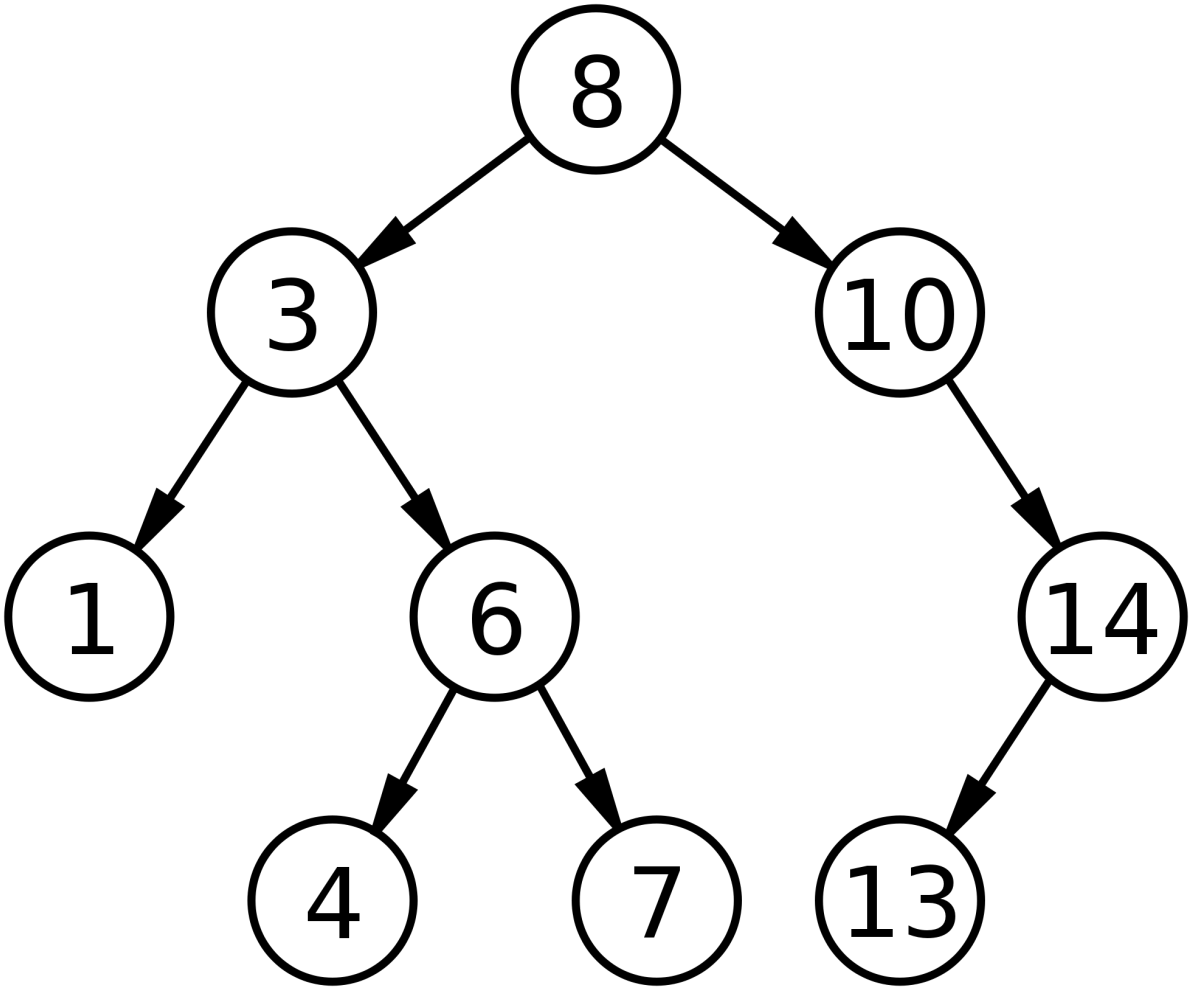
Binary Trees

CSCI-2270

Elizabeth Boese

BST



Binary Tree

Binary Trees

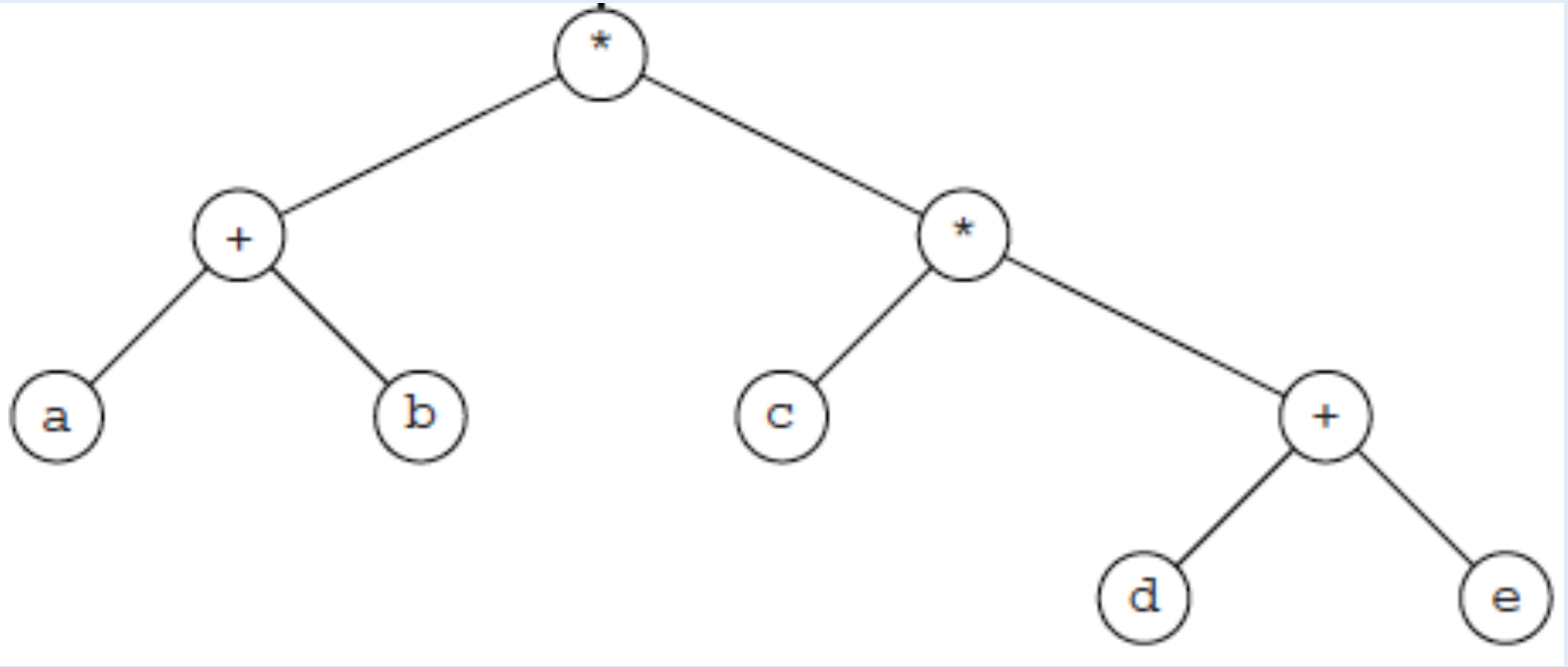
- At most 2 children
- *Can* work faster than lists/arrays
- Example uses
  - Expression trees using binary operators
  - An ancestral tree including an individual, his or her parents, their parents, and so on
  - Phylogenetic trees *(evolutionary biology)*
  - Binary space partitioning
  - Huffman coding (lossless data compression)
  - Strings in memory

Expression Trees

Expression Trees

if interested, take PL, static analysis

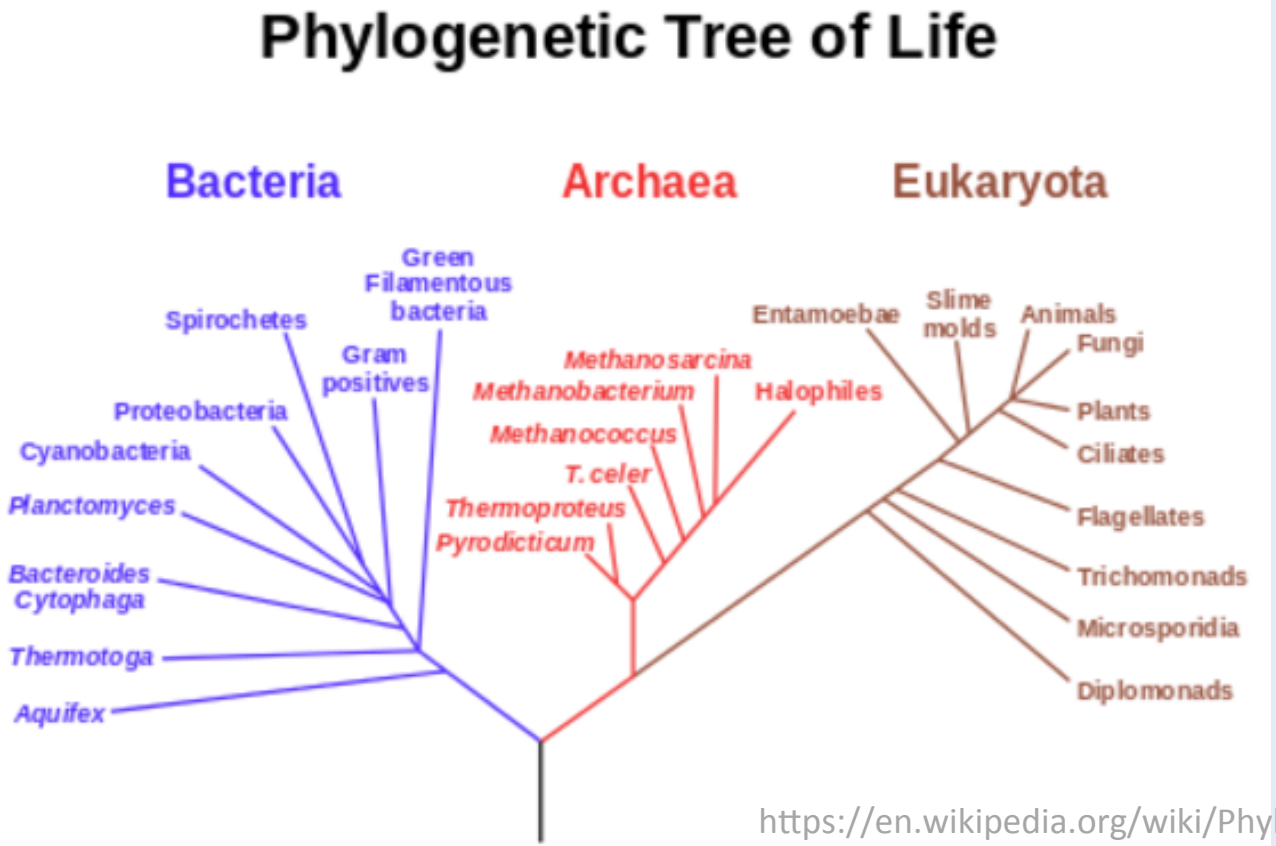
(a + b) * (c * (d + e))



Phylogenetic Trees

Phylogenetic Trees

if interested study DNA, genetics, see
http://www.cs.cornell.edu/courses/cs426/2003fa/week10%20phylogenetic%20trees.pdf

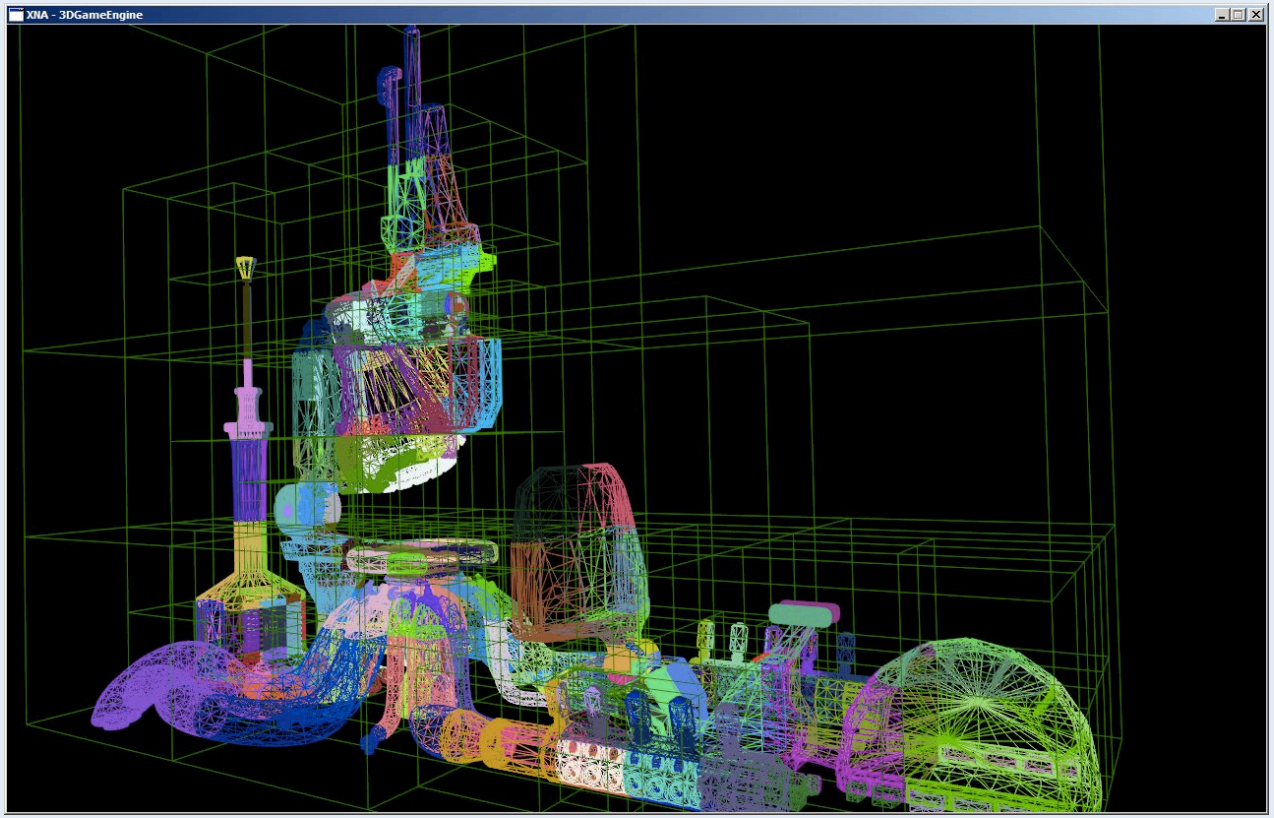


https://en.wikipedia.org/wiki/Phylogenetic_tree

Binary Space Partitioning Trees

BSP Trees

if interested, see
http://togeskov.net/xnagame.html



Huffman Coding

Image Compression using Huffman Coding

7	3	3	3
3	3	3	3
1	1	9	9
5	6	9	9

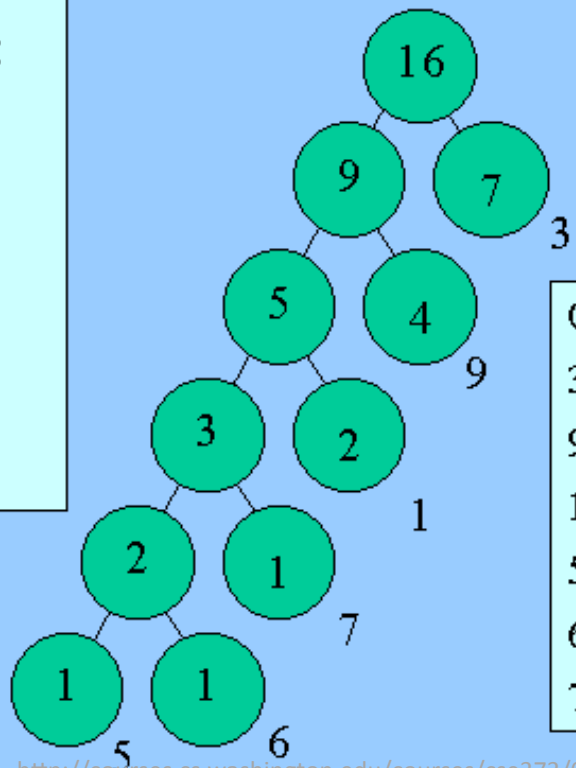
Frequency table:

1: 2
3: 7
5: 1
6: 1
7: 1
9: 4

Sorted table:

5: 1
6: 1
7: 1
1: 2
9: 4
3: 7

Huffman Tree:



Code

3: 1
9: 01
1: 001
5: 00000
6: 00001
7: 0001

To build the Huffman tree, keep taking the two smallest-weight subtrees and merge them. To code a pixel, trace the path from the root to that leaf, writing a 0 whenever going left and 1 whenever going right.

http://courses.cs.washington.edu/courses/cse373/02wi/slides/ImageADT/sld012.htm

eenshot.png

Ropes

C-strings are stored as arrays.

Arrays must be contiguous in memory.

How to store a 1,000 page manuscript?

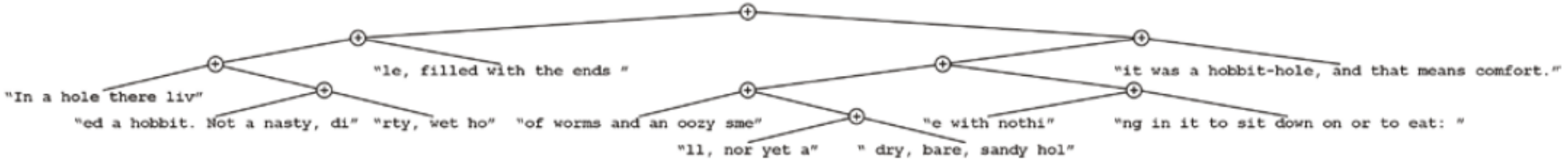


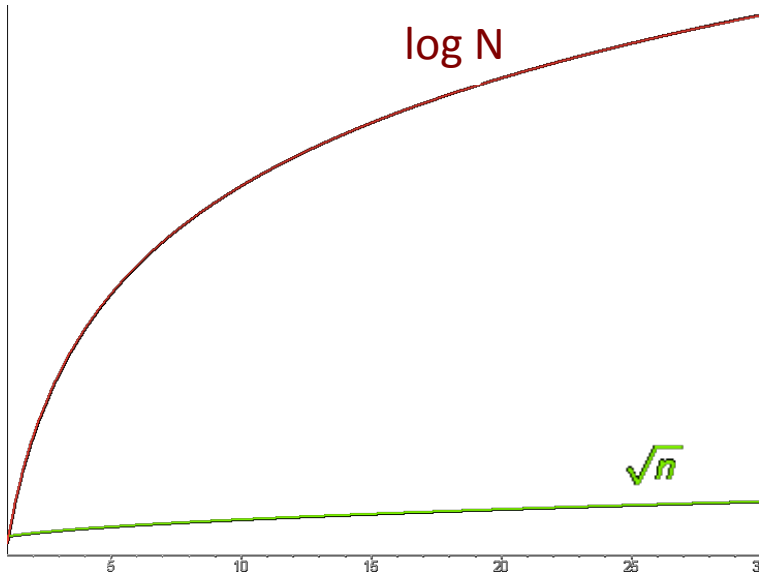
Figure 7. The first paragraph of "The Hobbit" representing the features of a rope.

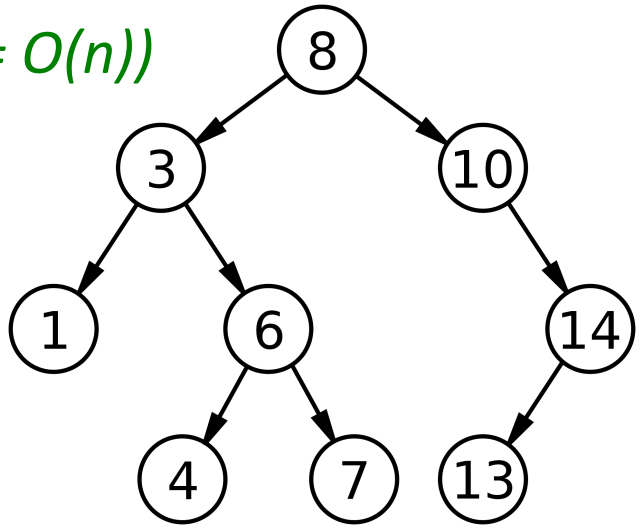
http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9450&rep=rep1&type=pdf

BST

- Binary Tree
  - Average depth = O(√N)
- Binary Search Tree
  - Average depth = O(log N)
  - Ordered
  - Worst-case search = O(h)

(which can == O(n))





Binary Tree

```
struct BinaryNode
{
    Type data;           // The data in the node
    BinaryNode *left;    // Left child
    BinaryNode *right;   // Right child
};
```

Traversal

Preorder

```
preorder_print (BinaryTree<ItemType> bintree) :
    if (bintree is not empty)
    {
        print out data at root
        preorder_print(left subtree of bintree's root)
        preorder_print(right subtree of bintree's root)
    }
```

Traversal

Inorder

```
inorder_print (BinaryTree<ItemType> bintree) :
    if (bintree is not empty)
    {
        inorder_print(left subtree of bintree's root)
        print out data at root
        inorder_print(right subtree of bintree's root)
    }
```

Traversal

Postorder

```
postorder_print (BinaryTree<ItemType> bintree) :
    if (bintree is not empty)
    {
        postorder_print(left subtree of bintree's root)
        postorder_print(right subtree of bintree's root)
        print out data at root
    }
```

Search

Recall that data in node n is >= data in n's left subtree
data in node n is < data in n's right subtree

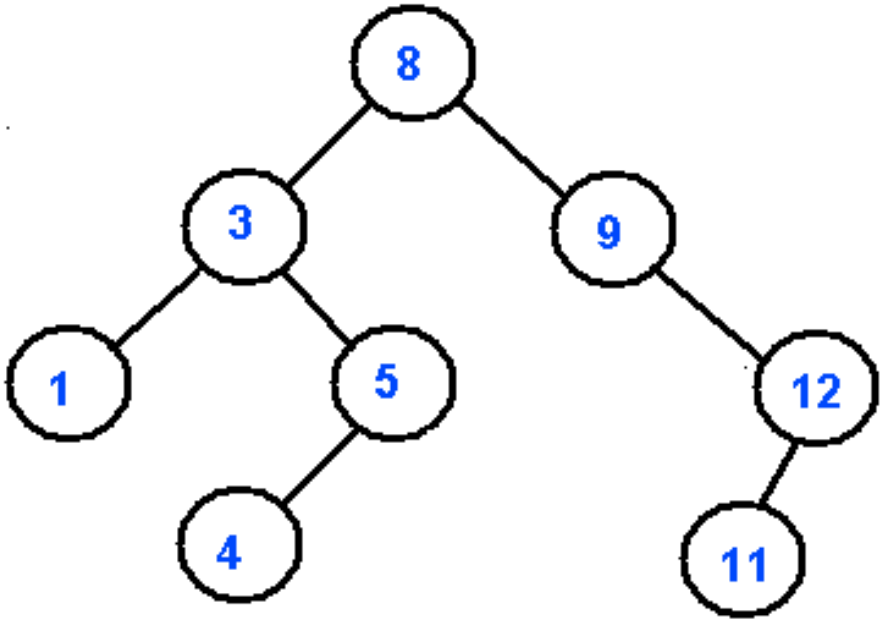
```
search(BinarySearchTree bsttree, ItemType target) :
    if (bsttree is empty)
        target not found, boo!
    else if (target equals data item at root)
        target found, yay!
    else if (target < data item at root)
        search(left subtree of bsttree, target)
    else
        search(right subtree of bsttree, target)
```

Height

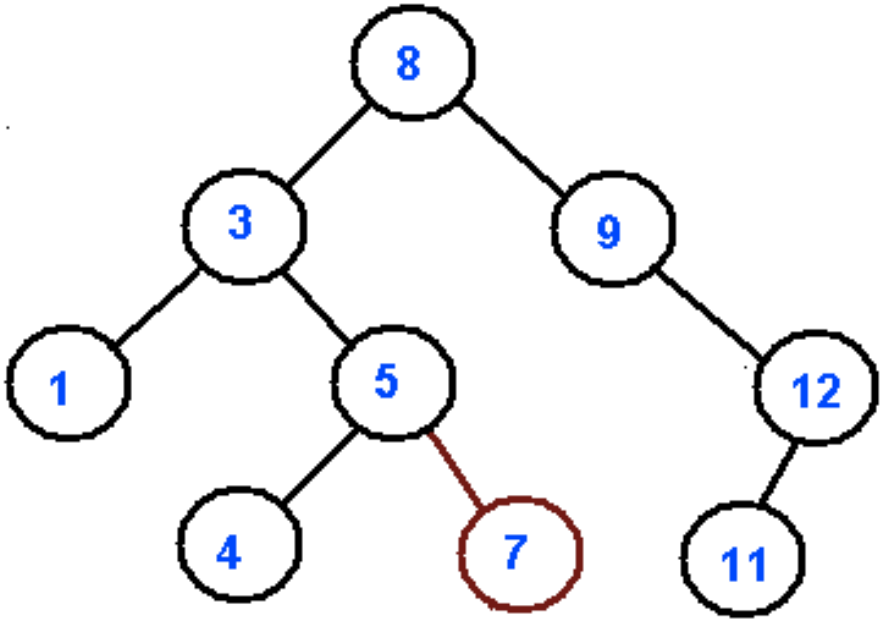
- Height
  - 1 + height of largest subtree
- Full binary tree
  - every node has either 0 or 2 children
- Complete binary tree
  - last row can be partially filled left to right, no skipped children

BST

Insert



before insertion



after insertion

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

BST

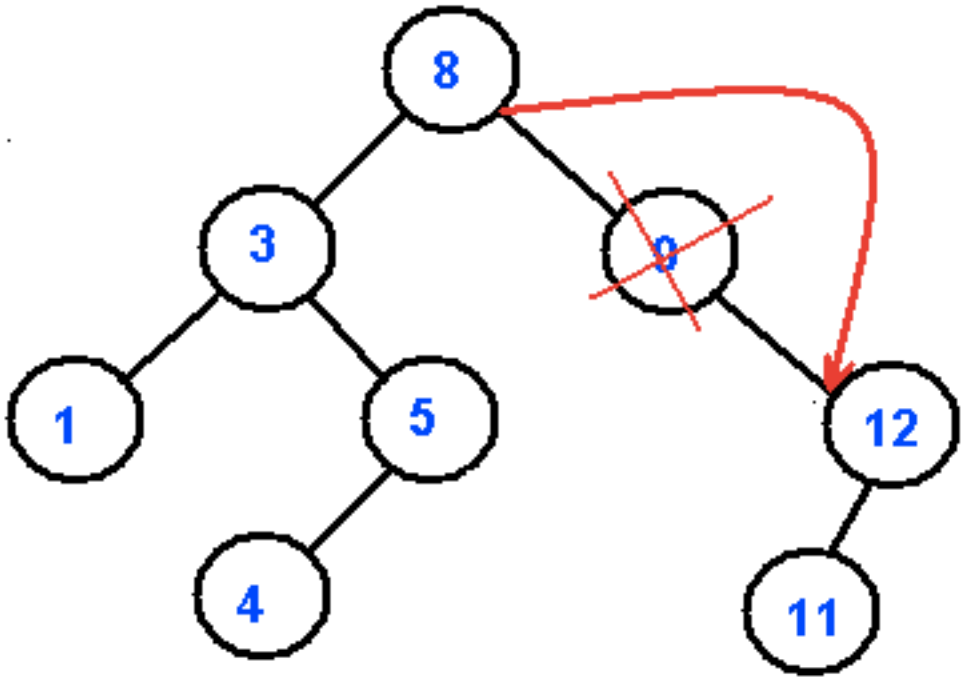
Delete

- is not in a tree
- is a leaf
- has only one child
- has two children

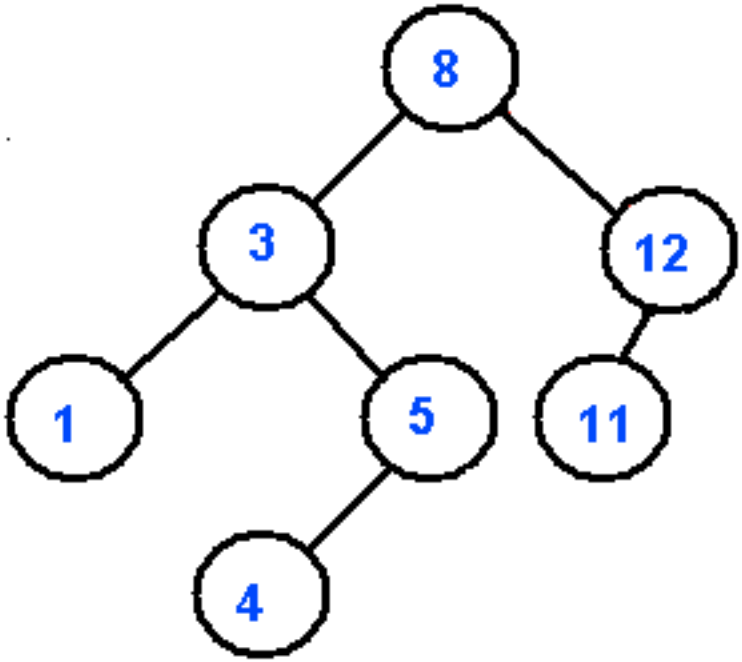
https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

BST

- Delete – has one child



before deletion

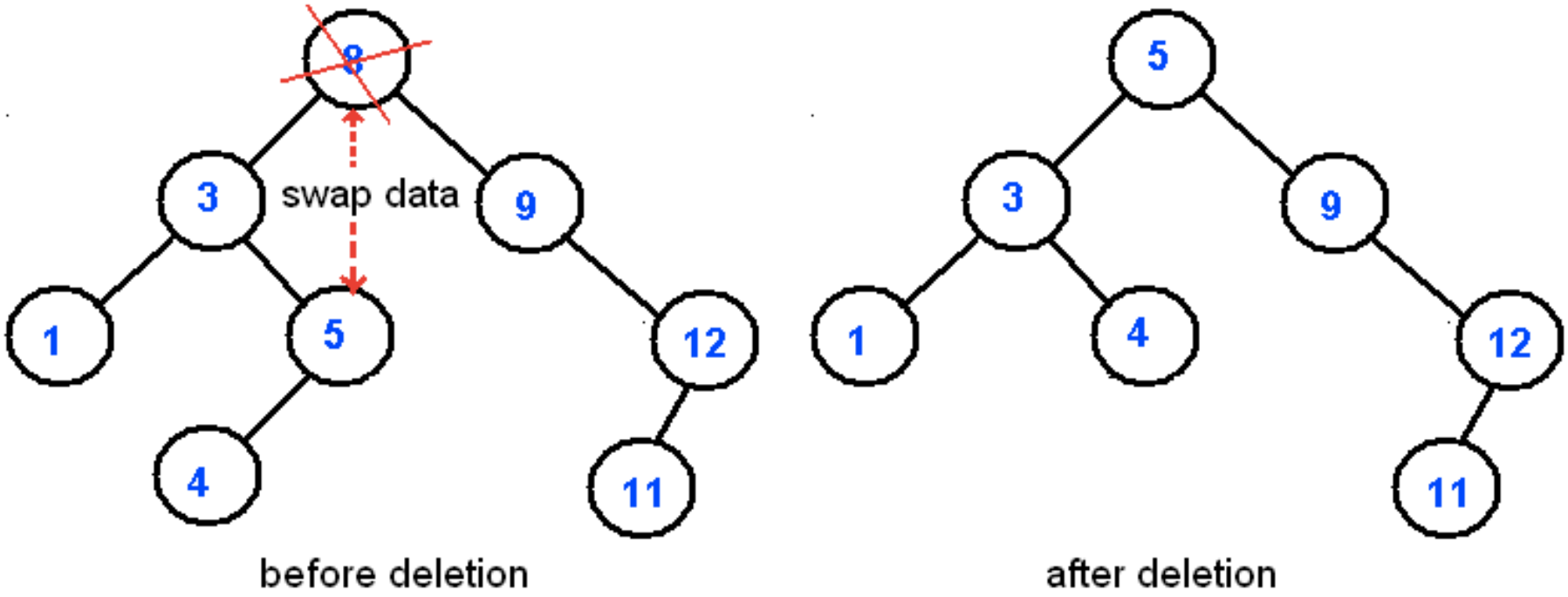


after deletion

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

BST

- Delete – has two children



- Replace the node being deleted with the largest node in the left subtree and then delete that largest node. By symmetry, the node being deleted can be swapped with the smallest node is the right subtree.

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

Binary Search Trees

OPERATION	WORST-CASE TIME	AVERAGE-CASE TIME
search	O(h) which can be O(n)	O(log n)
insert	O(h) which can be O(n)	O(log n)
remove	O(h) which can be O(n)	O(log n)
traverse	O(h) which can be O(n)	O(n)

QUESTIONS TO PONDER

Questions

1. In what case would you insert a node in a BST by adding it in between a parent and child node?
2. Given a sequence of numbers:

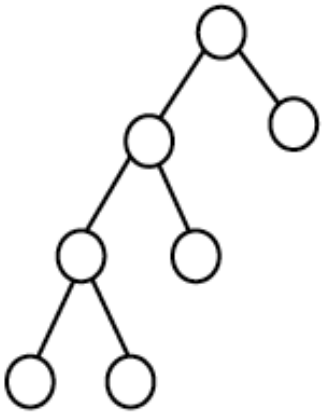
   11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

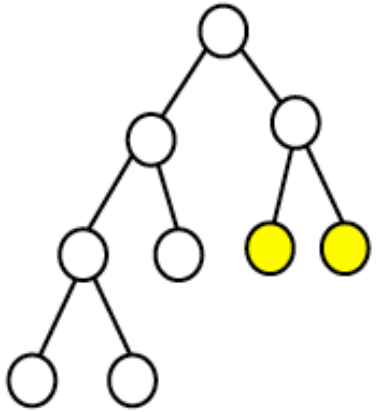
   Draw a binary search tree by inserting the above numbers from left to right.

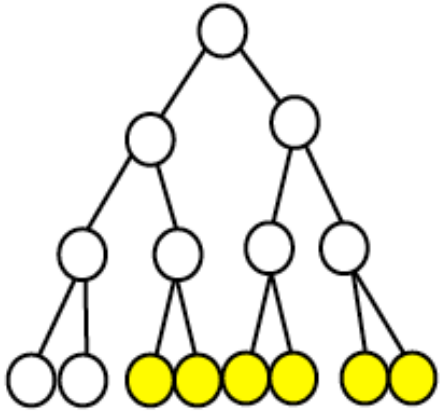
3. Now delete node 11 from the previous tree. What two trees can be created from this deletion?

Questions

1. How would find the max value in a BST?
2. Match the following trees to the type: complete, full, perfect

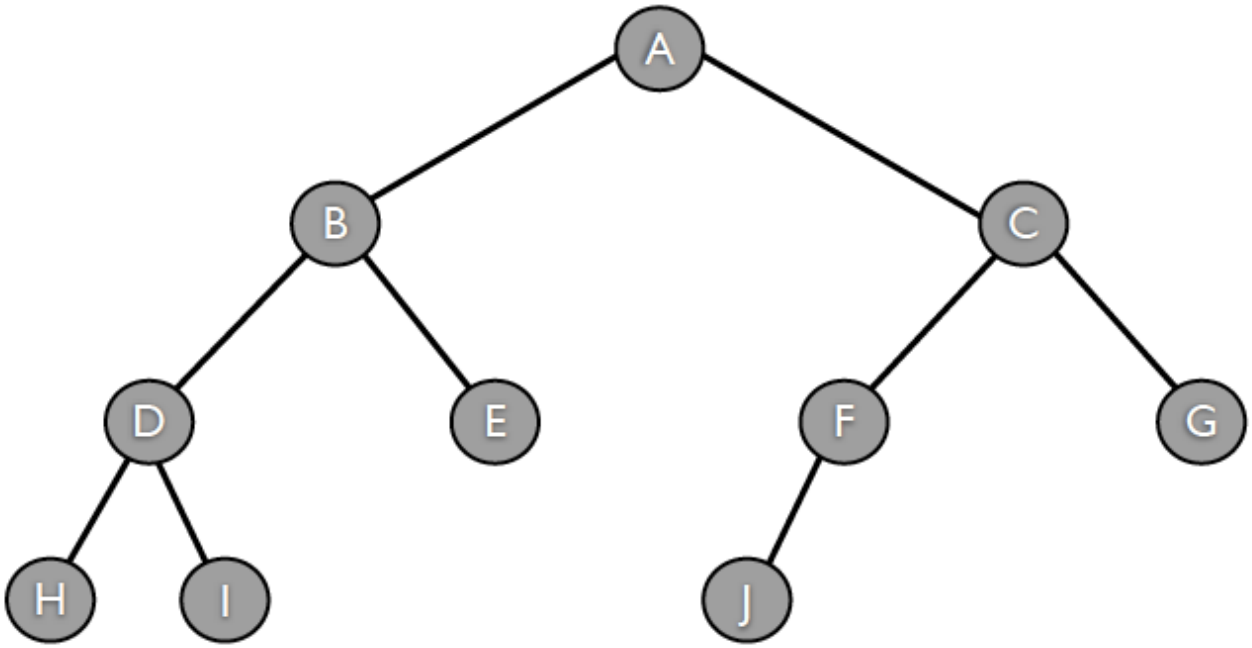






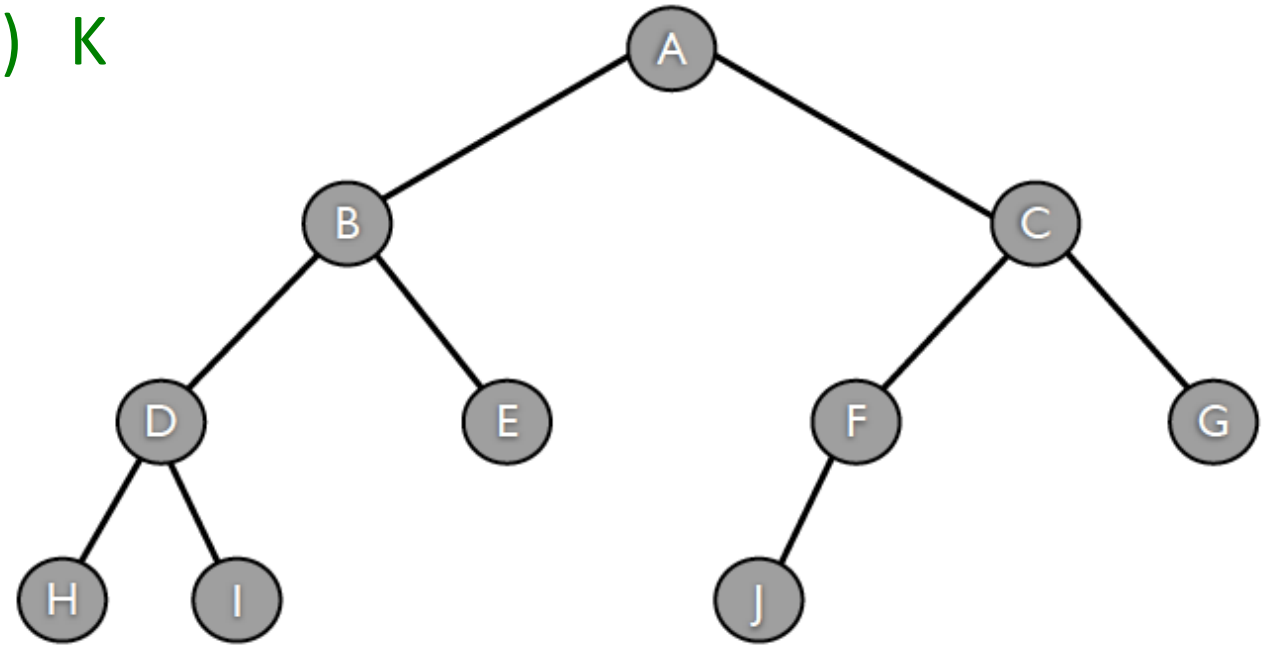
Questions

1. Traverse the following binary tree
    1. inorder
    2. preorder
    3. postorder



Questions

1. Where would you add...
   a) C
   b) H
   c) F
   d) K



Questions

1. Where would you find the max value in a BST?
2. Where would you find the min value in a BST?
3. When would you know the BST does not contain a particular value and you can you stop traversing?