7
bTree[] =	-	a	e	b	-	d	c	-

The tree array size (1)



Right-skewed binary tree

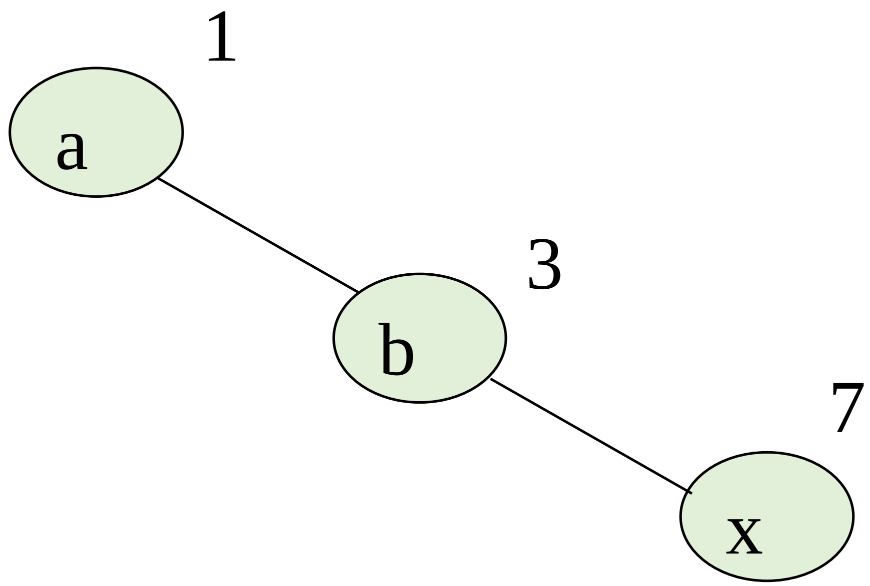
➤ $\log_2(n+1) <= h <= n$

- Given a tree height h, the size of the tree array is 2^h (including the 0-node)

$$n = 1 + 2 + 4 + 8 + \dots + 2^{h-1}$$
$$= 2^h - 1$$

	0	1	2	3	4	5	6	7
bTree[] =	-	a	-	b	-	-	-	x

The tree array size (2)



- $\log_2(n+1) <= h <= n$

- Given n, the size of the tree array is 2^n (including the 0-node).
- An array representation may take an exponential amount of space.

	0	1	2	3	4	5	6	7
bTree[] =	-	a	-	b	-	-	-	x

Linked List Representation

- Each binary tree node is represented as an object of the Node class.
- The space required by an n-node binary tree is linear to n = n * (space required by one node).
- This presentation takes space that is linear in the number of elements in the tree.

The Node class

```
<class T> Node
{
  T data;
  Node<T> *leftChild, *rightChild;
  Node()
      {leftChild = rightChild =     ;}
   ......
};
```

The BinaryTree class

```
<class T> BinaryTree {
    T data;
    Node<T> *root;
    BinaryTree operator=( const BinaryTree &);
    int getHeight( ) const;
    int getNumOfNodes( ) const;
    Node<T>* addData( const T &data );
    vector<Node<T>> traverse_PostOrder( ) const;
    vector<Node<T>> traverse_InOrder( ) const;
    vector<Node<T>> traverse_PreOrder( ) const;
    void printf( ) const;
};
```

The BinaryTree class

```
<class T> BinaryTree {
    T data;
   Node<T> *root;
   BinaryTree operator=( const BinaryTree &);
    int getHeight( ) const;
    int getNumOfNodes( ) const;
    Node<T>* addData( const T &data );
    vector<Node<T>*> traverse_PostOrder( ) const;
    vector<Node<T>*> traverse_InOrder( ) const;
    vector<Node<T>*> traverse_PreOrder( ) const;
    void printf( ) const;
};
```

Return pointers of nodes. Save memory space.

The BinaryTree class

```
<class T> BinaryTree {
    T data;
   Node<T> *root;
   BinaryTree operator=( const BinaryTree &);
    int getHeight( ) const;
    int getNumOfNodes( ) const;
    Node<T>* addData( const T &data );
    vector<const Node<T>*> traverse_PostOrder( ) const;
    vector<const Node<T>*> traverse_InOrder( ) const;
    vector<const Node<T>*> traverse_PreOrder( ) const;
    void printf( ) const;
};
```

Return pointers of nodes. Save memory space.

Exercises

- Implement a recursive function to count the number of nodes of a binary tree.

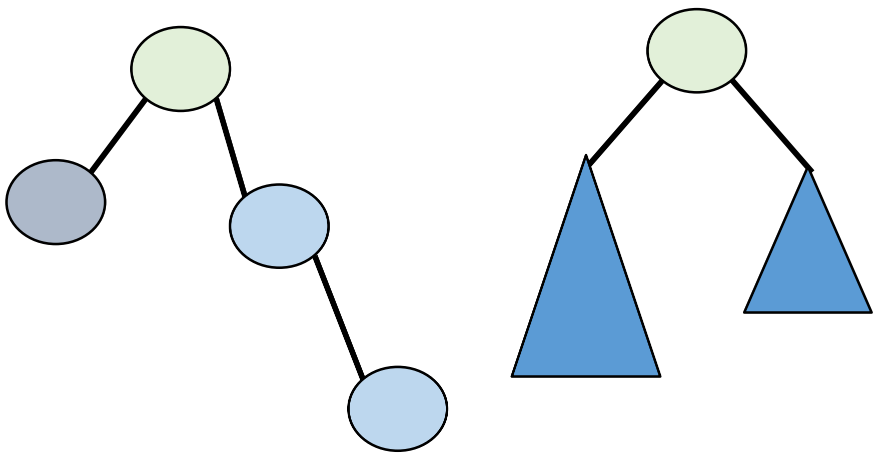
```
void count( const BinaryTree *node, int &c)
{
   if (!node) return;
   ++c;
   count( node->leftChild, c);
   count( node->rightChild, c);
}
int count( ) const {
   count = 0;
   return count( root, count );
}
```

Exercises

- Implement a recursive function to determine the height of a binary tree.

```
int getHeight( const BinaryTree *node, int h) {
    if (!node) return;
    int hL = getHeight( node->leftChild, h+1);
    int hR = getHeight( node->rightChild, h+1);
    return max( hL, hR ) + 1;
}

int getHeight( ) const {
    int h = 0;
    getHeight( root, h );
    return h;
}
```



Exercises

- Implement a recursive function to determine the number of leaves of a binary tree.

```
int getNumOfLeaves( const BinaryTree *node) {
   if ( !node ) return 0;
   if ( !node->leftChild & !node->rightChild ) return 1;
   int nL = getNumOfLeaves( node->leftChild );
   int nR = getNumOfLeaves( node->rightChild );
   return nL + nR;
}

int getNumOfLeaves( ) const {
   int n = 0;
   if ( !root ) return 0;
   return getNumOfLeaves( root );
}
```