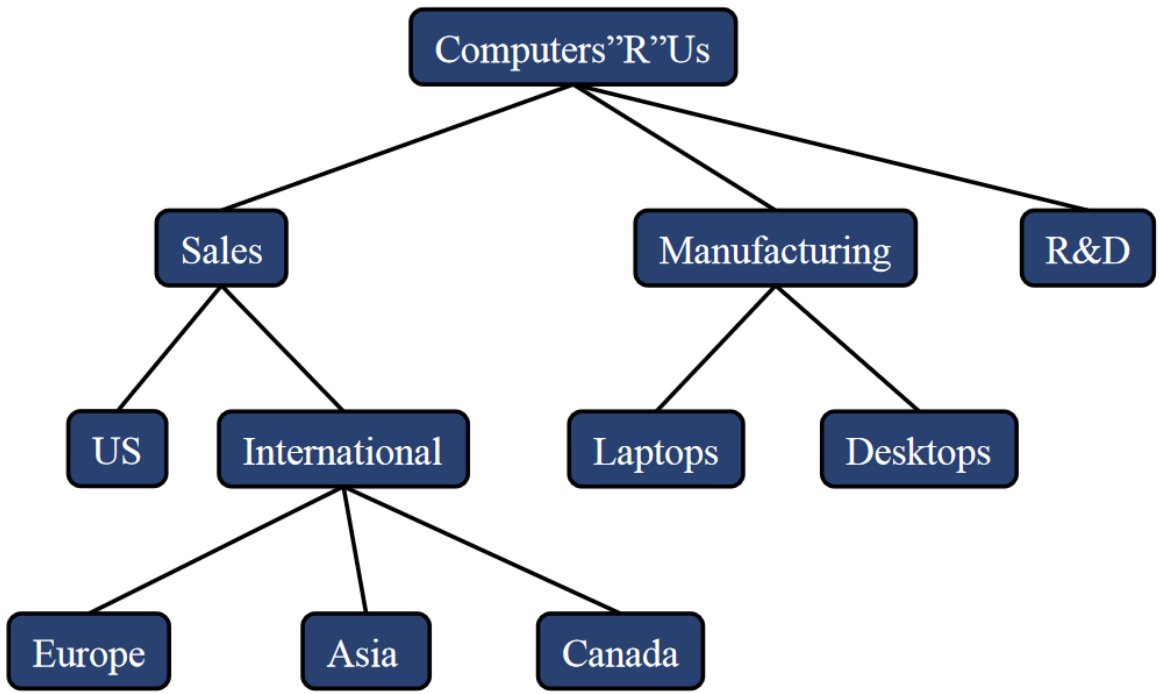
Binary Tree ADT

- Tree ADT
- Binary Search Tree (BST)
- Binary tree traversal

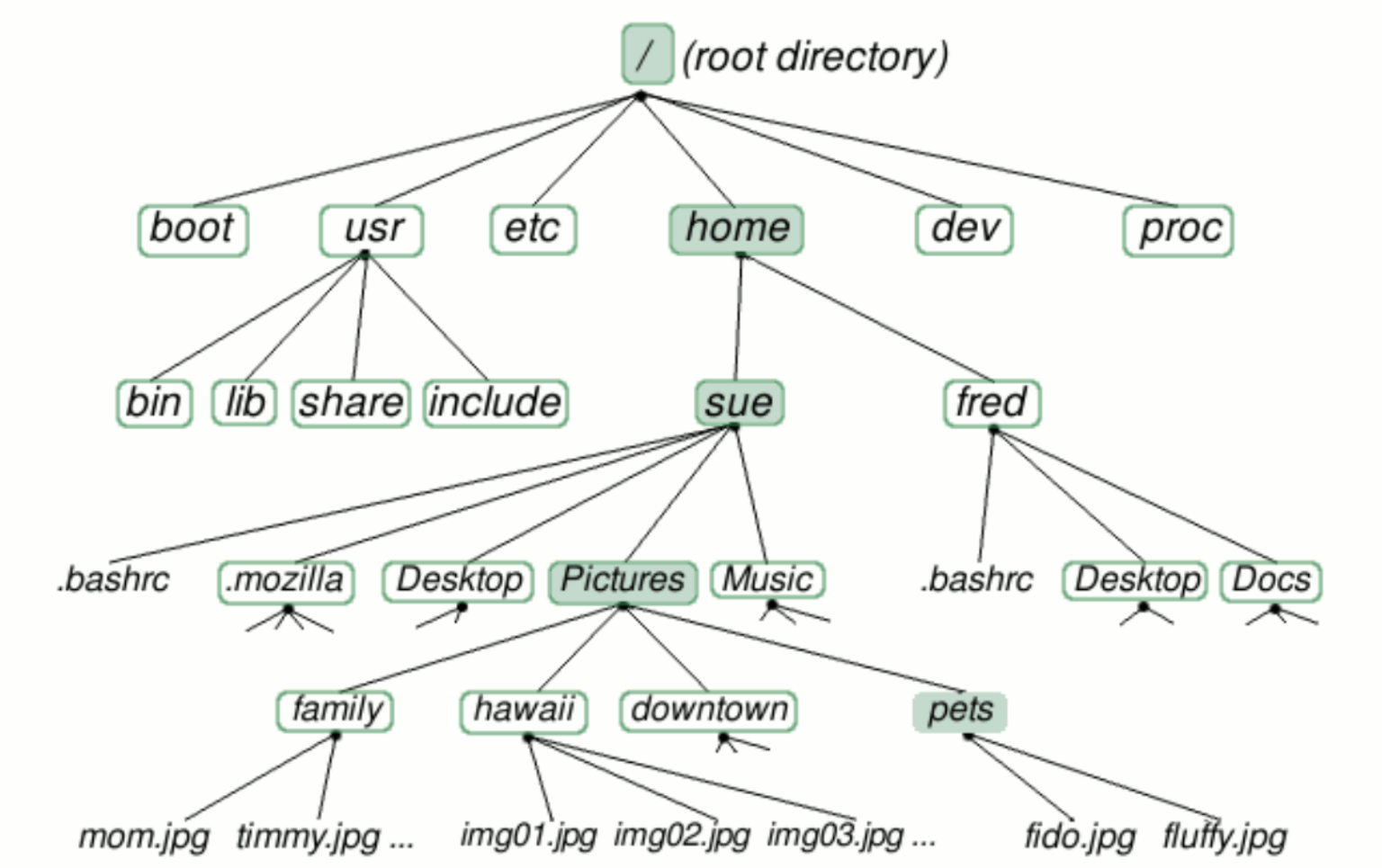
Introduction to Trees

- A tree (upside down) is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Each element (except the top element) has a parent and zero or more children elements



Tree examples

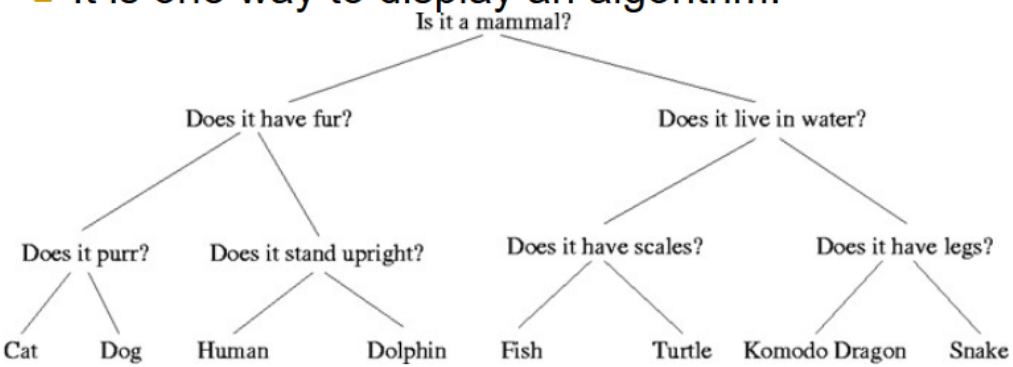
Linux/Unix file systems



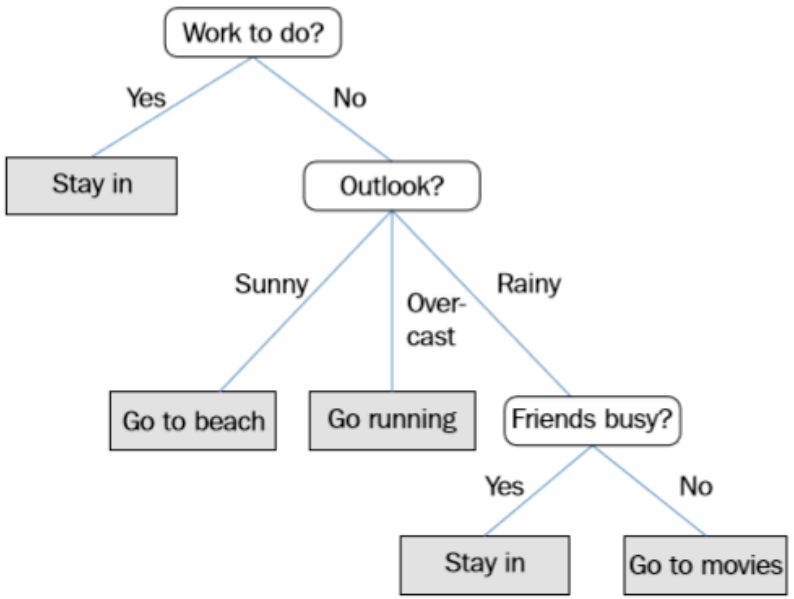
Tree examples

Tree Example – Decision Tree

- tool that uses a tree-like graph or model of decisions and their possible consequences
  - including chance event outcomes, resource costs, and utility
- It is one way to display an algorithm.



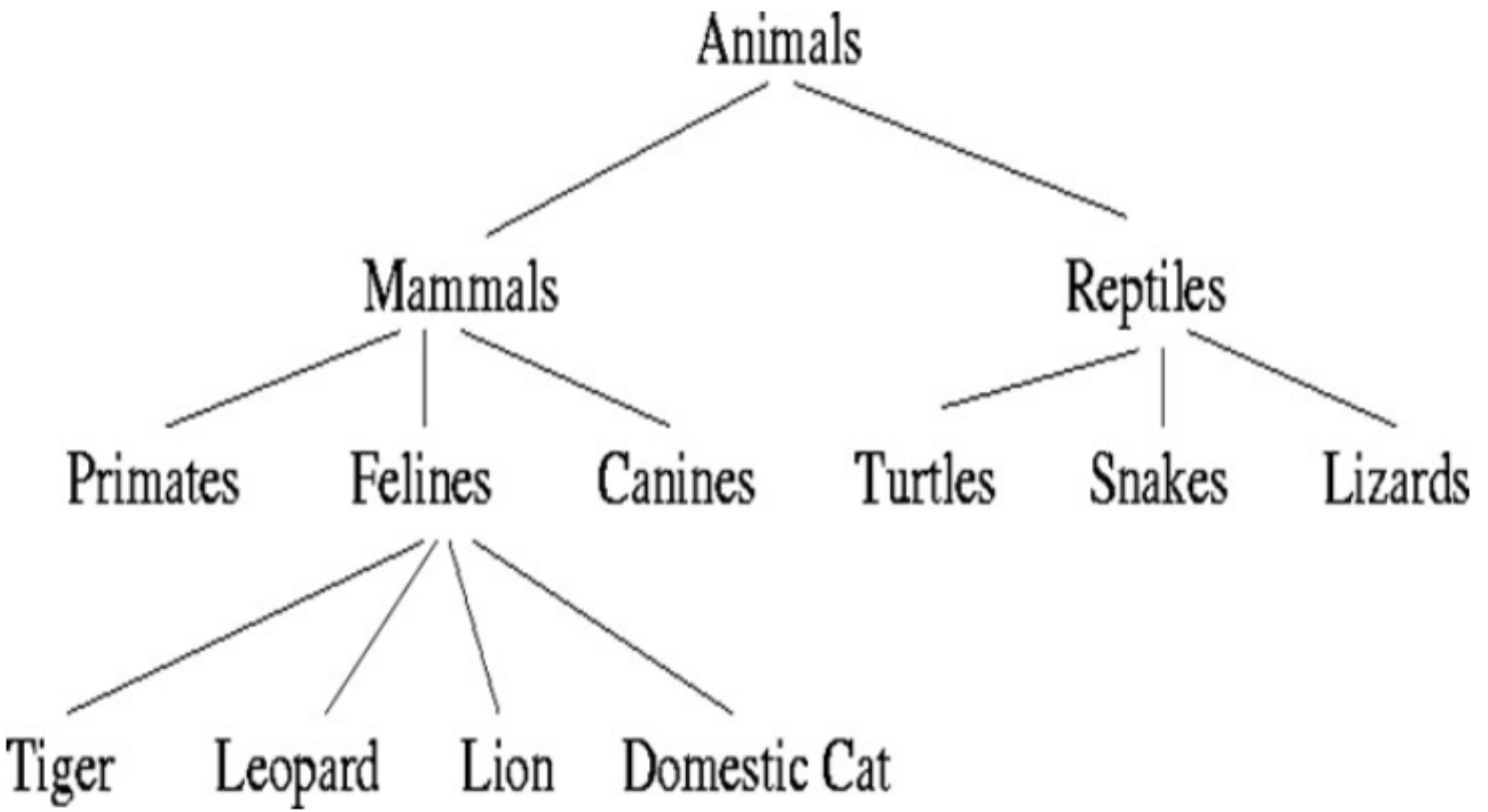
https://slideplayer.com/slide/8222301/



https://www.commonlounge.com/discussion/ee8b074936a041f2b5a57d2054dc3701

Tree examples

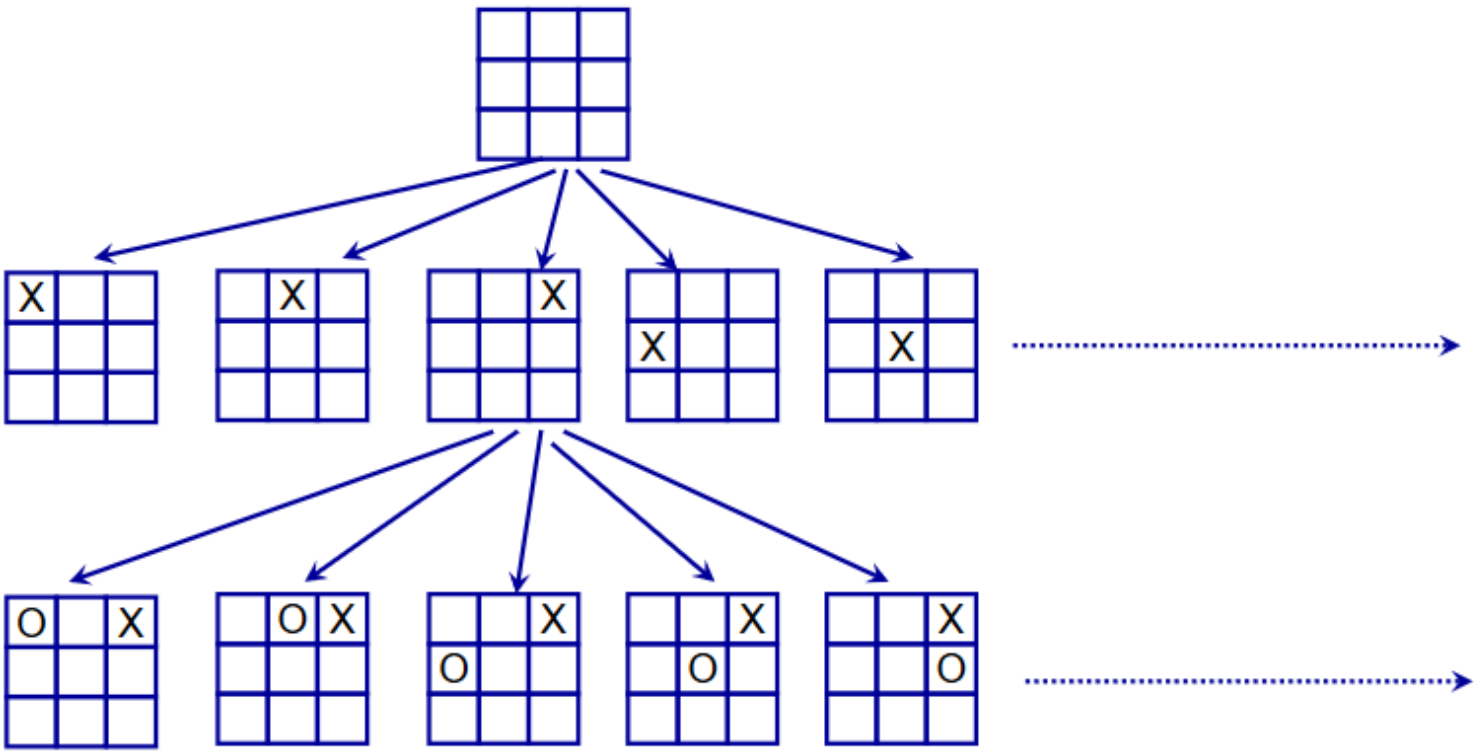
Tree Example – Taxonomy Tree



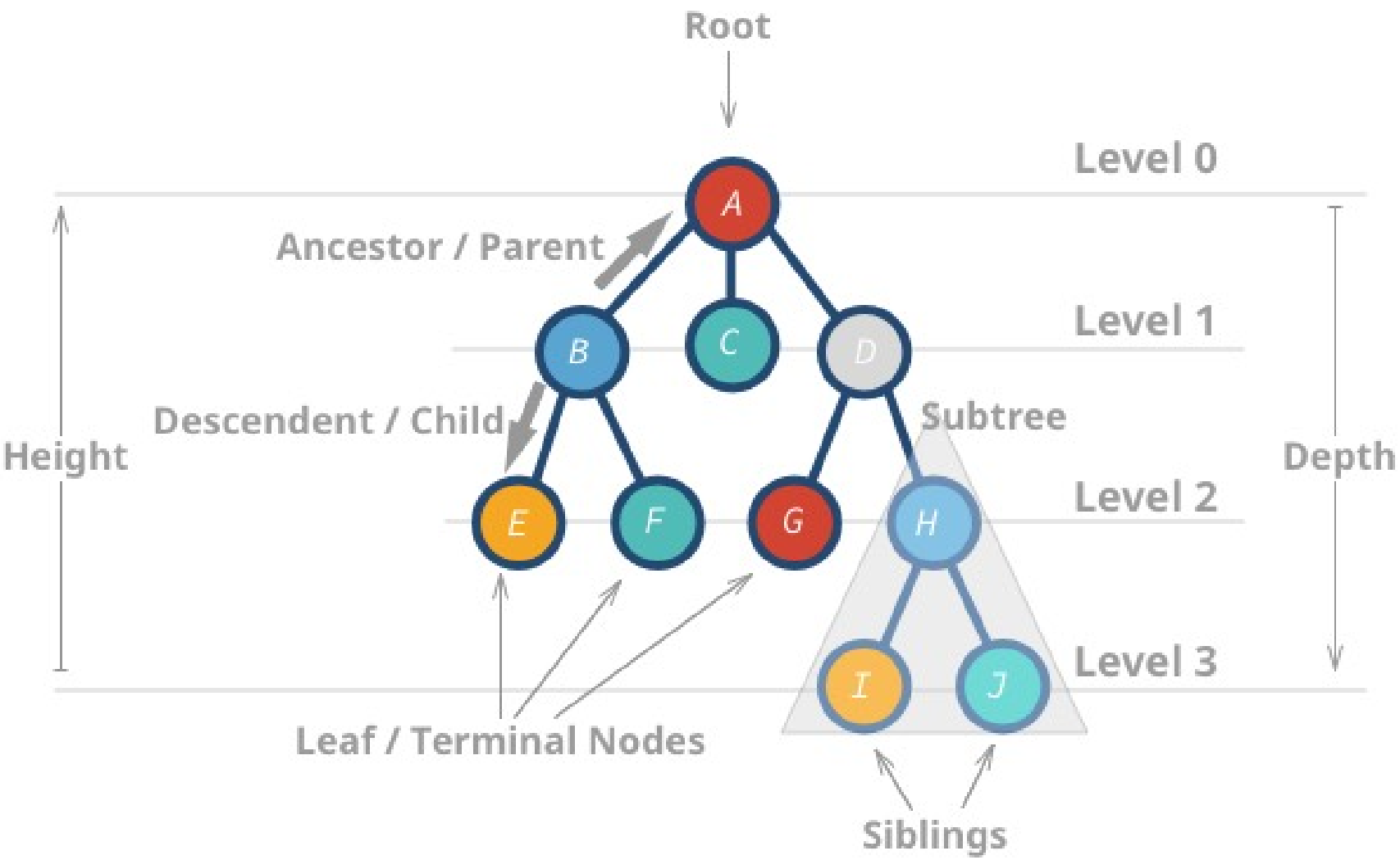
https://slideplayer.com/slide/8222301/

Tree examples

Tree Example – Tic Tac Toe



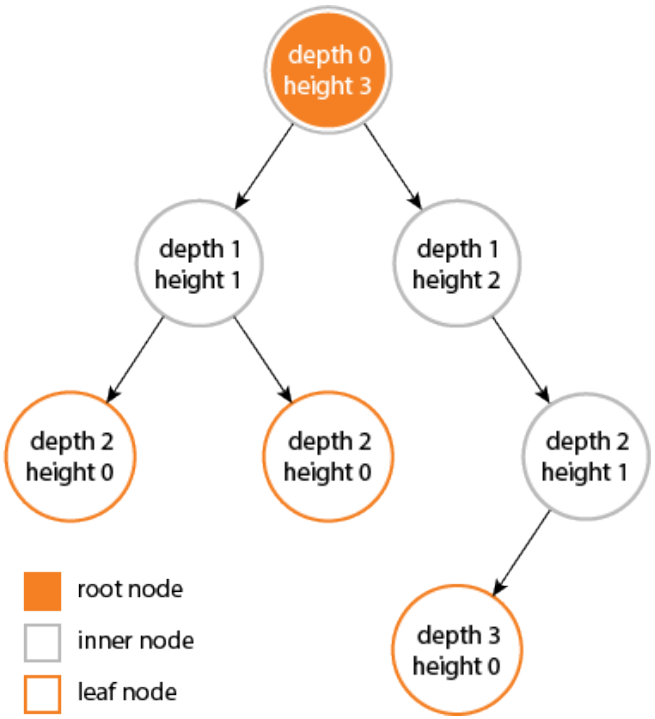
Definitions

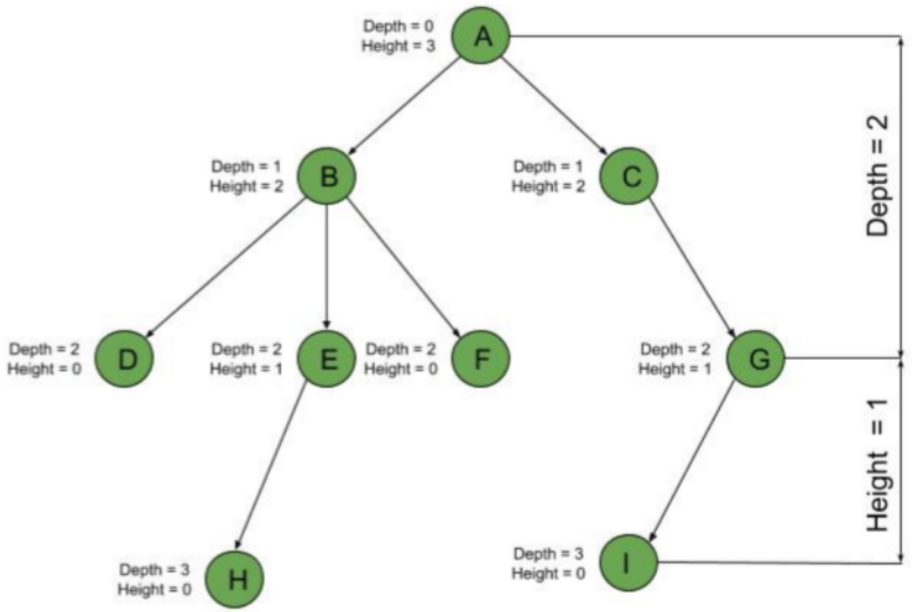


Definitions

- Root: the top node of the tree
- Edge: is the link between two nodes
- Child: a node with a parent node
- Parent: a node that has the edge leading to the child node
- Leaf: a node that does not have a child node
- Height: the longest distance to a leaf. The tree's height (h) is the distance between the farthest leaf and root (the number of edges).
- Depth: The depth of a node Y is
  - 0, if the Y is the root, or
  - 1 + the depth of the parent of Y
- Degree of a node: The number of child nodes.
- Degree of a tree: The highest degree node specifies the degree of the tree.

Depth and height of a tree

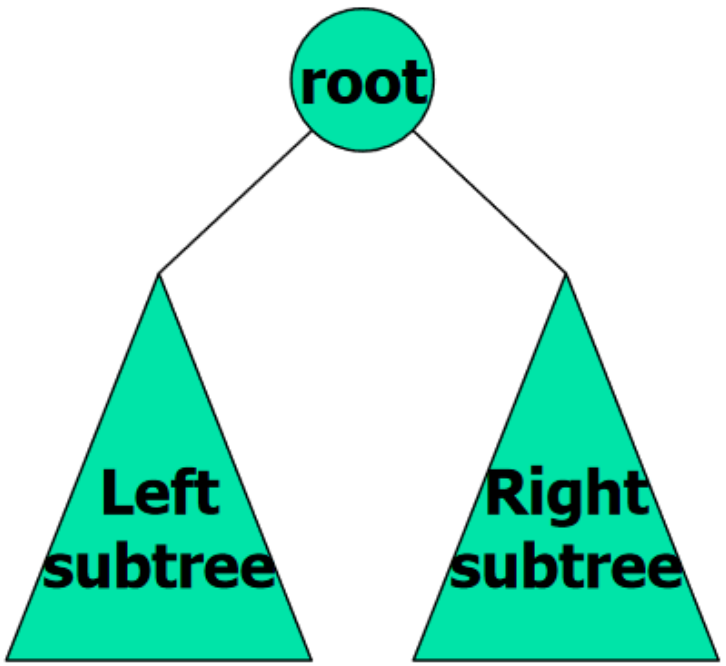




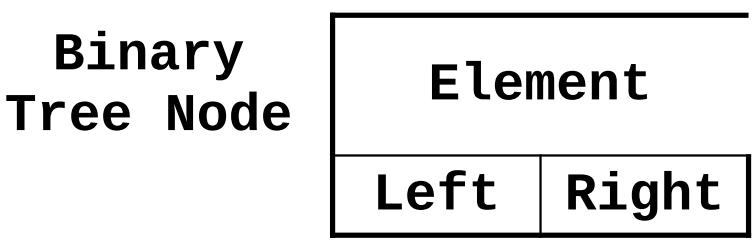
https://www.baeldung.com/cs/tree-depth-height-difference

https://stackoverflow.com/questions/2603692/what-is-the-difference-between-tree-depth-and-height/2603707#2603707

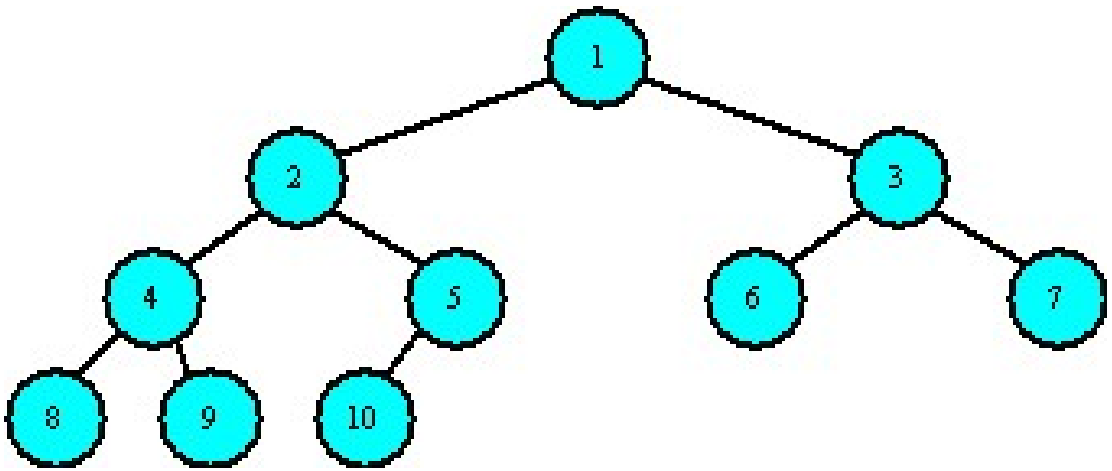
Binary trees



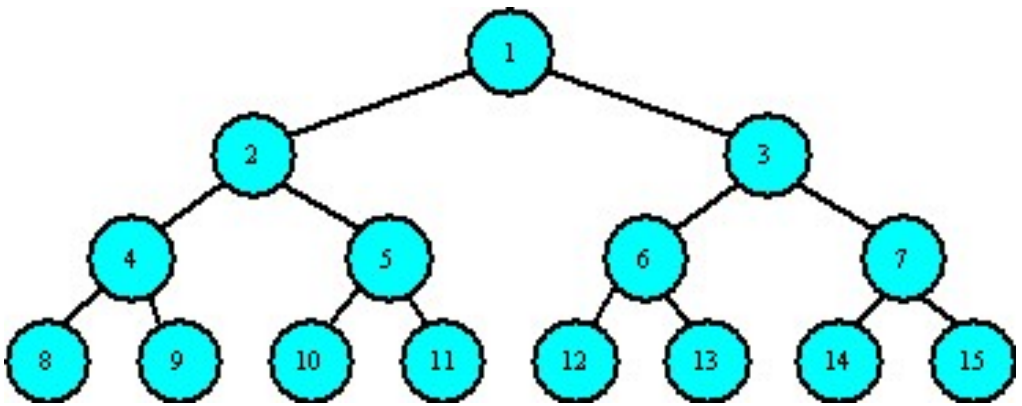
- A tree type with a maximum of two children of the nodes.
- These nodes are referred to as the left child and the right child



Complete and full binary tree cases



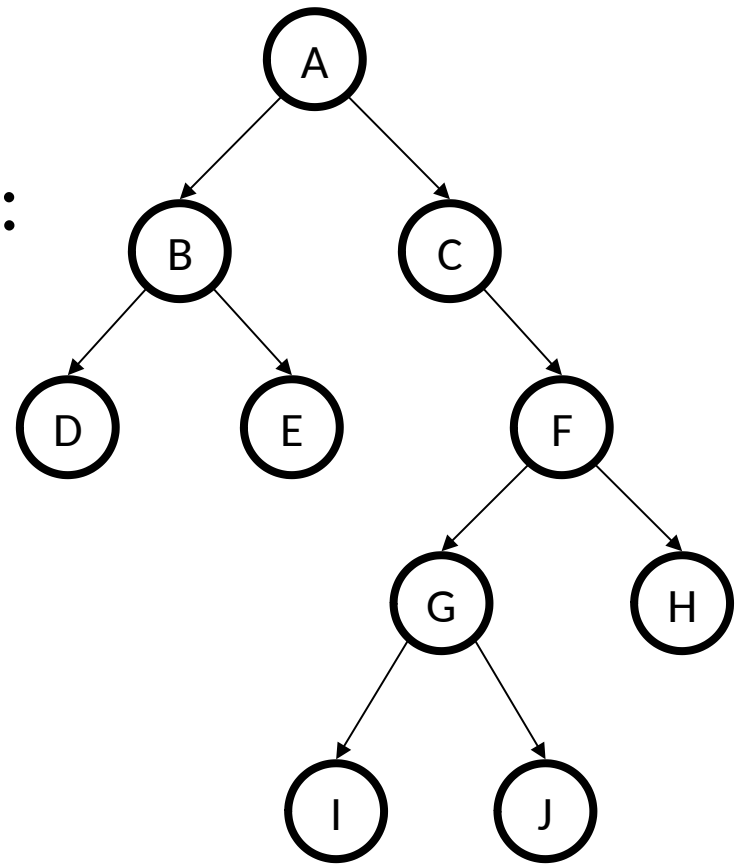
Complete Binary Tree



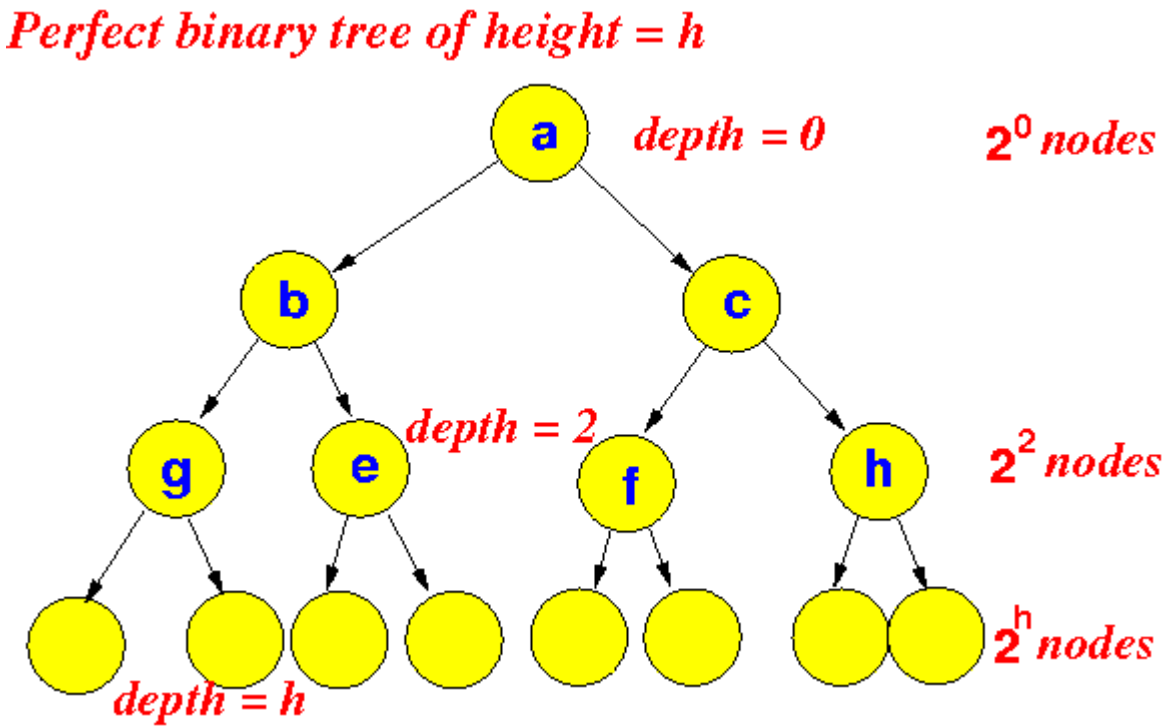
Full Binary Tree

Binary tree max depth – min depth

- Properties
  - max # of leaves = $2^{depth(tree)}$
  - max # of nodes = $2^{depth(tree)+1} - 1$
- We care a lot about the depth:
  - max depth = n-1
  - min depth = log(n)
  - average depth for n nodes = $\sqrt{n}$ (over all possible binary trees)



Number of nodes



$N=2^{d+1}-1$

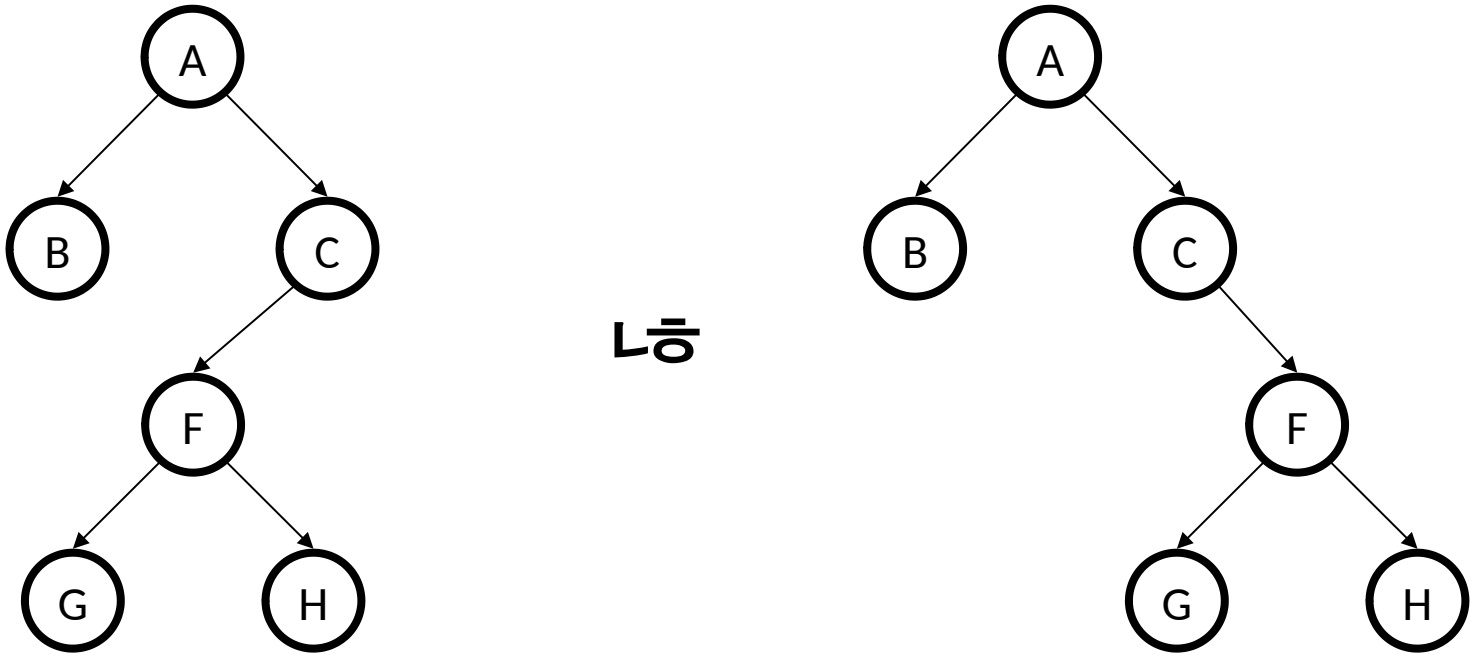
$N+1=2^{d+1}$

$d+1=\log_2(N+1)$

$d=\log_2(N+1)-1$

Max number of nodes $=2^0+2^1+2^2+\ldots+2^d=$ **$2^{d+1}-1$**

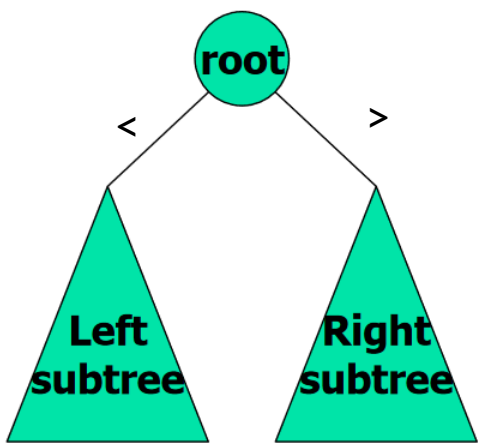
Left child and right child



https://courses.cs.washington.edu/courses/cse326/

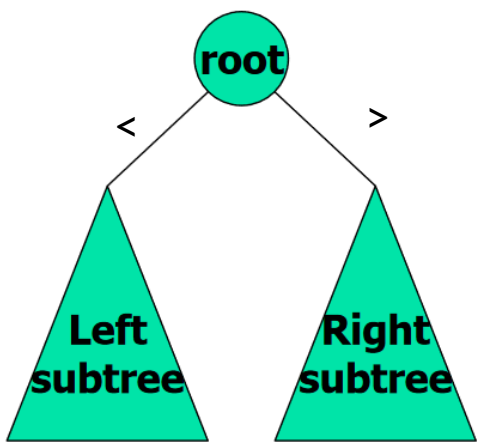
Binary Search Tree - BST

- In computer science, a binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree data structure whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree.
- Binary search trees allow binary search for fast lookup, addition and removal of data items, and can be used to implement dynamic sets and lookup tables.
- The order of nodes in a BST means that each comparison skips about half of the remaining tree, so the whole lookup takes time proportional to the binary logarithm of the number of items stored in the tree.
- This is much better than the linear time required to find items by key in an (unsorted) array, but slower than the corresponding operations on hash tables.

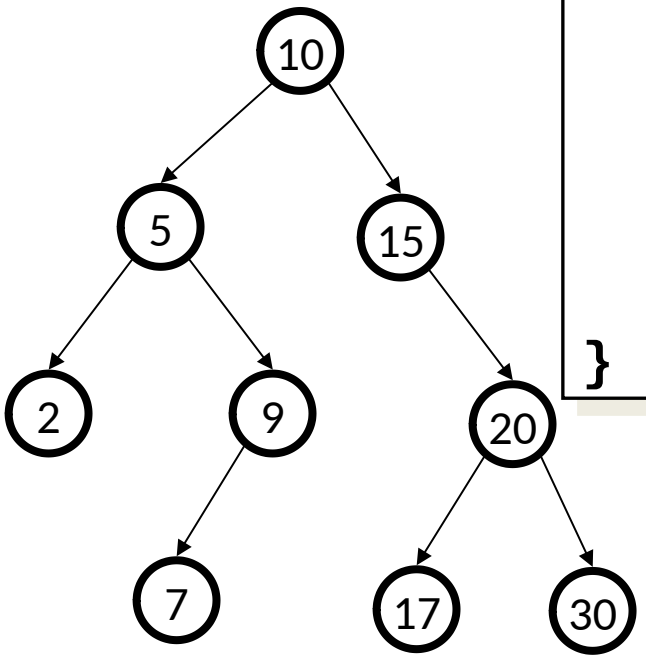


Binary Search Tree - BST

- Binary Search Tree has the following features:
  - A node's left subtree contains only nodes with keys that are smaller than the node's key.
  - A node's right subtree contains only owner nodes with keys greater than the node's key.
  - There should be no duplicate nodes.



Search for an element

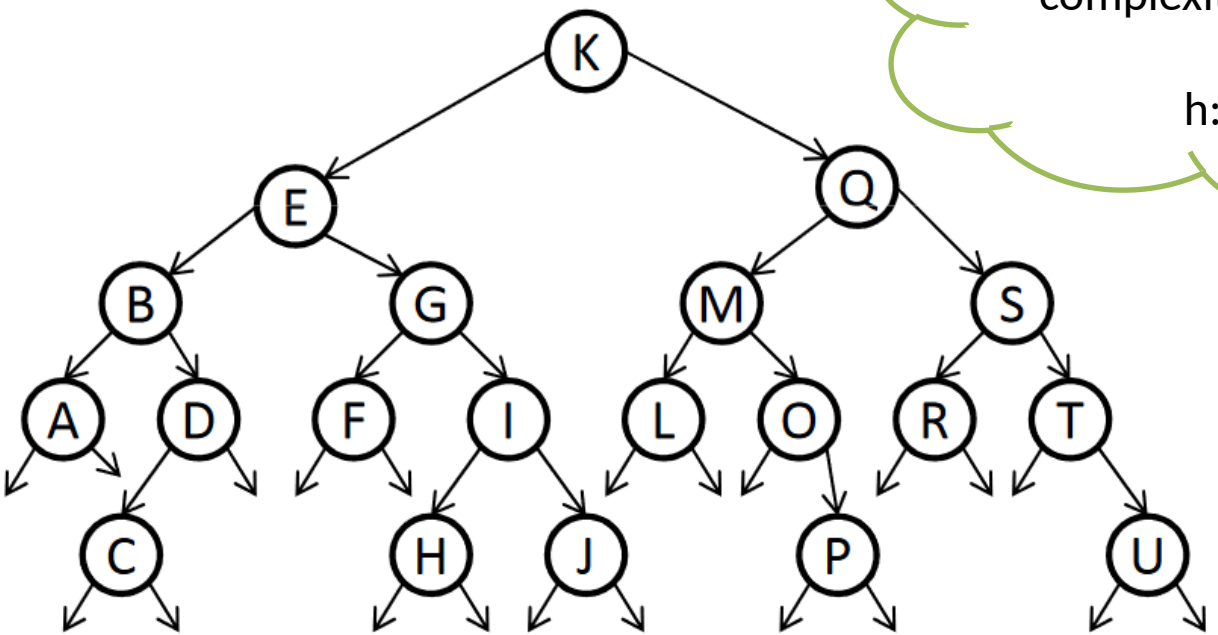



```
Boolean find(int x, TreeNode T)
{ if  ( T == NULL )
      return false;
  if (x == T.Element)
      return true;
  if (x < T.Element)
     return find(x, T.Left);
  return find(x, T.Right);
}
```

What is the running time ?

Search for an element

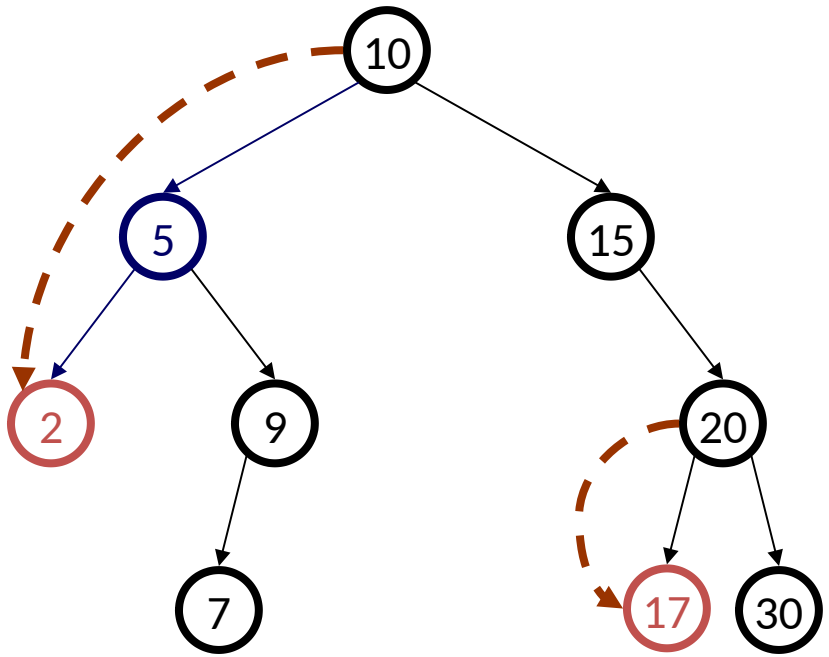
At best the tree is balanced.
Worst case computational complexity for seaching:
O(h)
h: height



http://www.cs.princeton.edu/courses/archive/spr11/cos423/Lectures/BinarySearchTrees.pdf

Find minimum

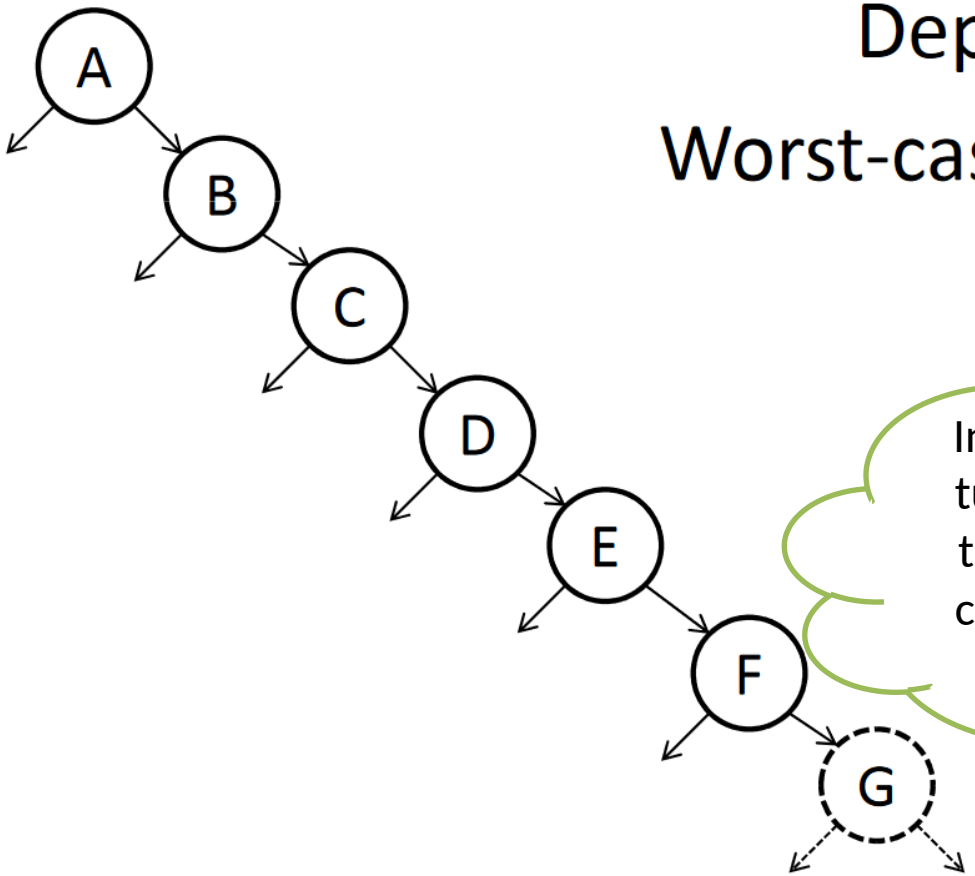
If we want to find the minimum element of the tree, we need to go to the leftmost element of the tree, and if we want to find the maximum element, we need to go to the rightmost element.




```
TreeNode min(Node T) {
  if (T.Left == NULL)
    return T;
  else
    return min(T.Left);
}
```

Search for an element: Worst case

Insert A, B, C, D, E, F, G,…



Depth of tree is $n - 1$.

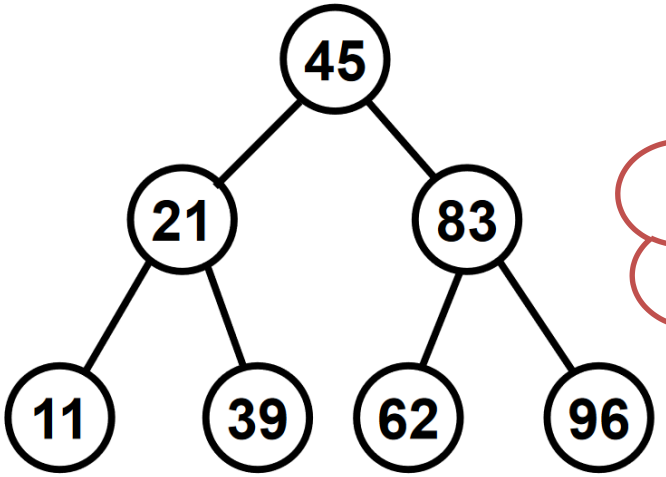
Worst-case access cost is n.

= list!

In the worst case, the tree turns into a linear list. In a tree with N elements, the computational complexity turns out to be O(n).

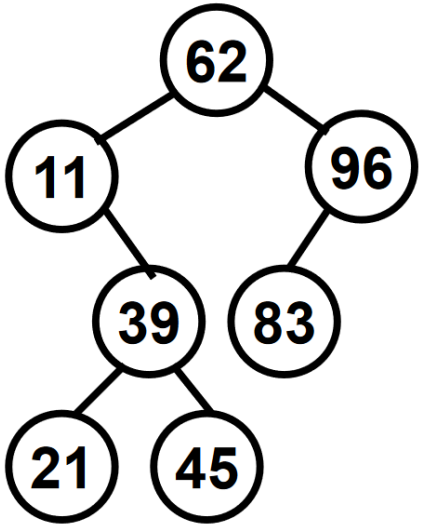
Adding an elments to BST

45 21 39 83 62 96 11



If we enter the same elements in different order, different BST will appear, the balance of the BST may change.

62 96 11 39 21 83 45



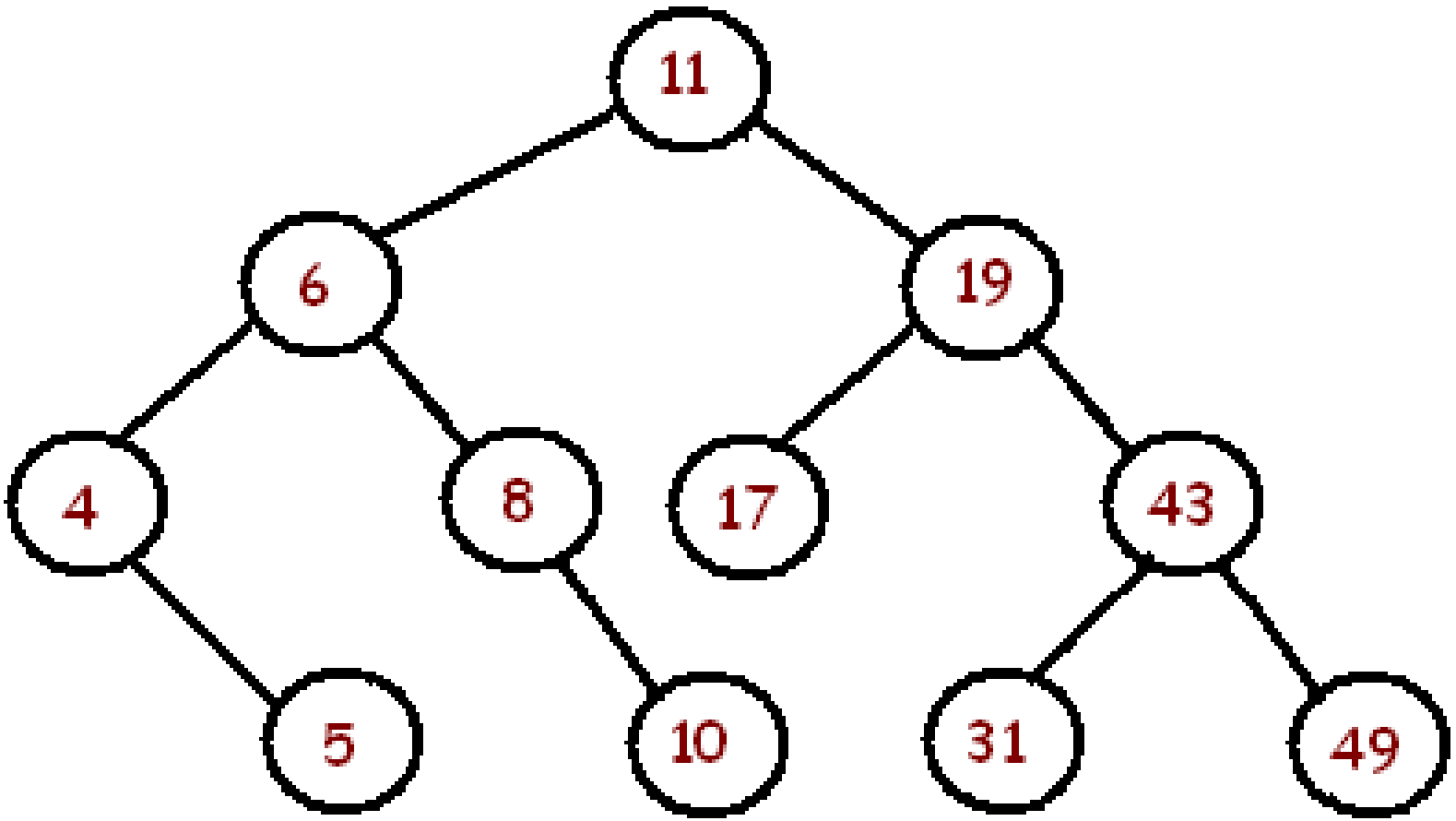
Example

If we add the following elements to a BST data structure sequentially, draw the tree to be obtained.

Elements: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

Example

Elements: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

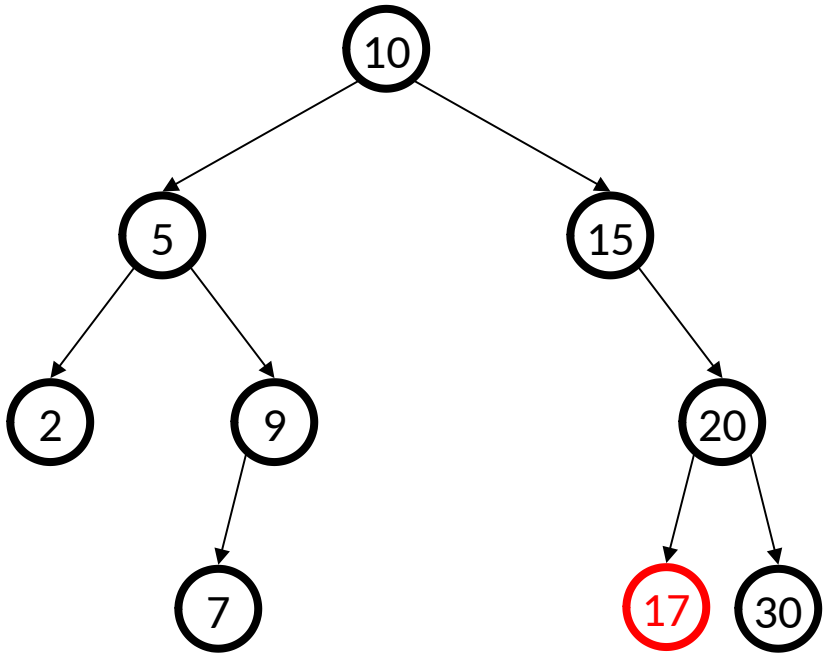


Deletion

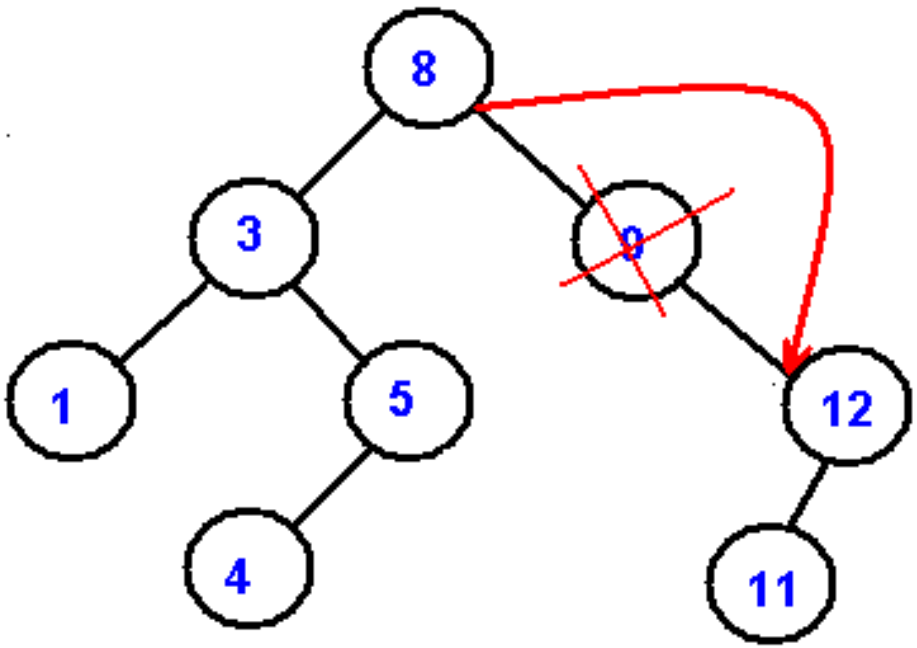
- There are three cases when deleting a node from a BST:
  - Leaf case
  - One child
  - caseTwo child case

Deletion – leaf case

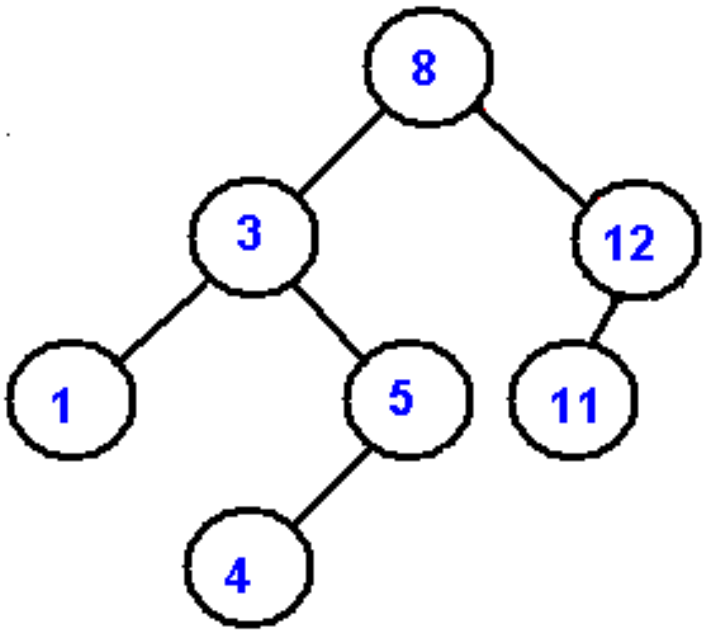
Delete(17)



Deletion – one child case



before deletion

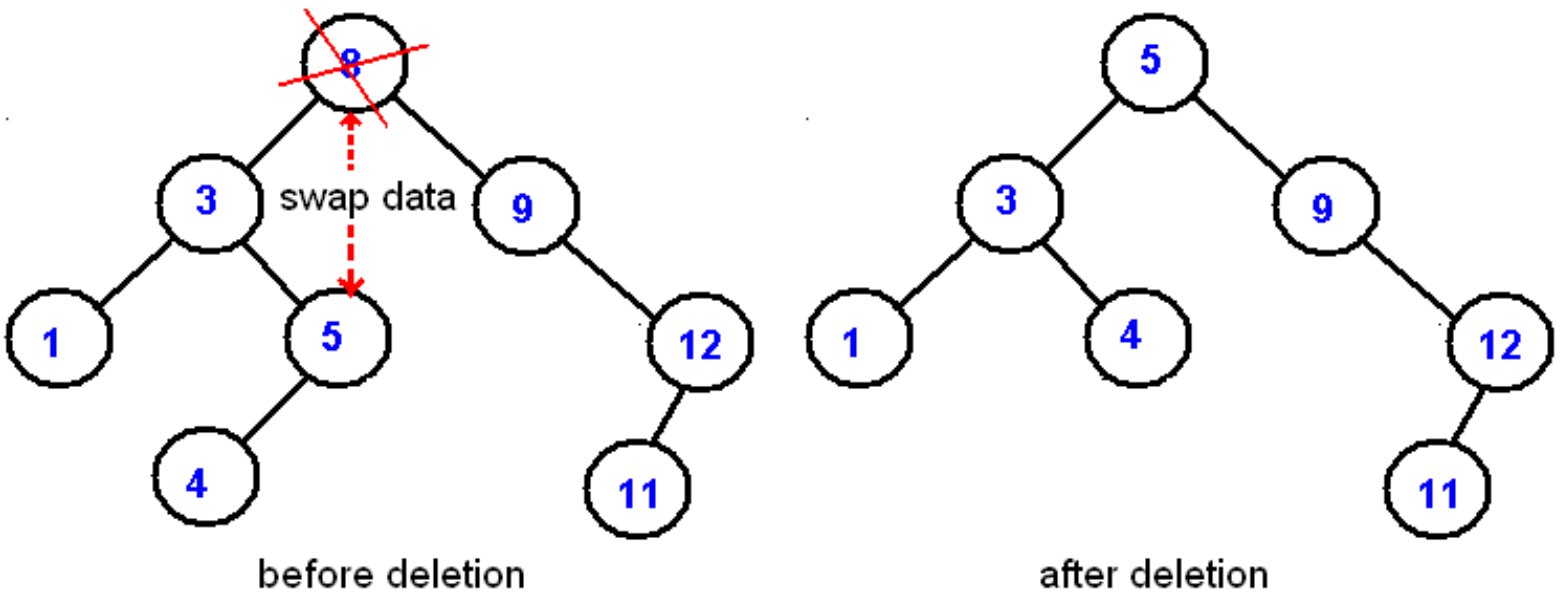


after deletion

Deletion – two child case

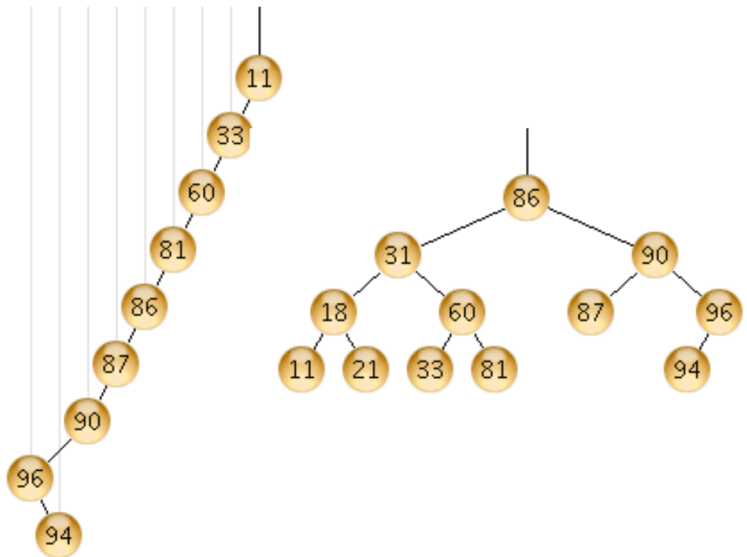
- There are two possible cases when deleting a node with two children:
  - The node to be deleted is replaced with the largest node of the left subtree.
  - Or the node to be deleted can be replaced with the smallest node of the right subtree.

Deletion – two child case



BST performance

- Because of search property, all operations follow one root-leaf path
  - insert: O(h)
  - delete: O(h)
  - search: O(h)
- We know that in a tree of n nodes
  - h >= lg (n+1) - 1
  - h <= n-1
- So in the worst case h is O(n)
  - BST insert, search, delete: O(n)
  - just like linked lists/arrays



- Ekleme, silme ve arama işlemlerinin hesaplama karmaşıklığı ağacın yüksekliği ile doğru orantılıdır.
- Ancak yükseklik, ***aynı veri kümesi için*** düğümlerin eklenme sıralarına göre aşağıdaki şekillerde görüldüğü gibi değişebilir.
- (log(n+1)-1)<=h<=n-1

Question

a) If we add the following elements to the binary search tree sequentially, draw trees to be obtained.
b) When the node with the value 10 is deleted, draw the new tree.

Elemanlar: 10, 5, 7, 17, 5, 11, 4, 19, 41, 45, 30

Tree traversal

- **Depth-First Search - DFS**
- Depth-first search is a type of traversal that goes as deep as possible for each child before discovering the next sibling.
- Types: in-order, pre-order ve post-order.

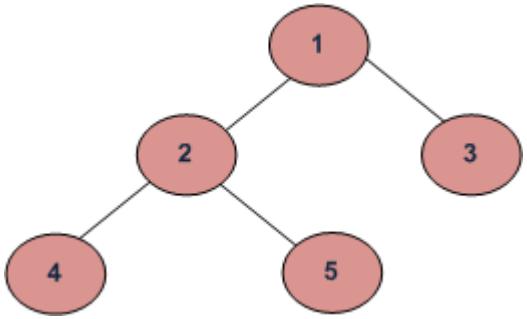
- **Breadth-First Search – BFS**
- It visits all nodes of a level before going to the next level.
- This type of traversal is also called level-order and visits all levels of the tree from left to right, starting at the root.

Tree traversal

Preorder Traversal (Practice):

Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)



Inorder Traversal (Practice):

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Postorder Traversal (Practice):

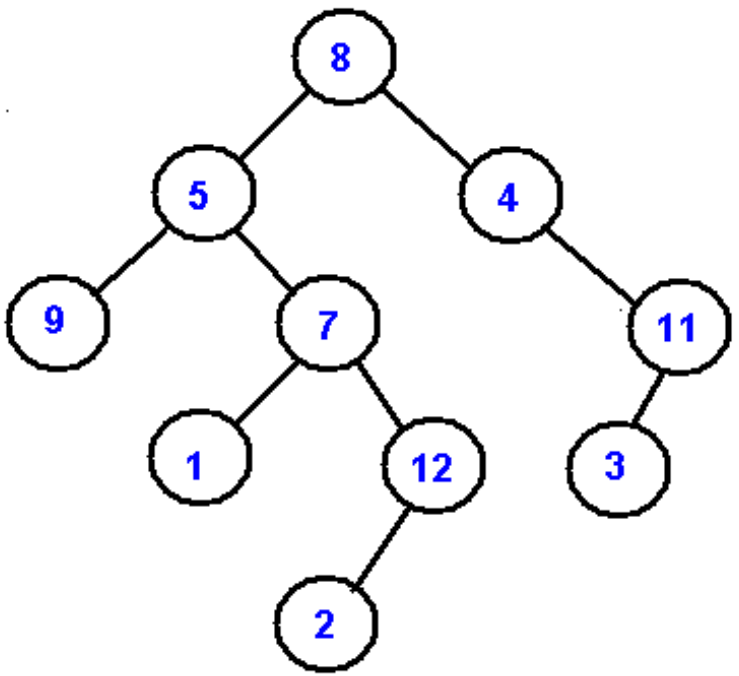
Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

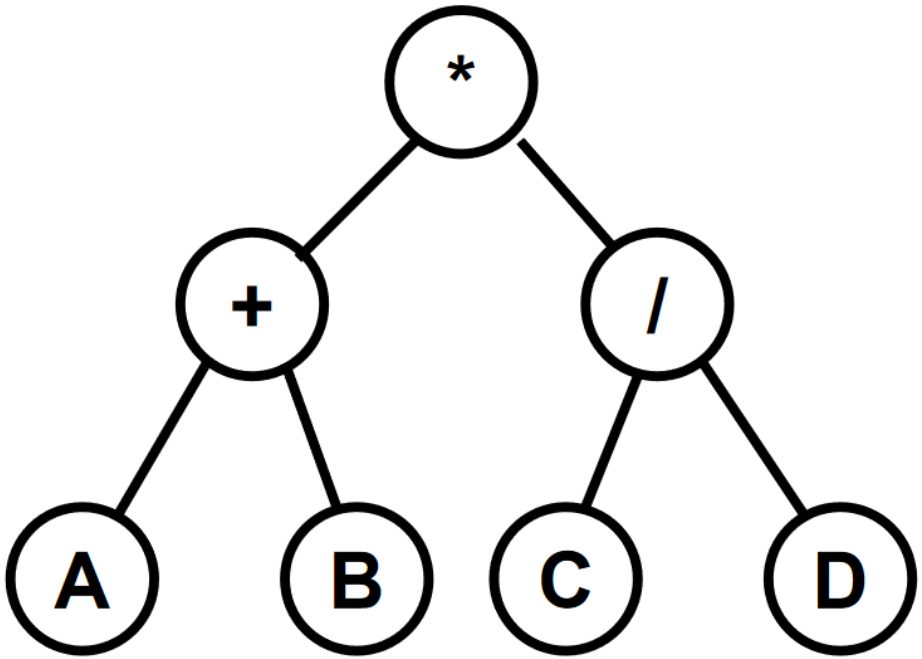
Example



PreOrder : 8, 5, 9, 7, 1, 12, 2, 4, 11, 3
InOrder : 9, 5, 1, 7, 2, 12, 8, 4, 3, 11
PostOrder : 9, 1, 2, 12, 7, 5, 3, 11, 4, 8
LevelOrder: 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

Example: Expression tree



preorder

*+AB/CD

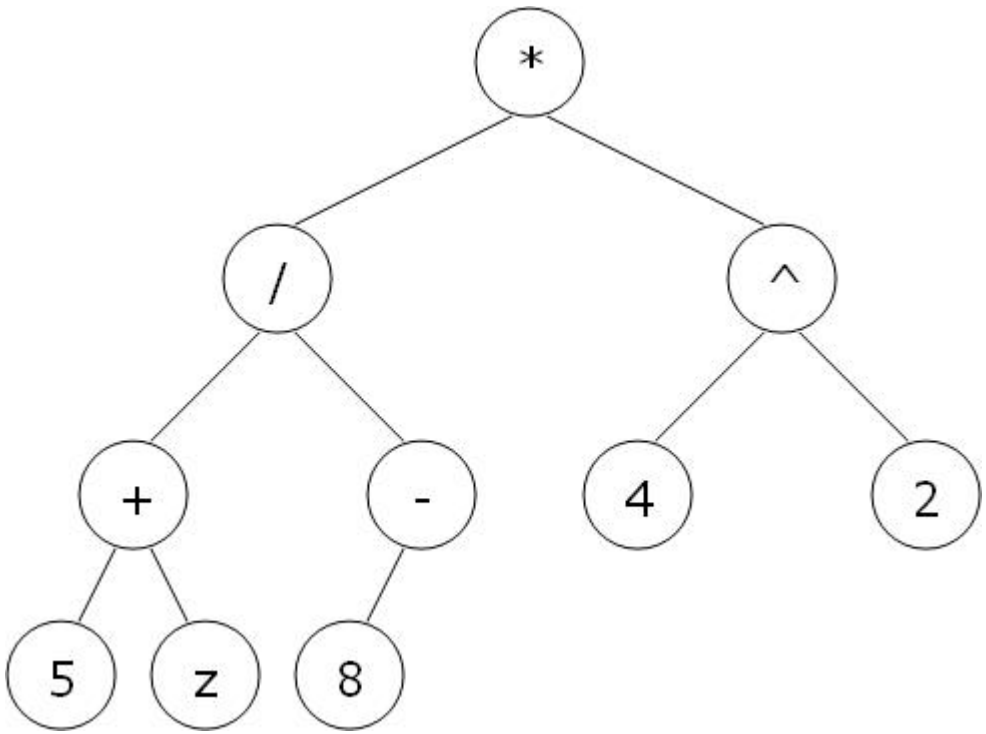
inorder

A+B*C/D

postorder

AB+CD/*

Question



Preorder : ?

Postorder: ?