**1.** What is the minimum possible depth of a binary tree with 300 leaves?

**Solution** 9.

**2.** What is the minimum possible depth of a ternary tree with 300 leaves?

**Solution** 6.

**3.** There are n nodes in a given regular binary tree. What is the number of its leaves?

**Solution** (n+1)/2.

**4.** A given binary tree T has exactly three leaves. Therefore

1. T has at most two inner nodes,
2. number of inner nodes is not limited,
3. all leaves are in the same depth,
4. all leaves cannot have the same depth,
5. T is regular.

**Solution** b).

**5.** Algorithm A traverses a balanced binary tree with n nodes. In each node the algorithm performs an additional procedure which complexity is Θ(*n*2). What is the asymptotic complexity of A?

**Solution** Θ(*n*3).

**6**. Algorithm A traverses a binary tree with depth n. The total number of operations performed in depth k is equal to *k*+*n*. Each operation has a constant asymptotic complexity. What is the asymptotic complexity of A?

**Solution** Θ(n2).



**7.**  Algorithm A traverses the given tree and in each node it prints the character stored in that node. Write the output of the algorithm when the traversal is

 A) Inorder, B) preoreder, C) postorder.

**Solution**

a) MHROKLBAEJVD

b) BOHMRLKJAEDV

c) MRHKLOEAVDJB

**8.**  Describe the shape of a binary tree when

 a) In- Pre- and Postorder traversal produces the same sequence of nodes,

 b) In- and Postorder traversal produces the same sequence of nodes,

 c) Pre- and Postorder traversal produces the same sequence of nodes,

 d) In- and Preorder traversal produces the same sequence of nodes.

**Solution**

a) Each internal node has only right child.

b) Each internal node has only left child.

c) & d) The tree contains at most 1 node.

**9.** A method first prints the keys of the nodes of a binary tree using the Inorder traversal an then it prints the keys of the nodes using the Preorder traversal. The result is

Inorder: 45 71 98 47 50 62 87 3 79 Preorder: 50 47 71 45 98 62 3 87 79

 a) Reconstruct the tree completely.

 b) Write an algorithm which will reconstruct any binary tree when it is given a sequence S1 of keys of the nodes produced by the Inorder traversal and a sequence S2 of keys of the nodes produced by the Preorder traversal.



**Solution**

a) See the picture.

b) Suppose that array inArr resp. preArr contains sequence S1 resp. sequence S2. sequence. The parametrs inFirst and inLast specify the continguous part of the inArr (subsequence of S1) which contains all keys in a particular subtree of the original tree. The parametrs preFirst and preLast have analogous meaning in the array preArr (subsequence of S2).

**Node treeFromInPre (int[] inArr, int inFirst, int inLast,**

 **int[] preArr, int preFirst, int preLast){**

 **if (preFirst > preLast) return null;** *// empty array, no node*

 *// find root*

 **int inRoot = inFirst;**

 **while (inArr[inRoot] != preArr[preFirst]) inRoot++;**

*// create and return current root node with its both L and R children*

 **Node left = treeFromInPre(inArr, inFirst, inRoot-1,**

 **preArr, preFirst+1, preFirst+ inRoot-inFirst );**

 **Node right = treeFromInPre(inArr, inRoot+1, inLast,**

 **preArr, preFirst+inRoot-inFirst+1, preLast);**

 **return new Node(inArr[inRoot], left, right);**

**}**

The function supposes existence of a type Node with constructor (called in the last code line) which receives references to the roots of the left and right subtree of the constructed node.

The tree is then buit by the call

**Node root =**

 **treeFromInPre(inArr, 0, inArr.lenght-1, preArr, 0, preArr.lenght-1);**

**10.** Suggest a method which for a given value n will generate a tree with n nodes and which depth will belong to .

**Solution**

One possible solution is in the picture. There are *m*×*m* grid nodes, the upper left node represents the tree root. The parent of each other node (child) is a randomly chosen node which lies immediately above the child or to the left of the child. The depth of the tree is 2*m*−1. Denote the number of all nodes by *n* = *m*×*m*. The depth of the tree is then equal to 2− 1 ∈ .



**11.**  We have to traverse a regular binary tree and visit all its n nodes. We can move

along each edge only in the direction from the root towards a leaf. We can also jump from any node directly back to the root. Each move along one edge and each jump to the root takes one millisecond. Compute the best possible complexity of the traversal. The tree is

A) perfectly balanced

B) maximally disbalanced (its depth is (n−1)/2 )

**Solution**

A) (n+1)/2 \*(log2(n+1)−1)

B) (N+1)(N+3)/8 −1

**12.** Write a function which will remove (delete) all the leaves of a given tree. Suppose there is a reference to the parent in each node. Can the same task be achieved in a tree where nodes do not contain a reference to the parent? What is the complexity of your solution?

**Solution**

The solution is based on Postorder traversal, the parent reference is unnecessary.

The solution runs in time linear with respect to the size of the tree.

**boolean deleteLeaves (Node node){** *// supposes unempty tree*

*// a leaf*

 **if (node.Left == null && node.Right == null) return true;**

 *// inner node*

 **if (deleteLeaves(node.Left)) node.Left = null;**

 **if (deleteLeaves(node.Right)) node.Right = null;**

 **return false;**

**}**

**13.** An arithmetic expression containing only positive integers, brackets and symbols of operations +,−,\*,/ can be represented as a binary tree.

Example:The expression 6 + (4-3+5)\*(9-7) is represented by the tree in the picture.

Write a node representation and a function which input is the reference to the tree root and the return value is equal to the value of the expression represented by the tree.

**Solution**

The solution is based on Postorder traversal.

Let the value of each node be defined as follows:

The value of the leaf is the number written in that leaf.

The value of an internal node is defined as

Hodnotou každého listu je číslo v listu uložené, hodnotou vnitřního uzlu je

<value of left child> <operation written in the node> <value of right child>.

The value of the root represents the value of the expression. Next image contains the values depicted at ech node.



In the implentation we will use the following definitions:

-- **type** - either leaf or internal node,

-- **op** - operation (may be a character)

-- **val** - value

**int eval(Node n) {**

 **if (n.type == LEAF) return n.val;**

 **switch (n.type) {**

 **case ‘+’: return (eval(n.left) + eval(n.right)); break;**

 **case ‘-’: return (eval(n.left) - eval(n.right)); break;**

 **case ‘\*’: return (eval(n.left) \* eval(n.right)); break;**

 **case ‘/’: return (eval(n.left) / eval(n.right)); break;**

 **}**

**}**

**14.** The height of a node X is defined as the number of edges on a path from X to the most distant leaf in the subtree which root is X. Write the function which will assign each node in the tree the value of its height.

**Solution**

**void setHeight(Node x) {**

 int heightL = -1; // if x has no L child

 **int heightR = -1;** // dtto

 **if (x.left != null) {**

 **setHeight(x.left); heightL = x.left.height;**

 **}**

 **if (x.right != null) {**

 **setHeight(x.right); heightR = x.right.height;**

 **}**

 **x.height = max(heightL, heightR) + 1;**

**}**

The function call would then be:

**if (ourTree.root != null) setHeight(ourTree.root);**

A more compact alternative function:

**int setheight(node x) {**

 **if (x == null) return -1;**

 **return (x.height = 1 + max(setHeight(x.left), setHeight(x.right)));**

**}**

The function call would then be:

**int foo = setVyska(ourTee.root);**

**15.** Write a function which will create an exact copy of a binary tree.

**Solution**

 **Node copyBinTree (Node root) {**

 **if (root == null) return null;**

 **Node root2 = copyNode(root);** // easy copy of a single object

 **root2.left = copyBinTree(root.left);**

 **root2.right = copyBinTree(root.right);**

 **return root2;**

 **}**

The function call would then be:

**Node newRoot = copyBinTree(rootOfOriginalTree);**

**16.** Write a function which modify the tree in such way that it will become a mirror copy of itself. This means that the Inoder traversal of the original tree and the same inorder traversal of the modified tree will yield sequences which are reverse of each other.

**Solution**

 **void mirror (Node root) {**

 **if (root == null) return;**

 **mirror(root.Left);**

 **mirror(root.Right);**

 **Node tmp = root.Left;**

 **root.Left = root.Right;**

 **root.Right = tmp;**

 **}**

**17.**  Write a function which will merge two binary trees in the following way:

1. It removes one of the deepest leaves in the tree which depth is not smaller the the depth of the other tree.

2. The removed node becomes the root of the new tree. The tree from which the node was removed becomes the left subtree of the root and the other tree becomes the right subtree of the root.

No nodes should be physically deleted by the operation, you should only manipulate the references of the nodes.

**Solution** Pseudocode:

  **Node mergeTrees (Node root1, Node root2) {**

 **int depth1 = setHeight(root1);**

 **int depth2 = setHeight(root2);**

 **if (depth2 > depth1)**

 **{ Node x = root1; root1 = root2; root2 = x;} //** *swapRoots*

 **Node newRoot = findDeepestNode(root1);** *// can be done in one traversal*

 *// or during the setHeight()*

 **if(newRoot.parent.left == newRoot) newRoot.parent.left = null;**

 **else newRoot.parent.right = null;**

 **newRoot.left = root1; root1.parent = newRoot;**

 **newRoot.right = root2; root2.parent = newRoot;**

 **return newRoot;**

 **}**