Data Structures and Algorithms

UBCO Master of Data Science – DATA 532

Recap…

We looked at the notion of data structures

- Hashing
- Collisions, Linear Probing, etc.

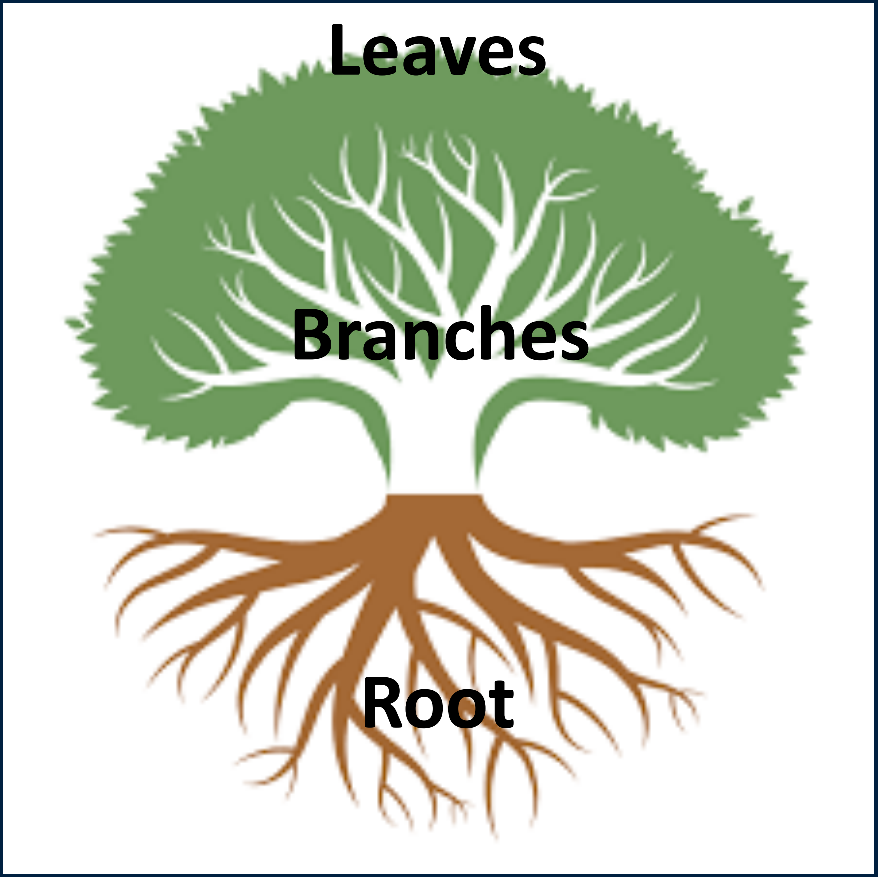
We learnt some basic algorithms

- Reviewed algorithmic complexity

What are we going to learn today ?

- What is a tree data structure
- Traversal in trees (pre-order, post-order)
- Idea of Binary Trees
- Concept of Binary Search Trees and how it affects searching

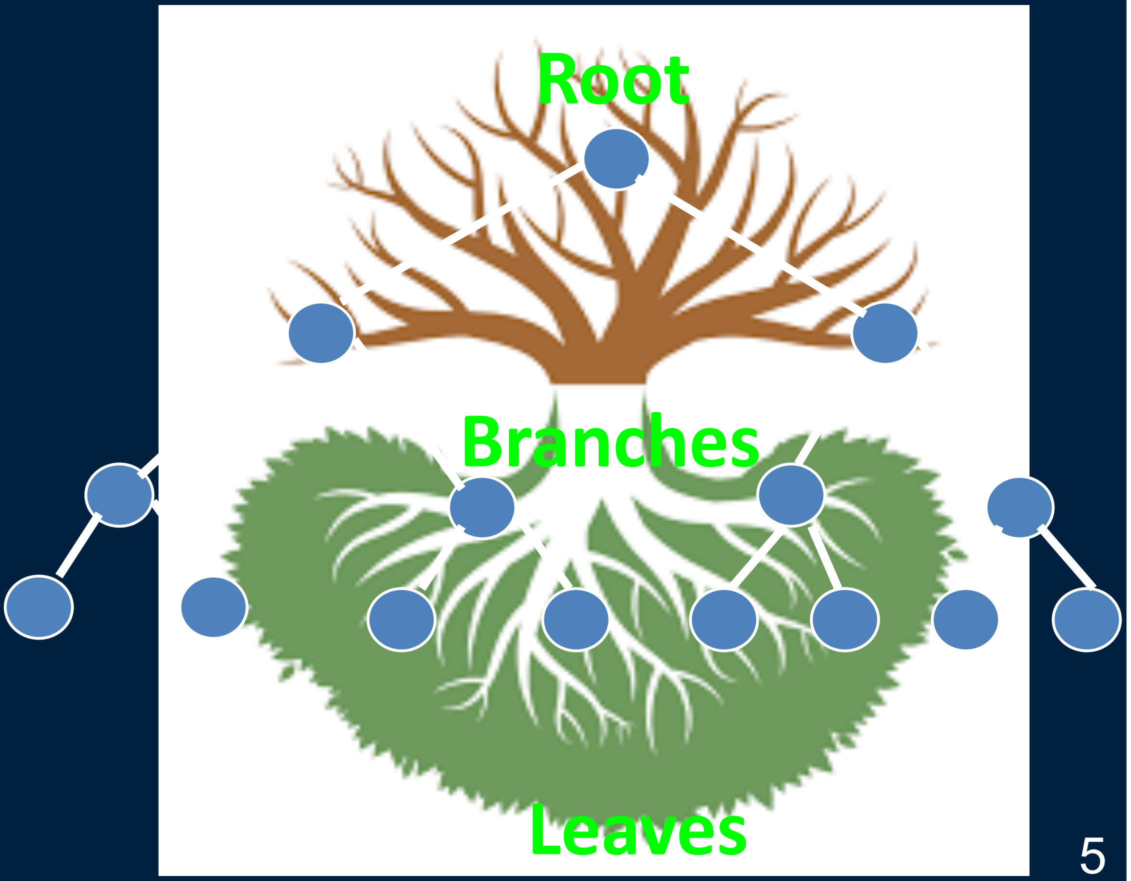
Natural View of Trees….



Data Structure View of Trees…

While linked lists, stacks, and queues are useful data structures, they do have limitations

- Linked lists are linear in form and cannot reflect hierarchically organized data
- Stacks and queues are one-dimensional structures and have limited expressiveness



What is a Tree ?

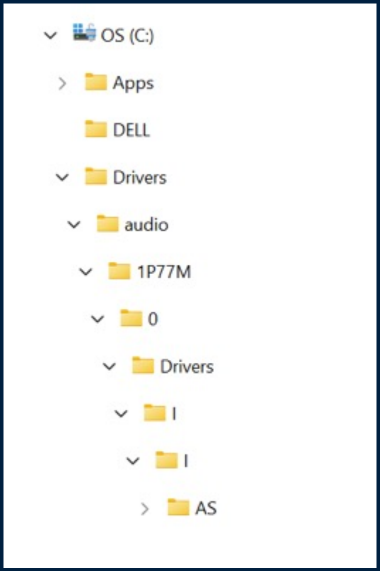
A tree is a finite nonempty set of elements.

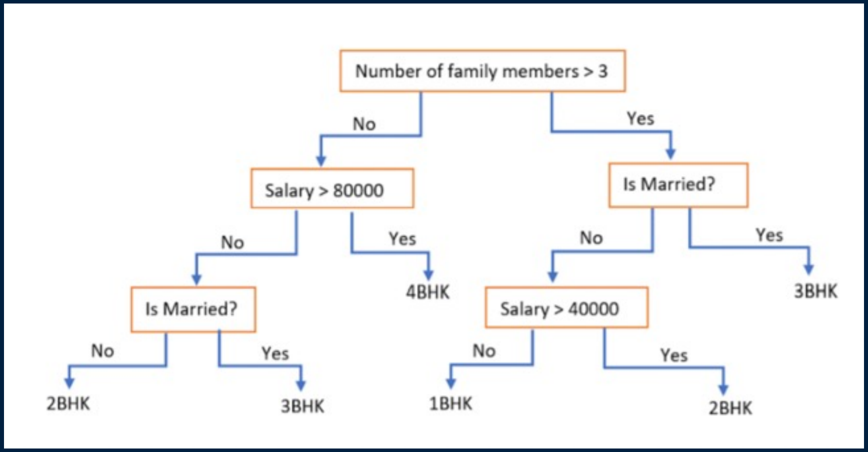
It is an abstract model of a hierarchical structure.

consists of nodes with a parent-child relation.

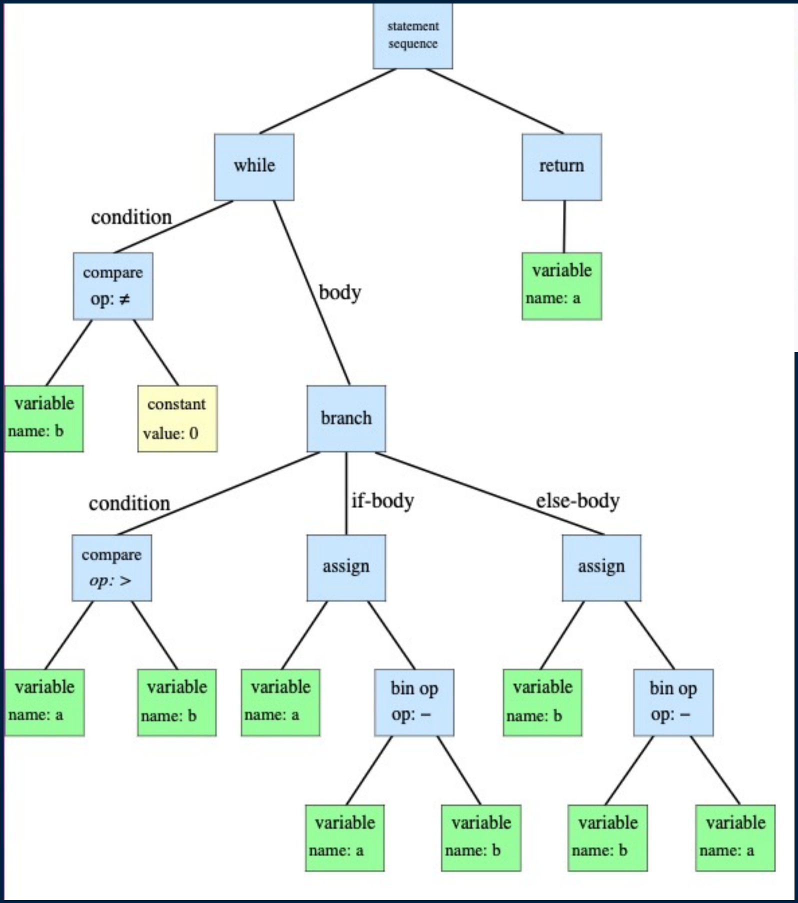
Applications:

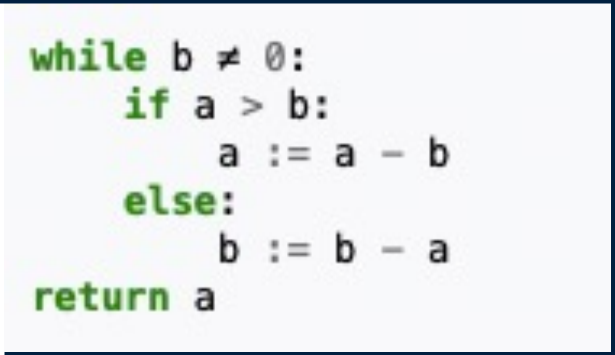
- File systems
- Decision Trees



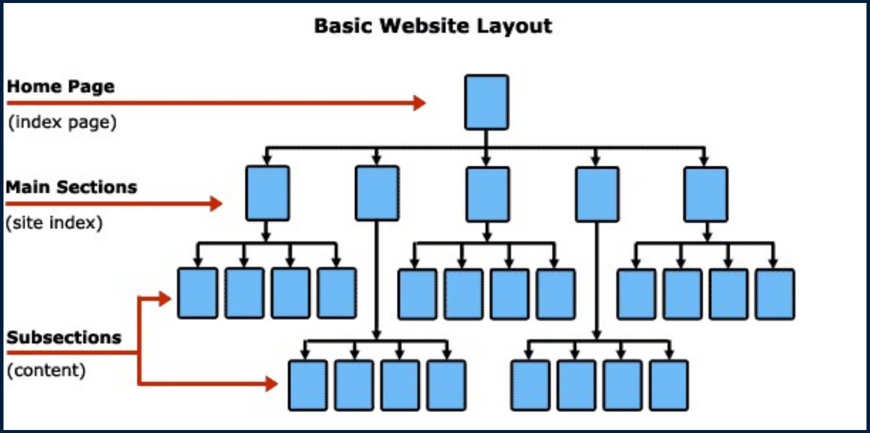


Tree Data Structure applications





```
while b ≠ 0:
    if a > b:
        a := a − b
    else:
        b := b − a
return a
```



Tree Terminology

Root: node without parent (A)

Siblings: nodes share the same parent

Internal node: node with at least one child (A, B, C, F)

External node (**LEAF**): node without children (E, I, J, K, G, H, D)

Ancestors of a node: parent, grandparent, grand-grandparent, etc.

Descendant of a node: child, grandchild, grand-grandchild, etc.

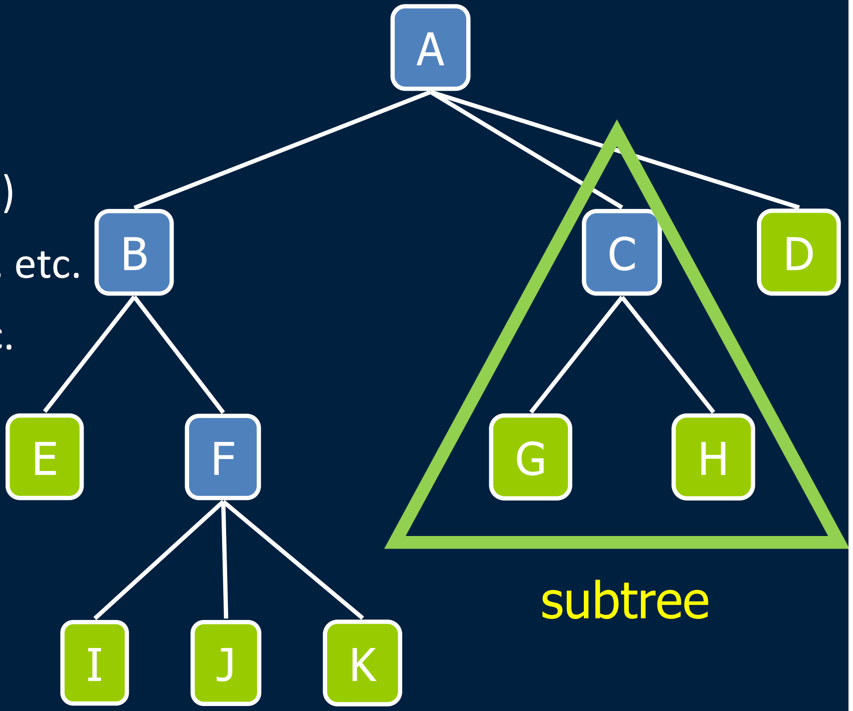
Depth of a node: number of ancestors

Height of a tree: maximum depth of a node (3)

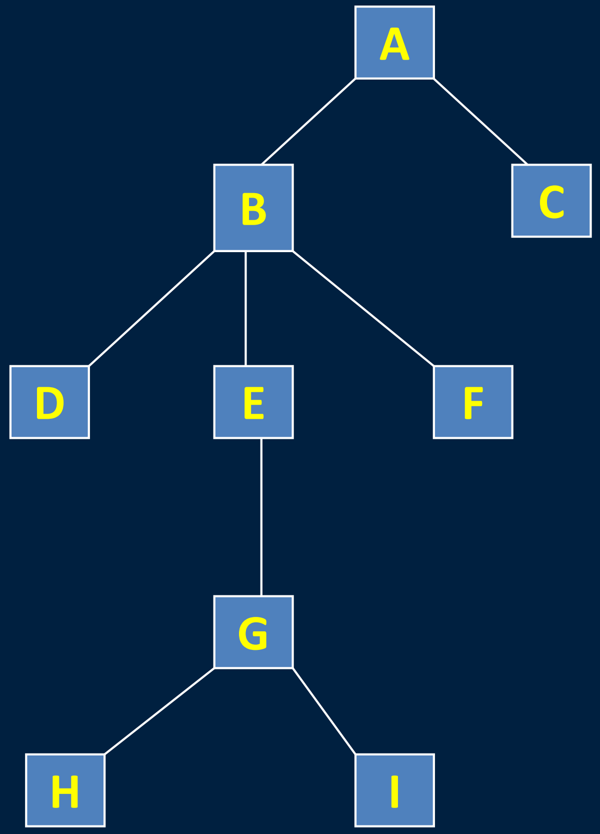
Degree of a node: the number of its children

Degree of a tree: the maximum number of its node.

Subtree: tree consisting of a node and its descendants



Quick check…



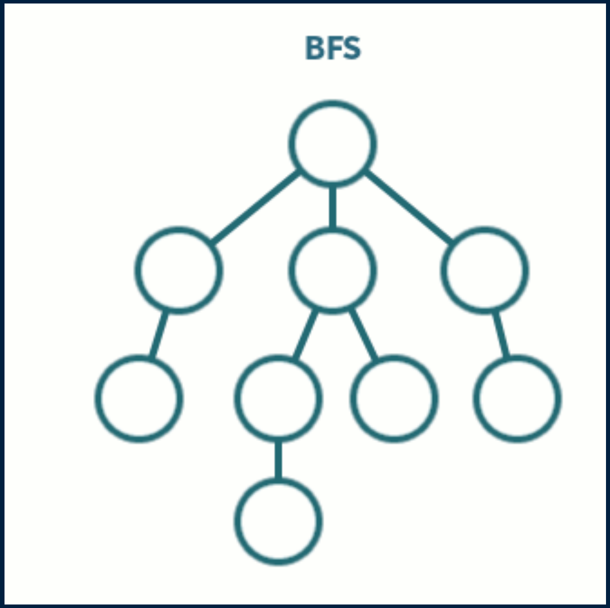
Property	Value
Number of nodes	
Height	
Root Node	
Leaves	
Interior nodes	
Ancestors of H	
Descendants of B	
Siblings of E	
Right subtree of A	
Degree of A	
Degree of F	

Tree Traversal: Breadth First Traversal

Tree Traversal: visit each node to perform a certain operation (e.g. print the value stored in the node)

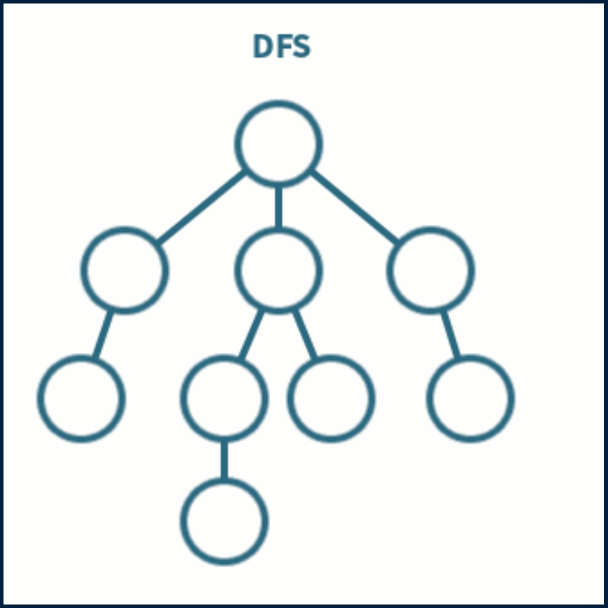
In Tree Traversal, all nodes must be visited and only once in a certain order

Breadth-First Traversal: We will go down the tree level by level. We will go left to right on each level looking at all sibling nodes



Tree Traversal: Depth-First Traversal

Deepth-First Traversal: We will go as far down the tree as possible. We will go left to right on each level. Once it hits the last leave, it starts back up at the top. Follows the same process.



Tree Traversal :Depth Traversal

Three main methods:

- Preorder
- Postorder
- Inorder

Preorder (Visit, Left, Right) :

- VLR
- visit the root
- traverse in preorder the left subtree
- Traverse in preorder the right subtree

Inorder (Left, Visit, Right)

- LVR

Postorder (Left, Right, Visit)

- LRV
- traverse the left subtree
- Visit the right subtree
- visit the root

V - visit the node

L - traverse the left subtree

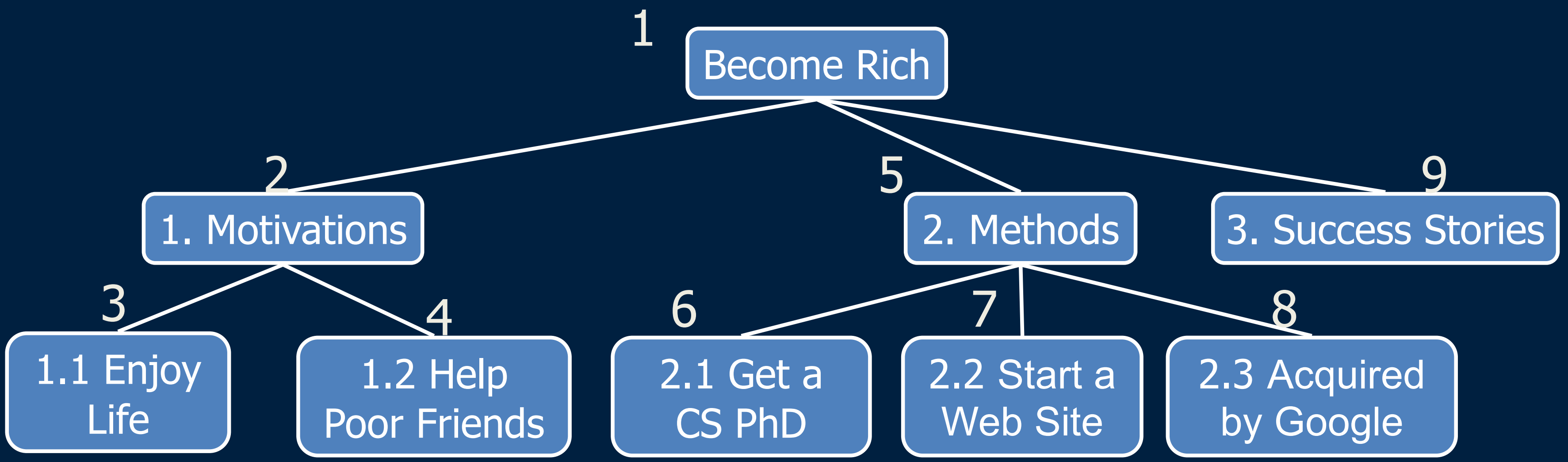
R - traverse the right subtree

Preorder Traversal…

In a preorder traversal, a node is visited before its descendants

Application: print a structured document

```
Algorithm preOrder(p)
    visit(p)
    preorder (p.left)
    preorder (p.right)
```

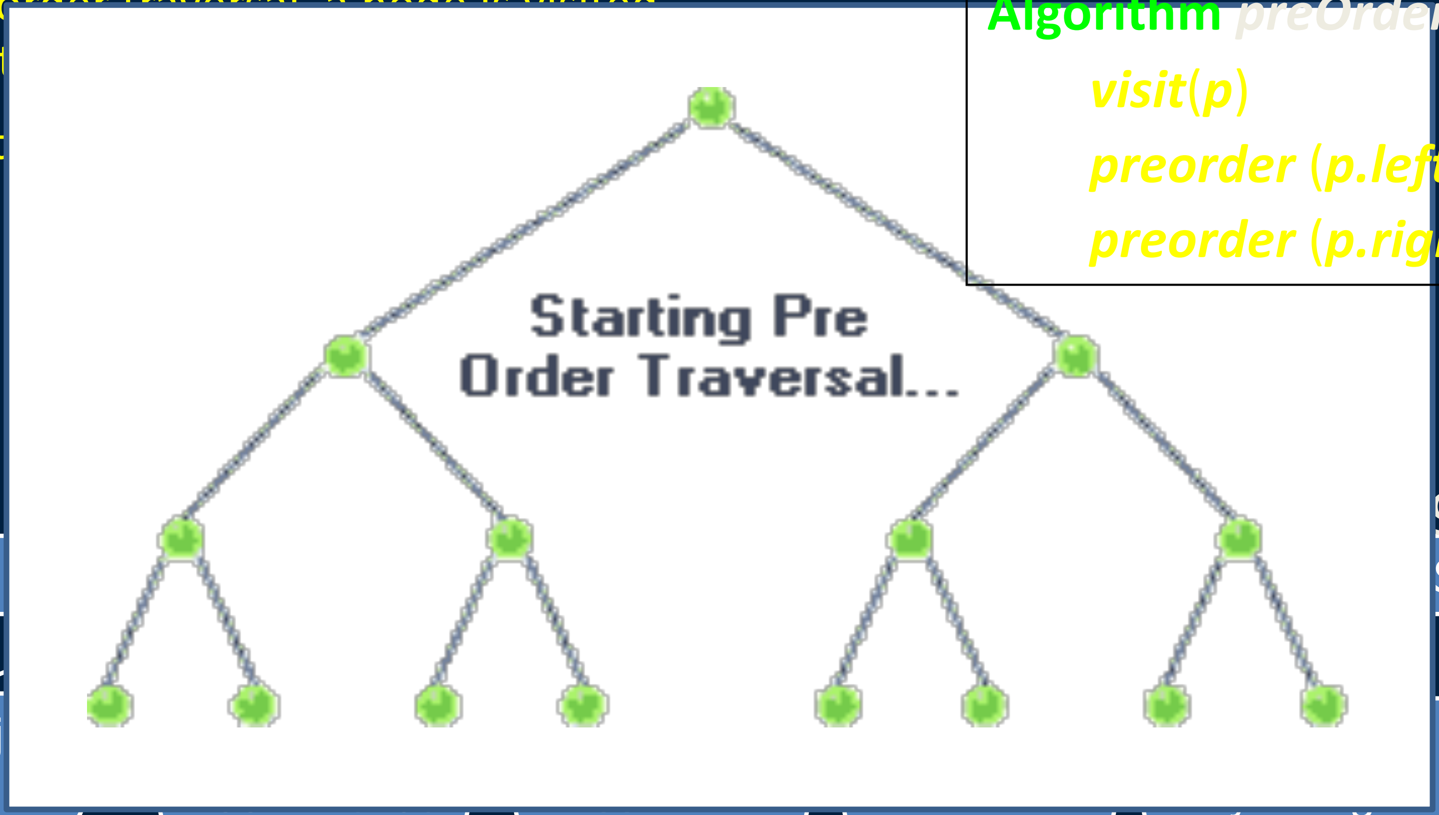


Preorder Traversal…

In a preorder traversal, a node is visited
before it

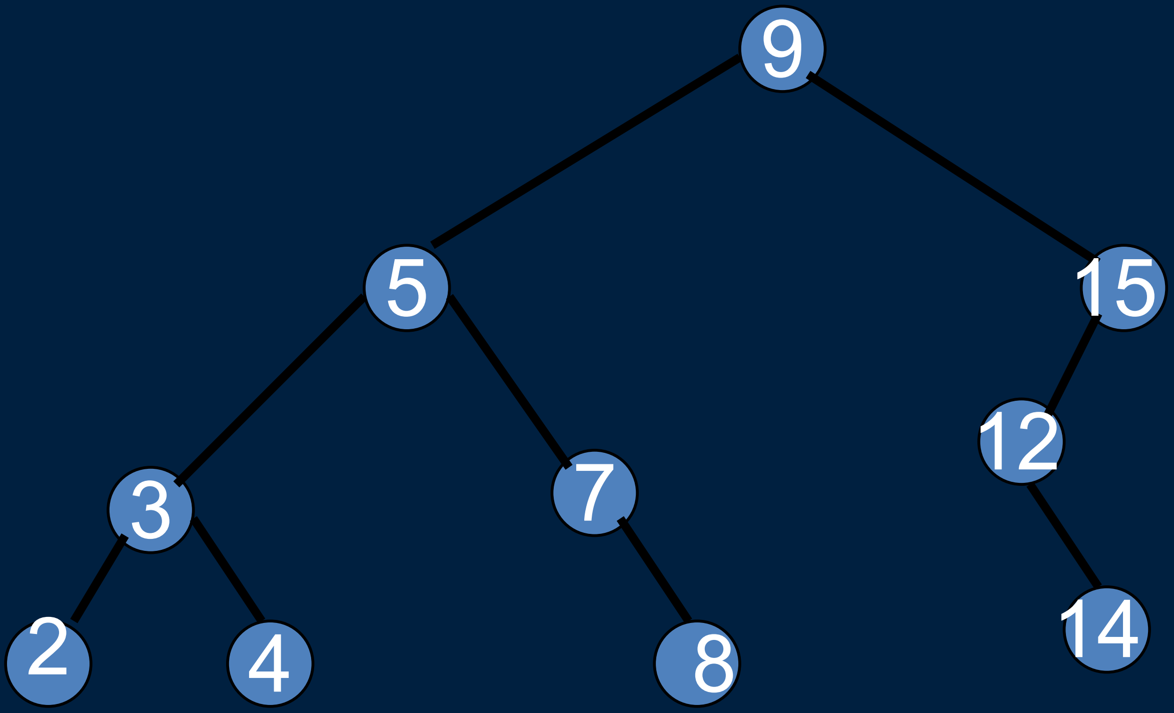
Applicat

```
Algorithm preOrder(p)
        visit(p)
        preorder (p.left)
        preorder (p.right)
```



9
Stories
3
1.1 Enj
Life

Preorder Example (VLR)

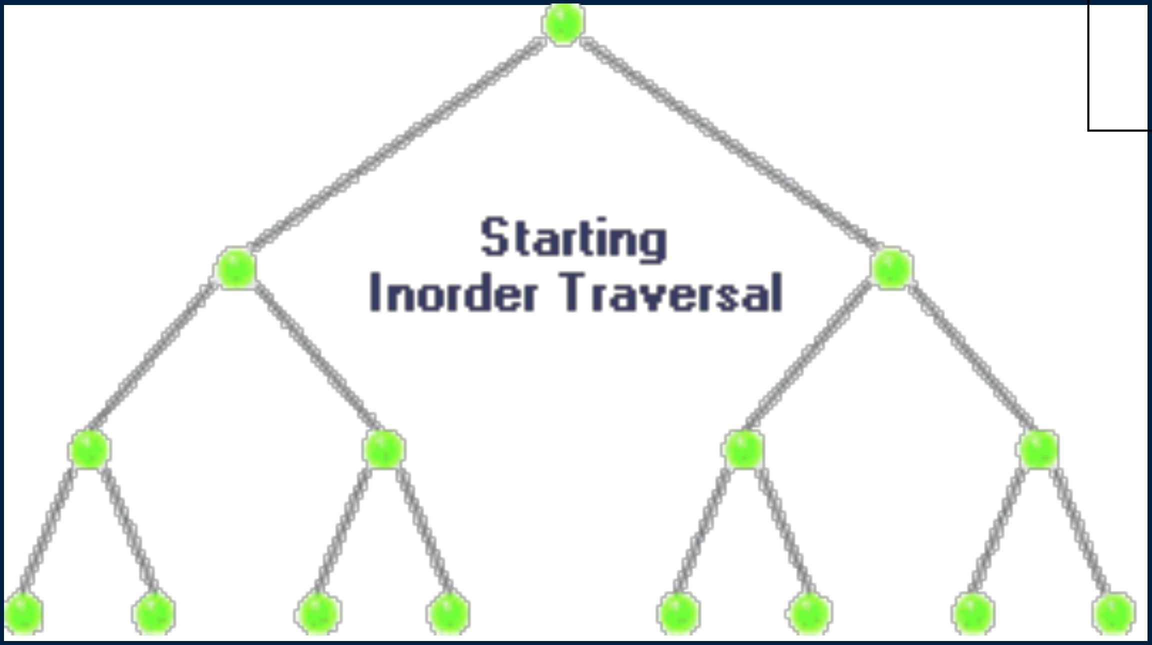


9 5 3 2 4 7 8 15 12 14

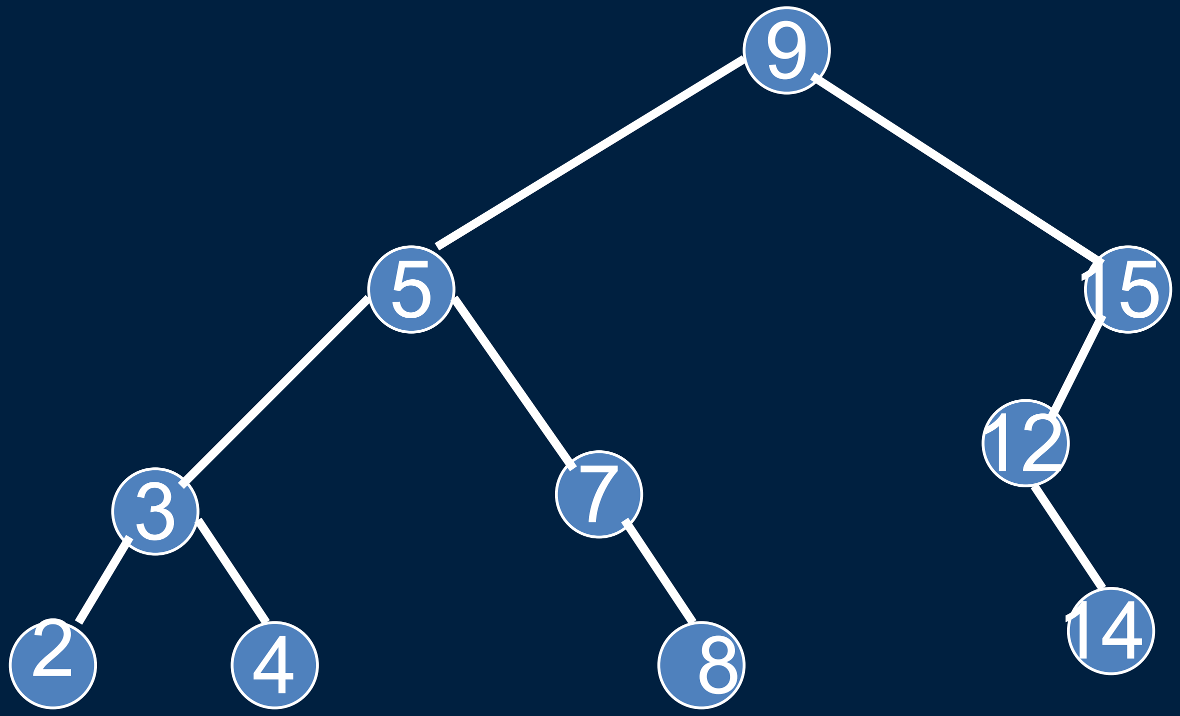
Inorder tree traversal

LVR

```
Algorithm inOrder(p)
    inOrder (p.left)
    visit(p)
    inOrder (p.right)
```



- Observe the output of the inorder traversal, what do you notice:
- It is ordered !
- This is not a coincidence, output in a BST (not any tree) using inorder is a sorted output



2 3 4 5 7 8 9 12 14 15

Postorder Traversal

In a posto

its descer

Applicati

directory

```
Algorithm postOrder(p)
        postOrder (p.left)
        PostOrder (p.right)
        visit(p)
```

1

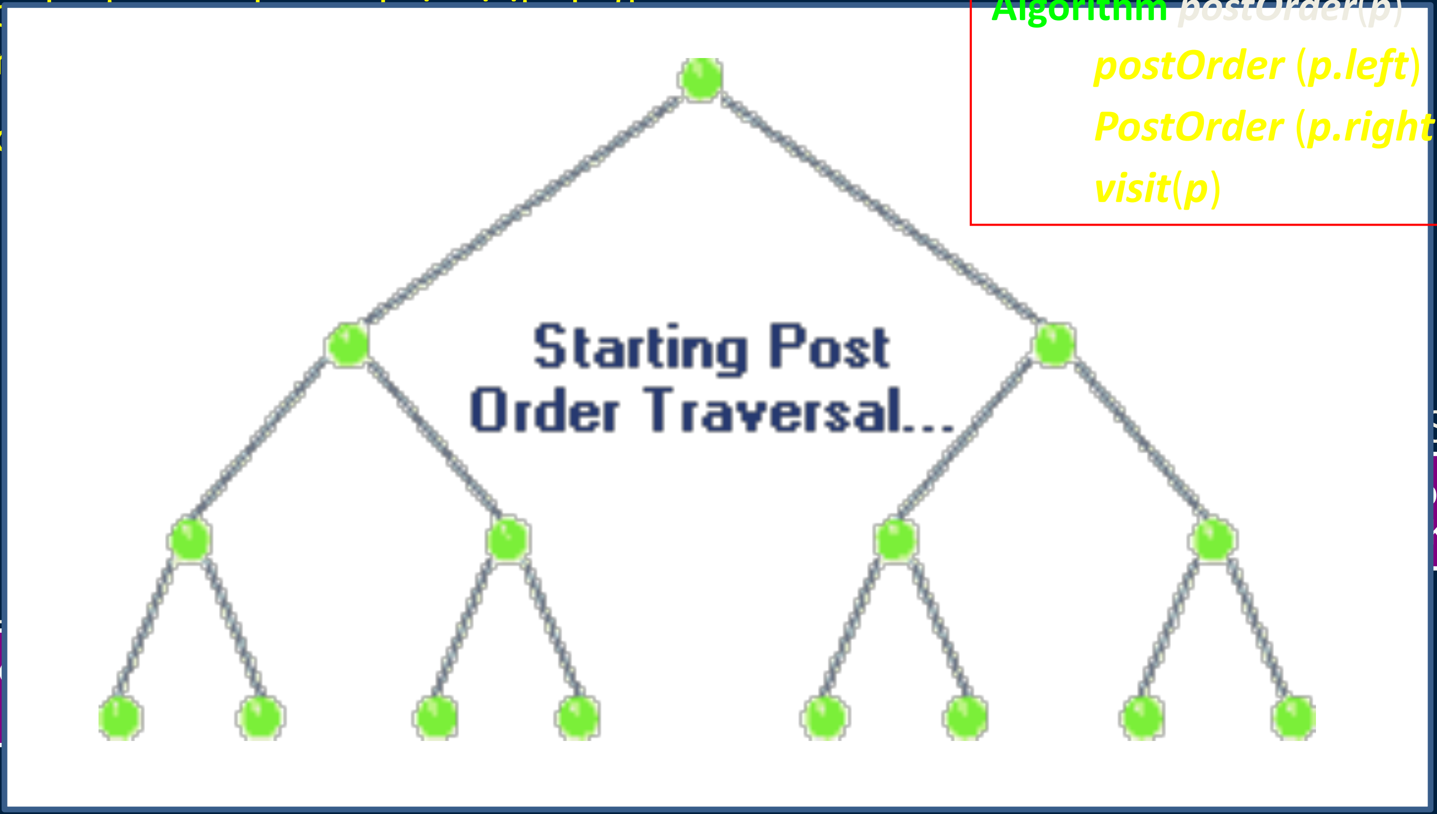
h1c.

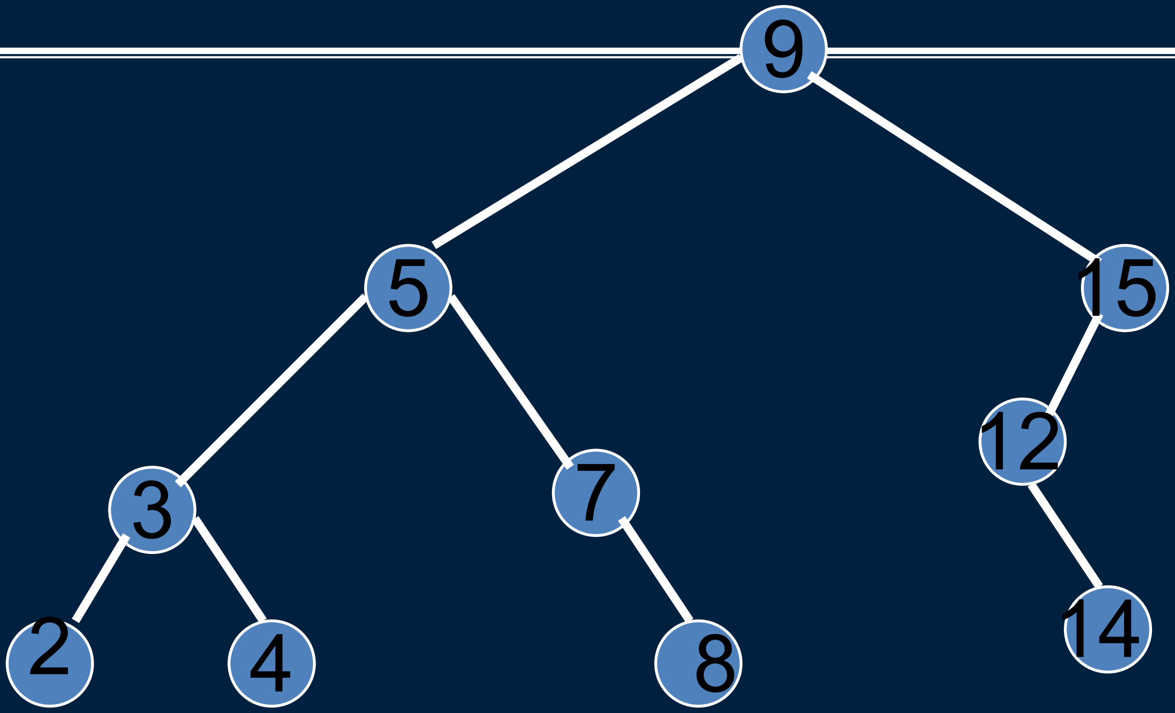
3

3

.txt

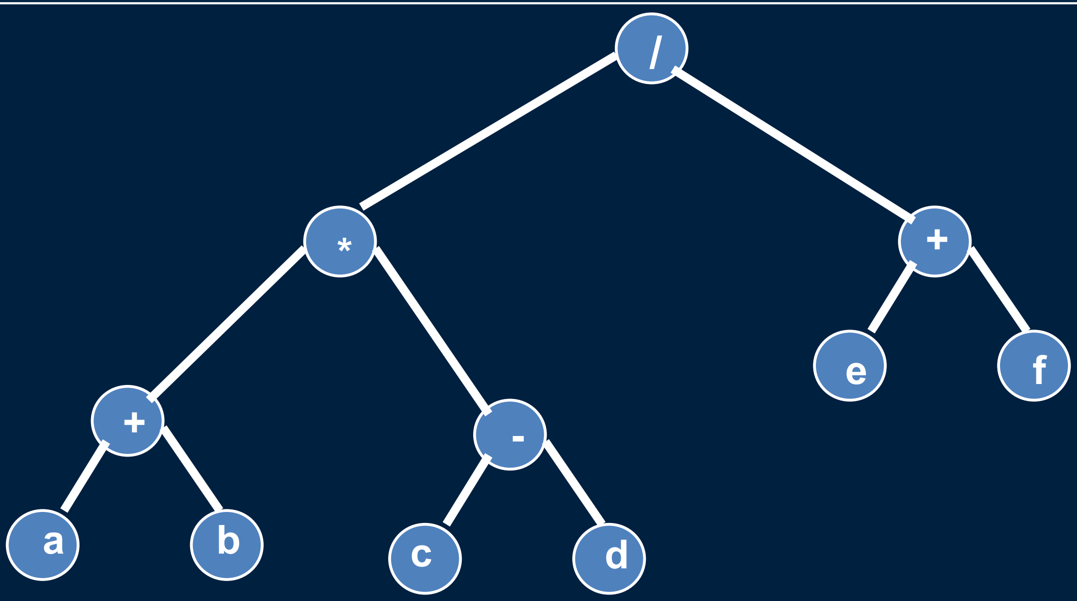
K





2 4 3 8 7 5 14 12 15 9

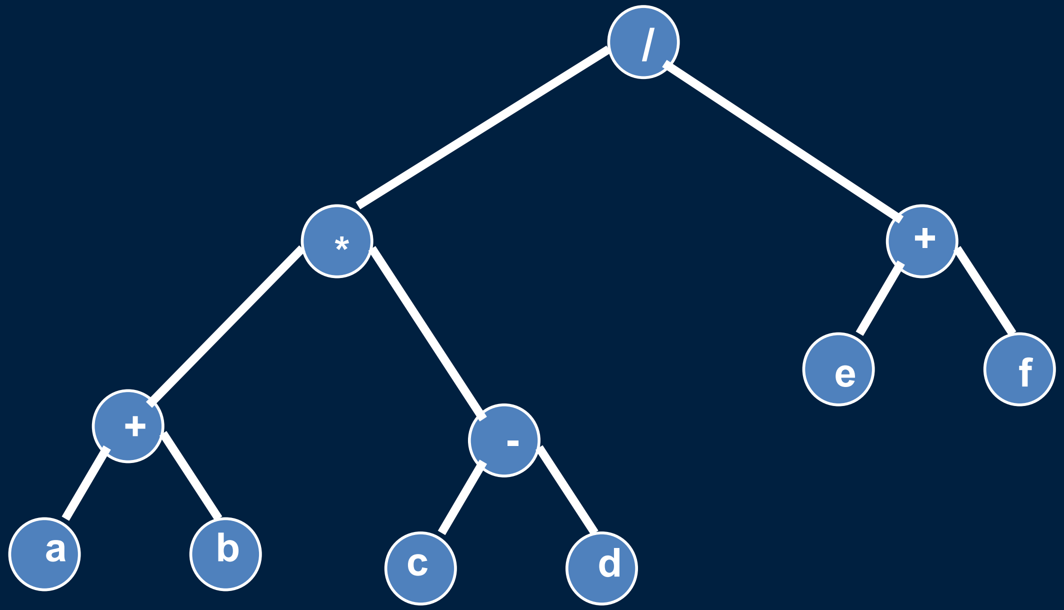
- Example2: traverse using postorder traversal



a b + c d - * e f + /

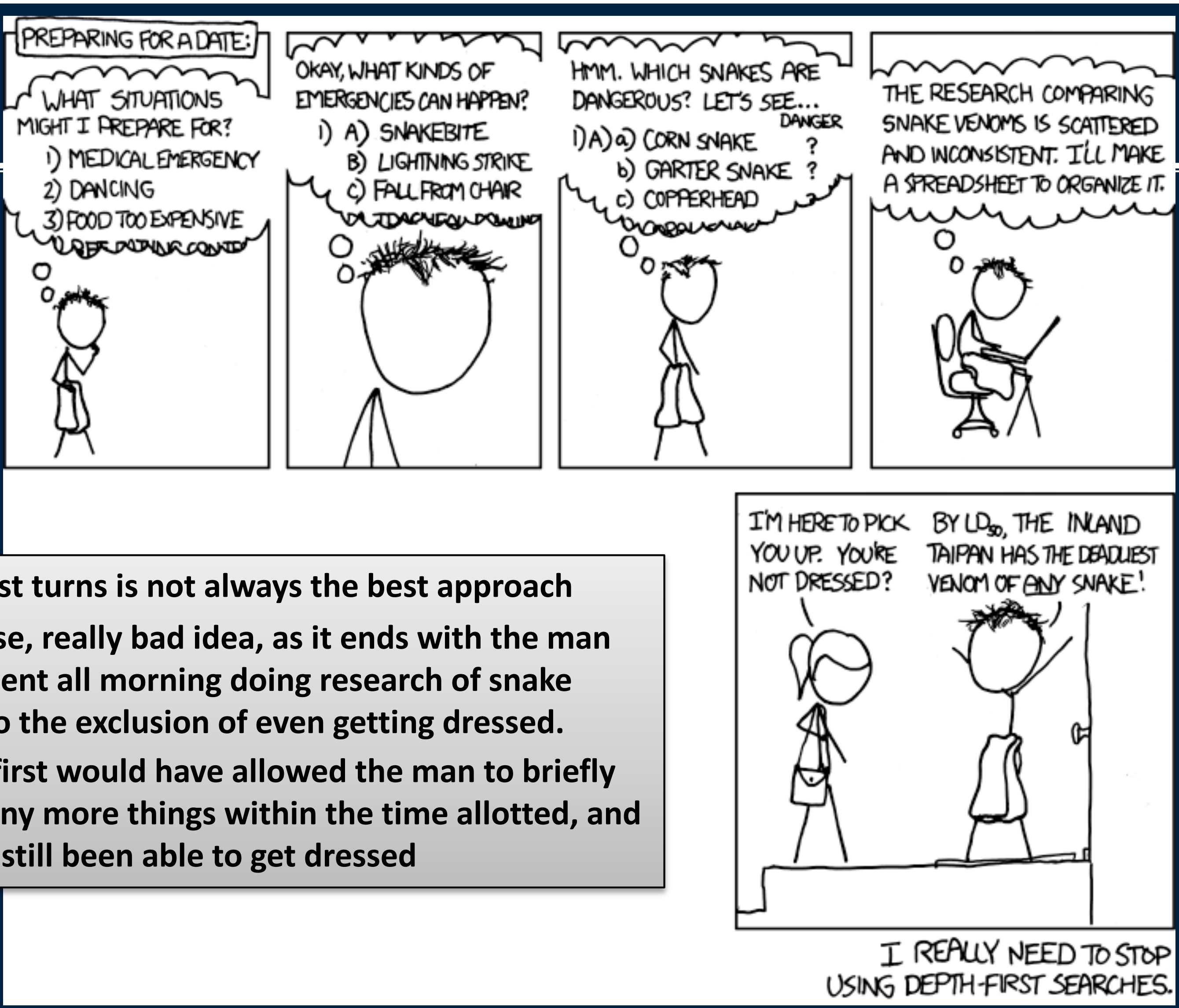
Gives postfix form of expression **of (a+b) * (c-d) / (e+f)**

Example 2: Traverse using preorder traversal



/ * + a b - c d + e f

Gives prefix form of expression of (a+b) * (c-d) / (e+f)


PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR? 1) MEDICAL EMERGENCY 2) DANCING 3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN? 1) A) SNAKEBITE B) LIGHTNING STRIKE C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE... DANGER 1) A) a) CORN SNAKE ? b) GARTER SNAKE ? c) COPPERHEAD
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD50, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

- Depth First turns is not always the best approach
- In this case, really bad idea, as it ends with the man having spent all morning doing research of snake venom, to the exclusion of even getting dressed.
- Breadth-first would have allowed the man to briefly check many more things within the time allotted, and probably still been able to get dressed

Binary Tree

A binary tree is a tree with the following properties:

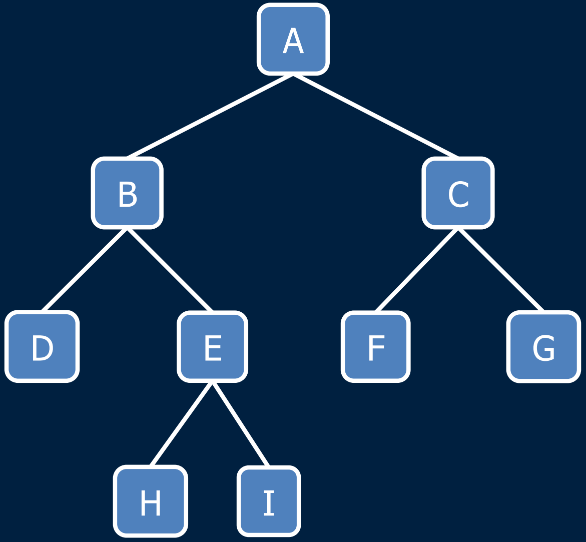
- Each internal node has at most two children (degree of two)
- The children of a node are an ordered pair
  - We call the children of an internal node left child and right child

Alternative recursive definition: a binary tree is either

- a tree consisting of a single node, OR
- a tree whose root has an ordered pair of children, each of which is a binary tree

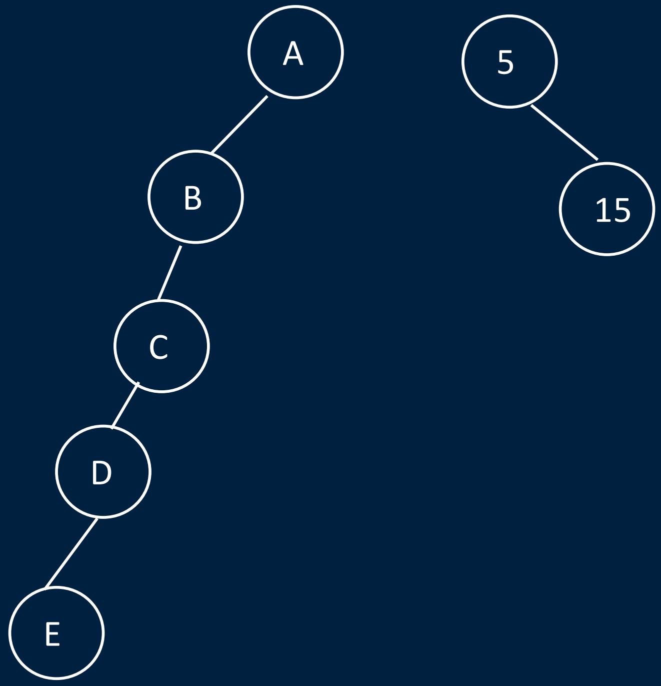
Applications:

- arithmetic expressions (see examples in previous slides)
- decision processes
- searching

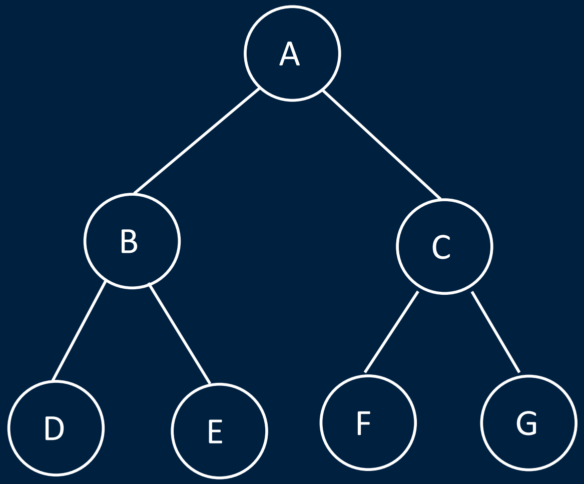


Other extreme examples of binary trees

Skewed Binary Tree

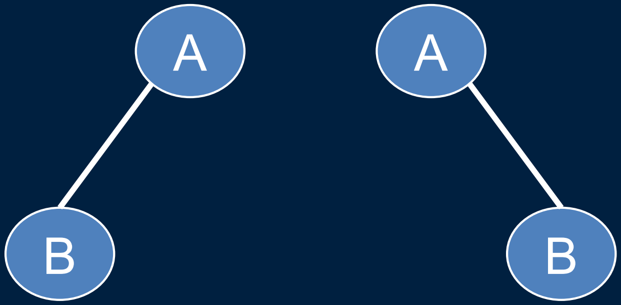


Complete Binary Tree



Differences Between A Tree and A Binary Tree

The subtrees of a binary tree are ordered; those of a tree are not ordered.



- Are different when viewed as binary trees.
- Are the same when viewed as trees.

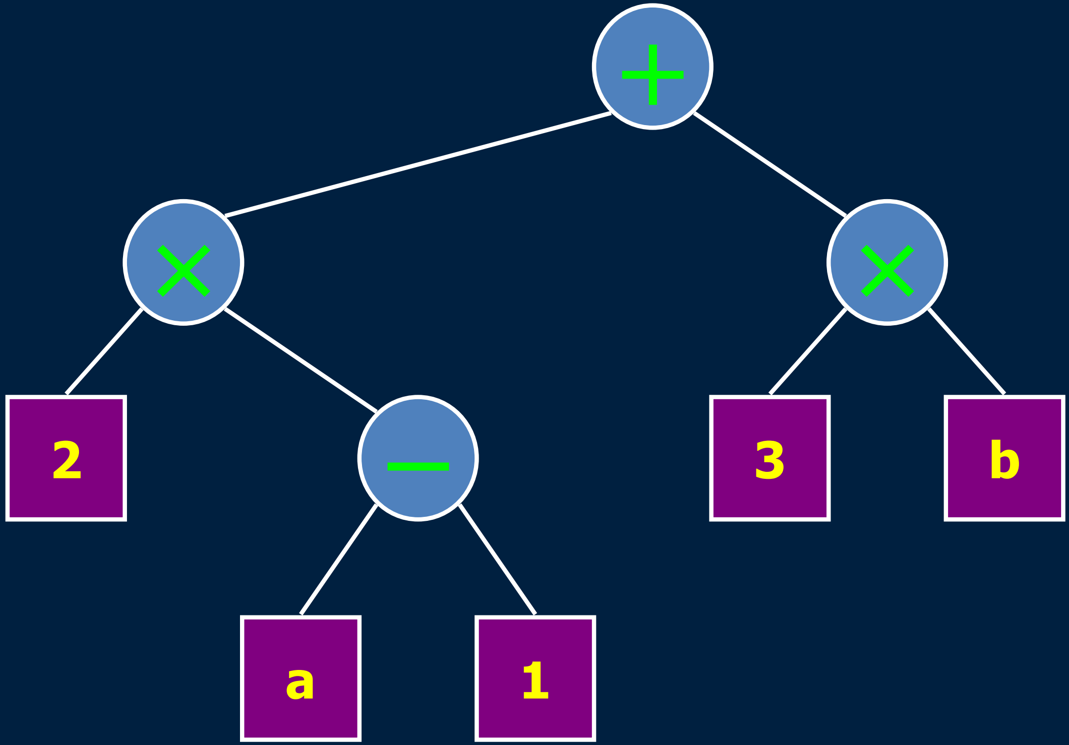
Do a inorder traversal and identify the expression?

A) $2 \times a - 1 + 3 \times b$

B) $((2 \times a) - 1) + (3 \times b)$

C) $(2 \times (a - 1) + b \times 3)$

D) None of the above

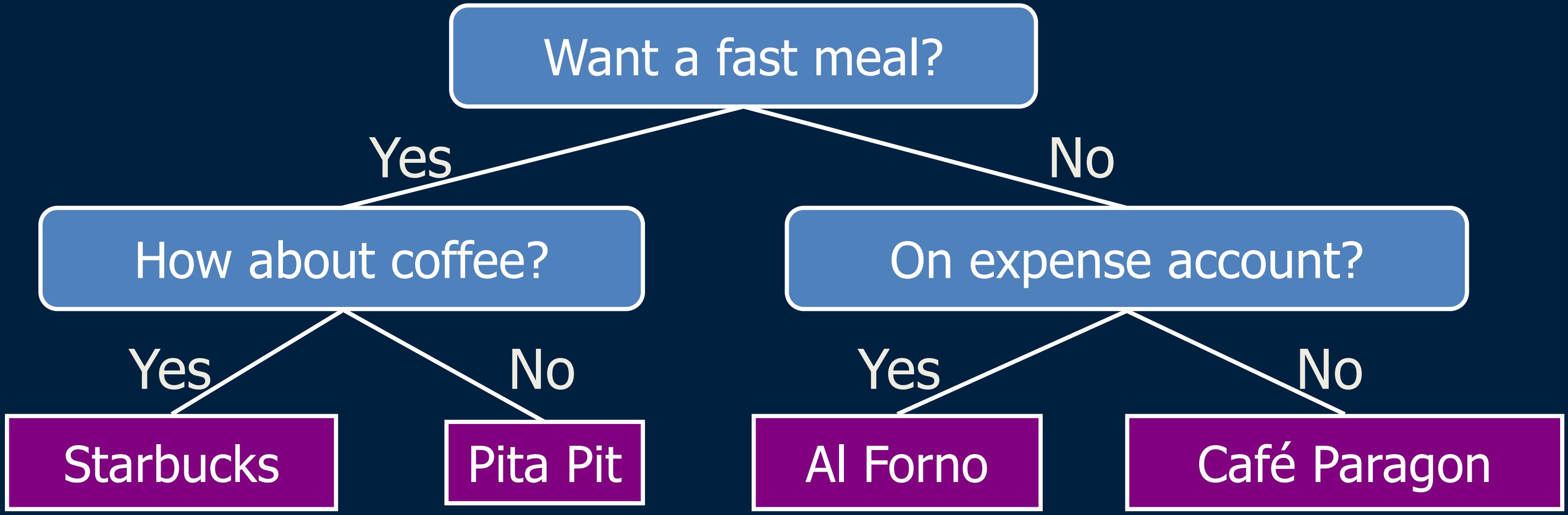


Binary Tree as a Decision Making Tool

Binary tree associated with a decision process

- internal nodes: questions with yes/no answer
- external nodes: decisions

Example: dining decision

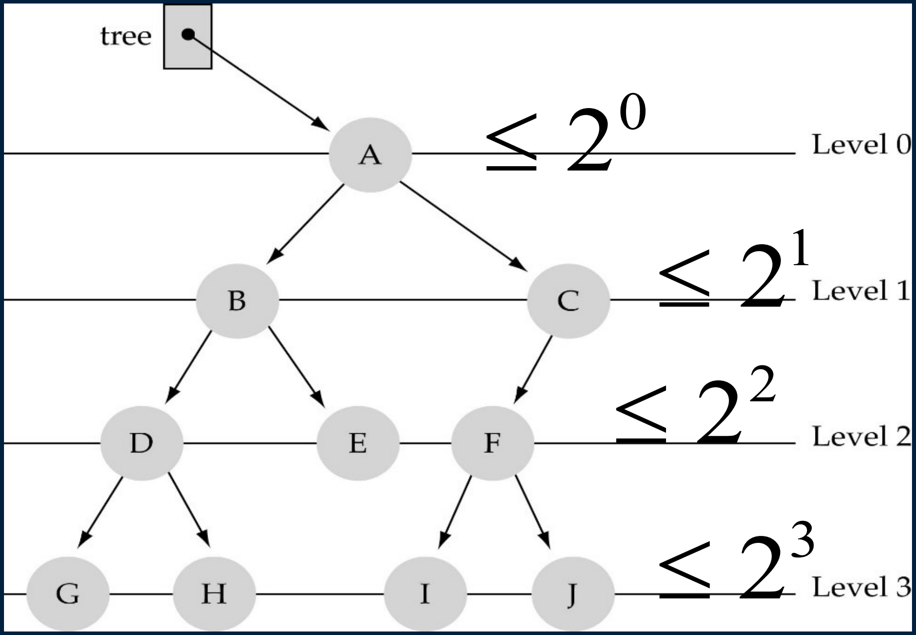


Some numbers…

- The maximum number of nodes on depth i of a binary tree is 2^i, $i >= 0$.
- The maximum number of nodes in a binary tree of height h is $2^{h+1}-1$, $h >= 0$.

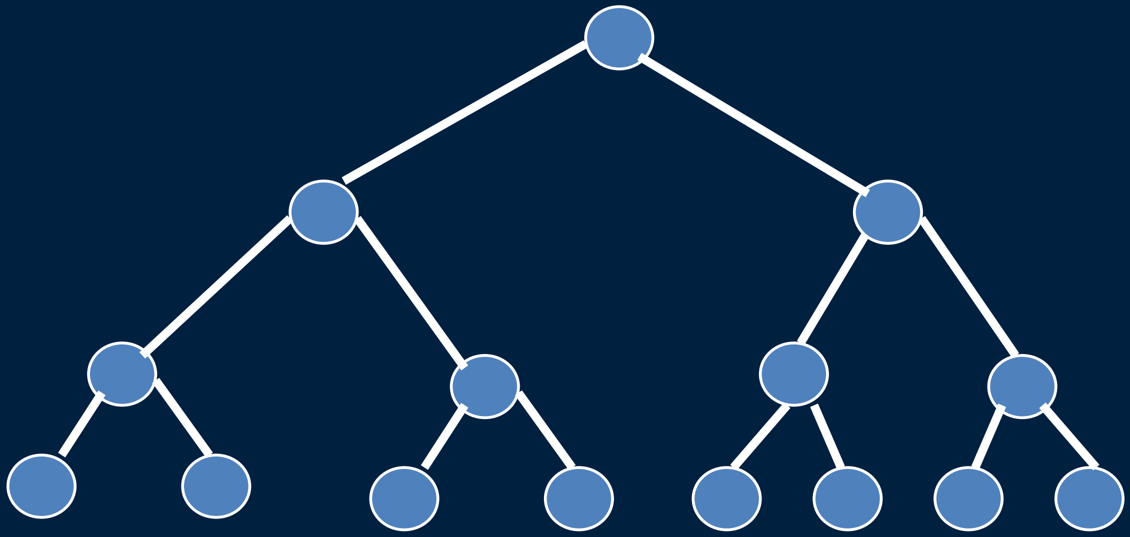
What is the max #nodes at some level *l*?

The max #nodes at level l is 2^l where $l=0,1,2,\ldots,L-1$



Complete (Full) Binary Tree

A complete binary tree of a given height h has exactly $2^{h+1}-1$ nodes.



Height 3 full binary tree.

What is the height h of a full tree with N nodes?

$$2^h - 1 = N$$

Why is h important?

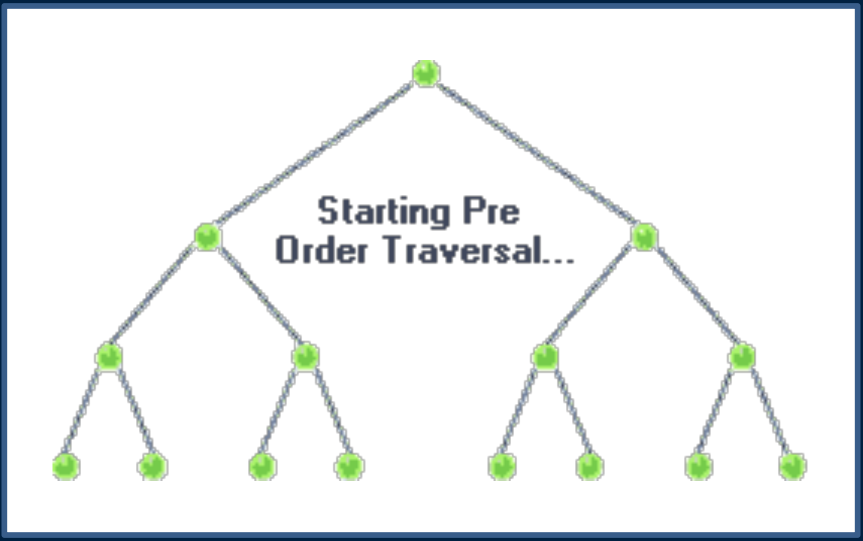
Tree operations (e.g., insert, delete, retrieve etc.) are typically expressed in terms of h.

So, h determines running time!

How to search a binary tree?

(1) Start at the root

(2) Search the tree level by level, until you find the element you are searching for or you reach a leaf.



Is this better than searching in a list ?

$\boxed{\text{No} \rightarrow O(N)}$

Binary Search Trees (BSTs)

Binary Search Tree Property:

The value stored at

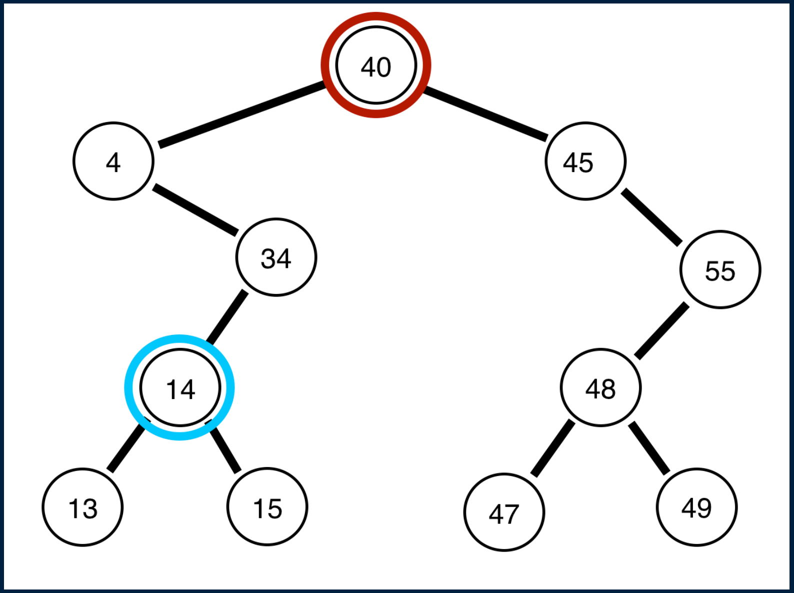
a node is *greater* than

the value stored at its

left subtree and *less* than

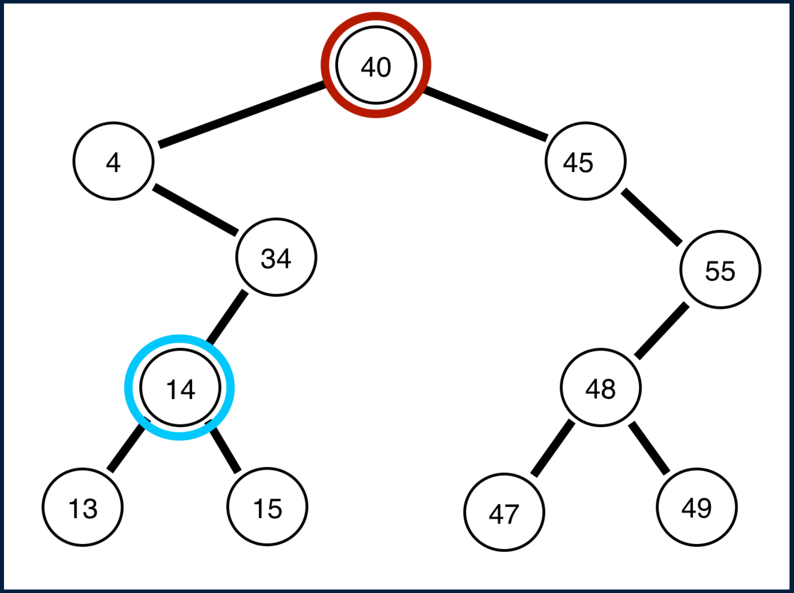
the value stored at its

right subtree



Binary **Search** Trees (BSTs)

In a BST, the value stored at the root of a subtree is *greater* than any value in its left subtree and *less* than any value in its right subtree!



Where is the smallest element stored in a BST ?

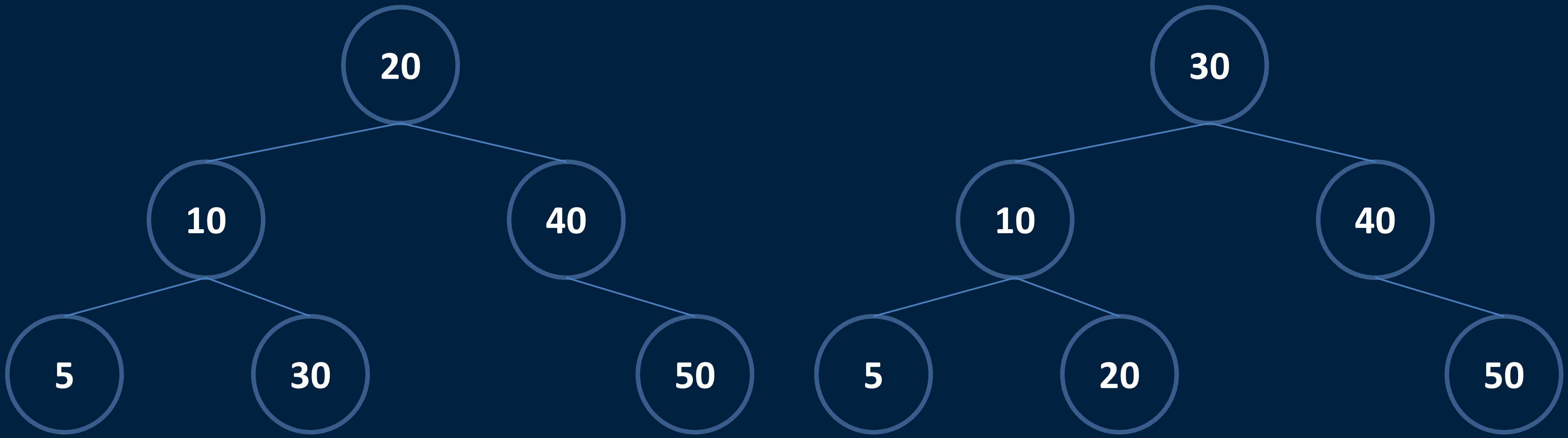
A) Root node

B) Left most node

C) Right most node

D) Any of the leaf nodes could be the smallest

Which of the following is BST?



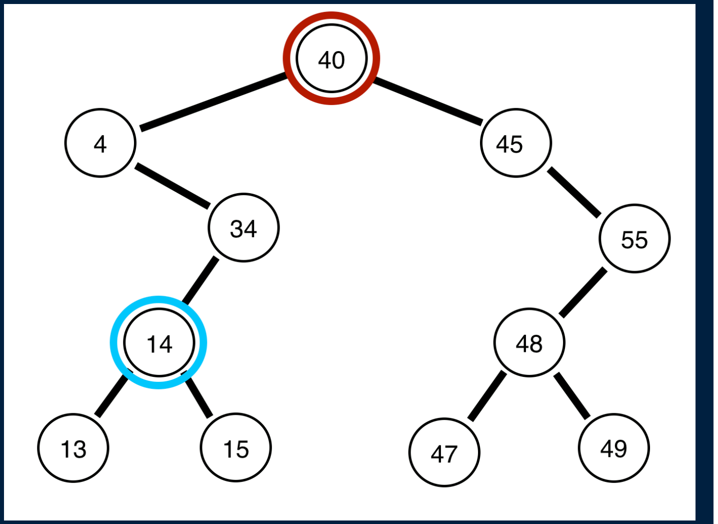
- The tree on the left is not a BST. Although every node has the appropriate relationship to its parent, a BST requires that the root node's value be greater than all values in the left sub-tree. The 30 node is out of place with respect to the 20.
- The tree on the right is a BST.

How to search a binary search tree?

(1) Start at the root

(2) Compare the value of the item you are searching for with the value stored at the root

(3) If the values are equal, then *item found*; otherwise, if it is a leaf node, then *not found*



Insertion

Searching a binary tree does not modify the tree

Traversals may temporarily modify the tree, but it is usually left in its original form when the traversal is done

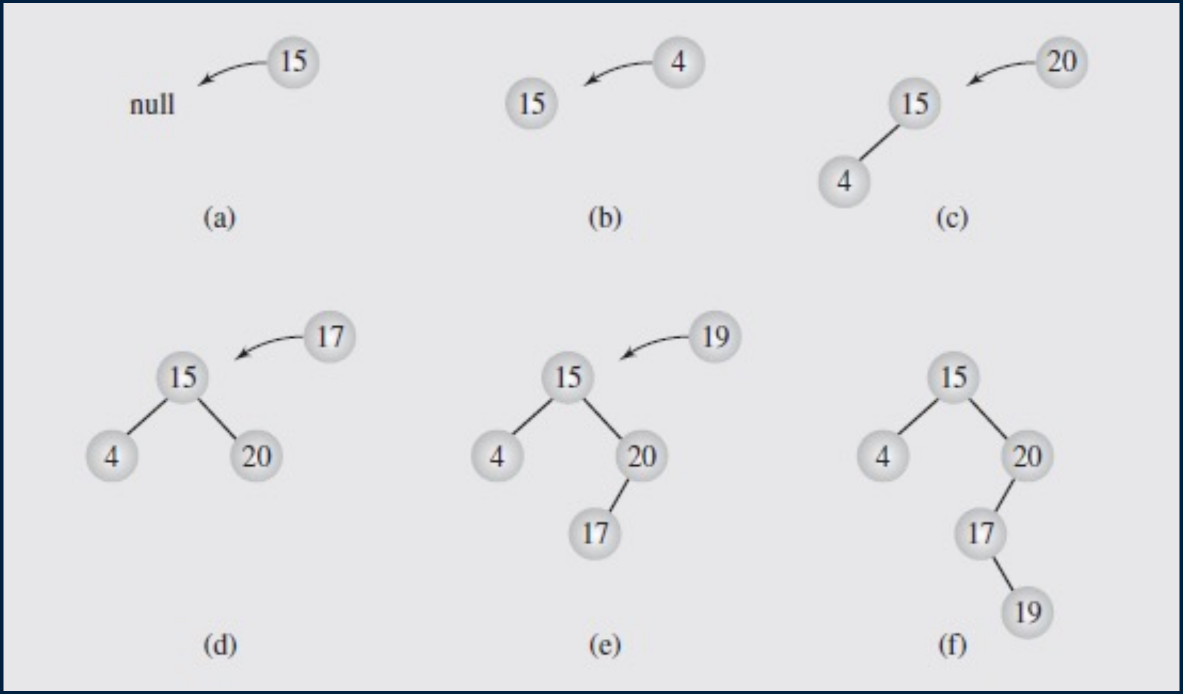
Operations like insertions, deletions, modifying values, merging trees, and balancing trees do alter the tree structure

We'll look at how insertions are managed in binary search trees

In order to insert a new node in a binary tree, we have to be at a node with a vacant left or right child

This is performed in the same way as searching:

- Compare the value of the node to be inserted to the current node
- If the value to be inserted is smaller, follow the left subtree; if it is larger, follow the right subtree
- If the branch we are to follow is empty, we stop the search and insert the new node as that child


15
null
(a)
4
15
(b)
20
15
4
(c)
17
15
4
20
(d)
19
15
4
20
17
(e)
15
4
20
17
19
(f)

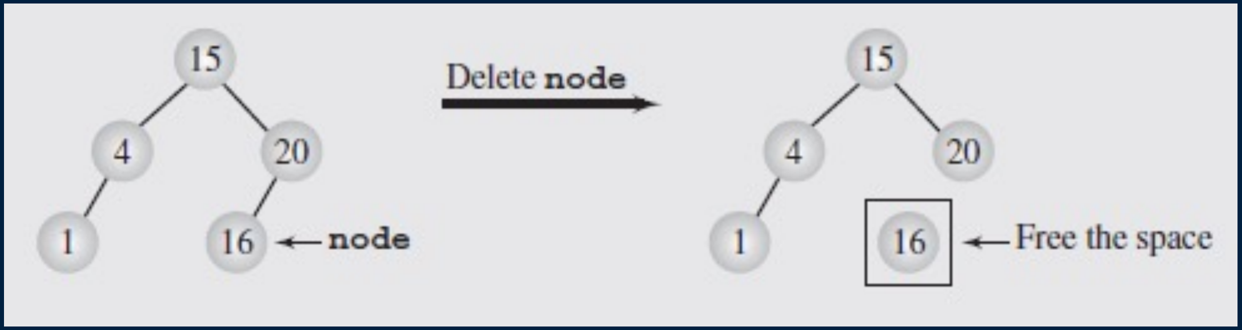
Deletion

Deletion is another operation essential to maintaining a binary search tree

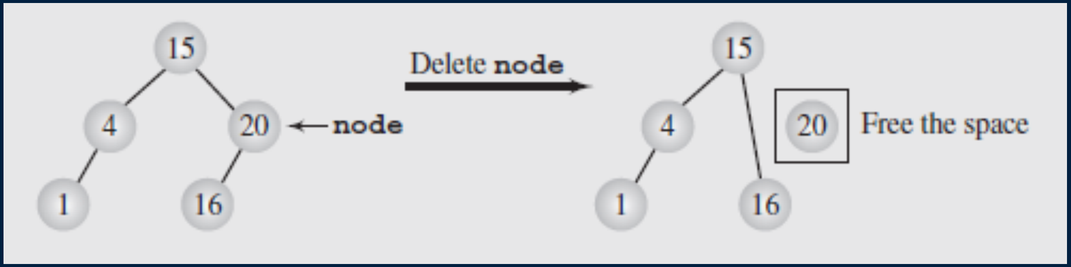
The more children a node has, the more complex the deletion process

There are three cases of deletion that need to be handled:

- The node is a leaf; this is the easiest case, because all that needs to be done is to set the parent link to null and delete the node



- The node has one child (which can be a single node or a tree); also easy, as we set the parent’s to point to the node’s child

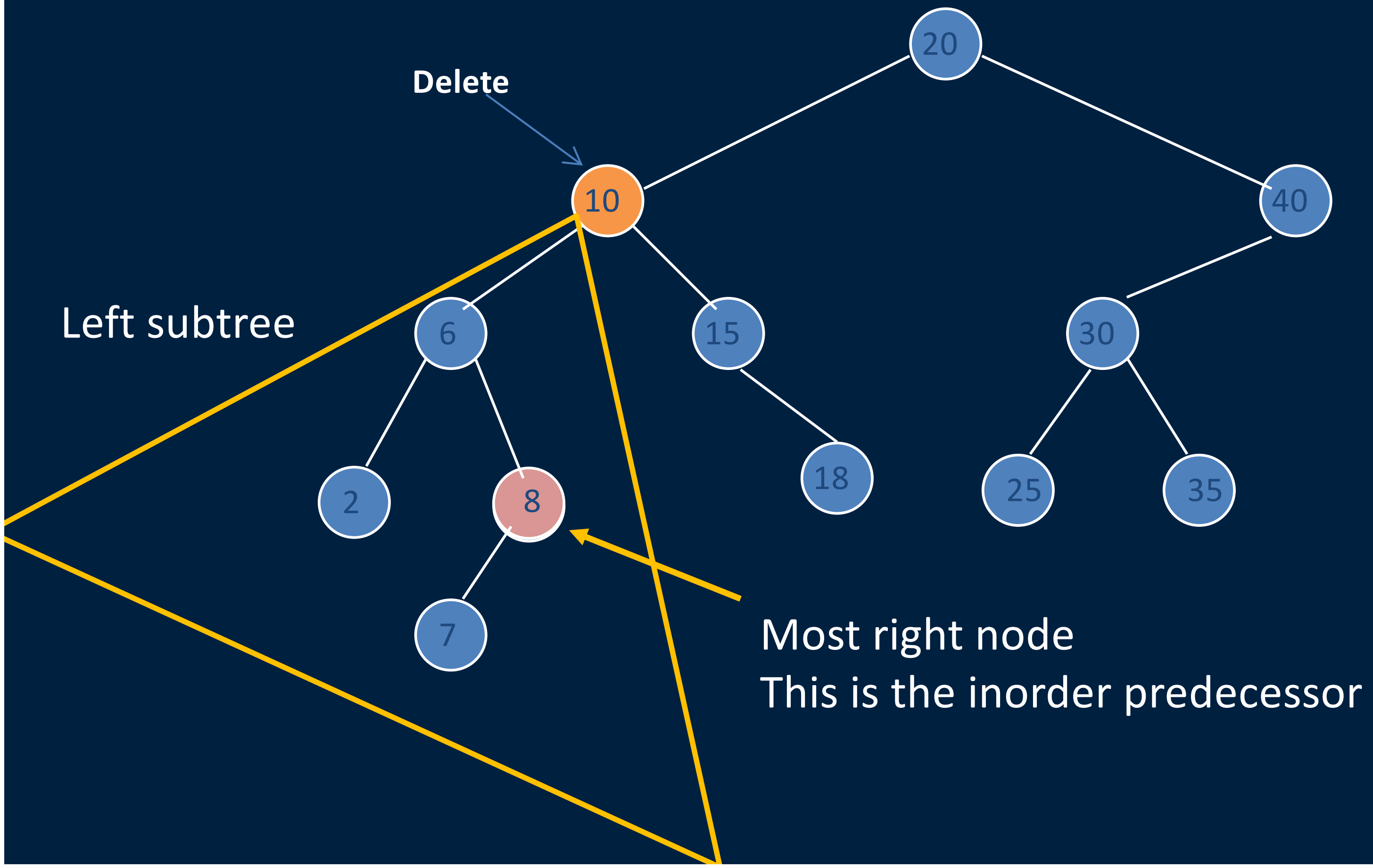


- The third case is when the node has two children. This is more tricky. We will consider one way to do it:
  - Deletion by copying

Delete by Copying

- One approach to handling deleting is called deletion by copying and was proposed by Thomas Hibbard and Donald Knuth in the 1960s
- If the node is easy to delete, we delete it (case 1 or 2)
- If case 3 (two children), we locate the node’s inorder predecessor by searching for the right-most node in the left subtree
- The key of this node replaces the key of the node to be deleted
- We delete the inorder predecessor:
  - We recall the two simple cases of deletion: if the rightmost node was a leaf, we delete it; if it has one child (left child), we set the parent’s to point to the node’s child
- This way, we delete a key k1 by overwriting it by a key k2 and then deleting the node holding k2

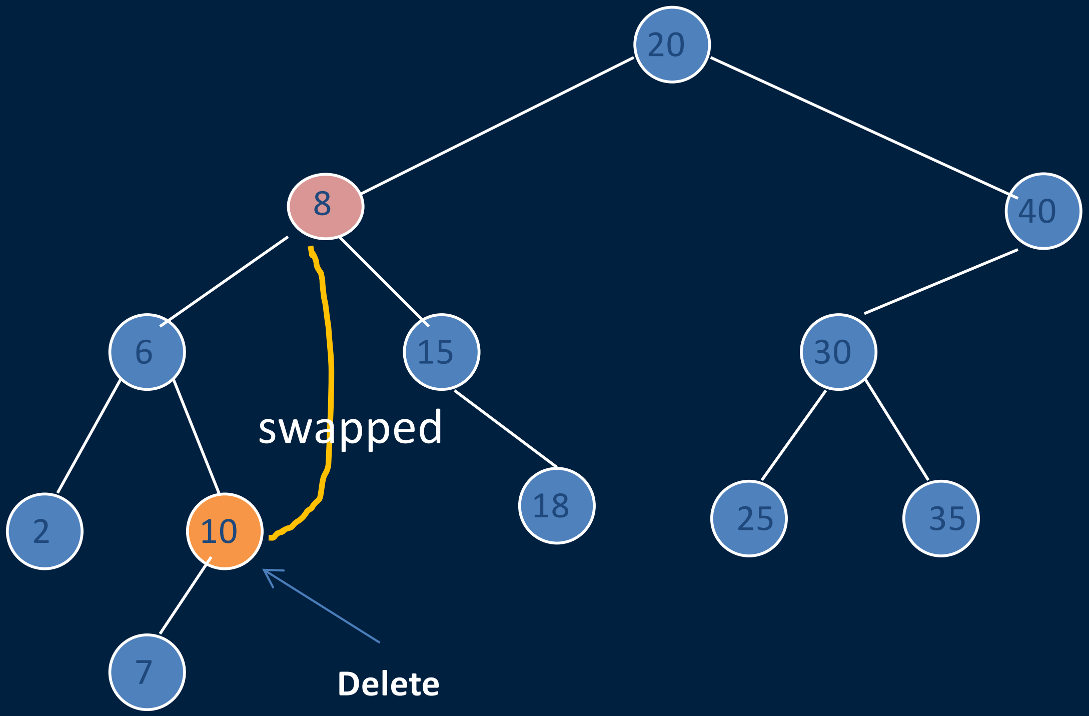
Delete by Copying using inorder predecessor



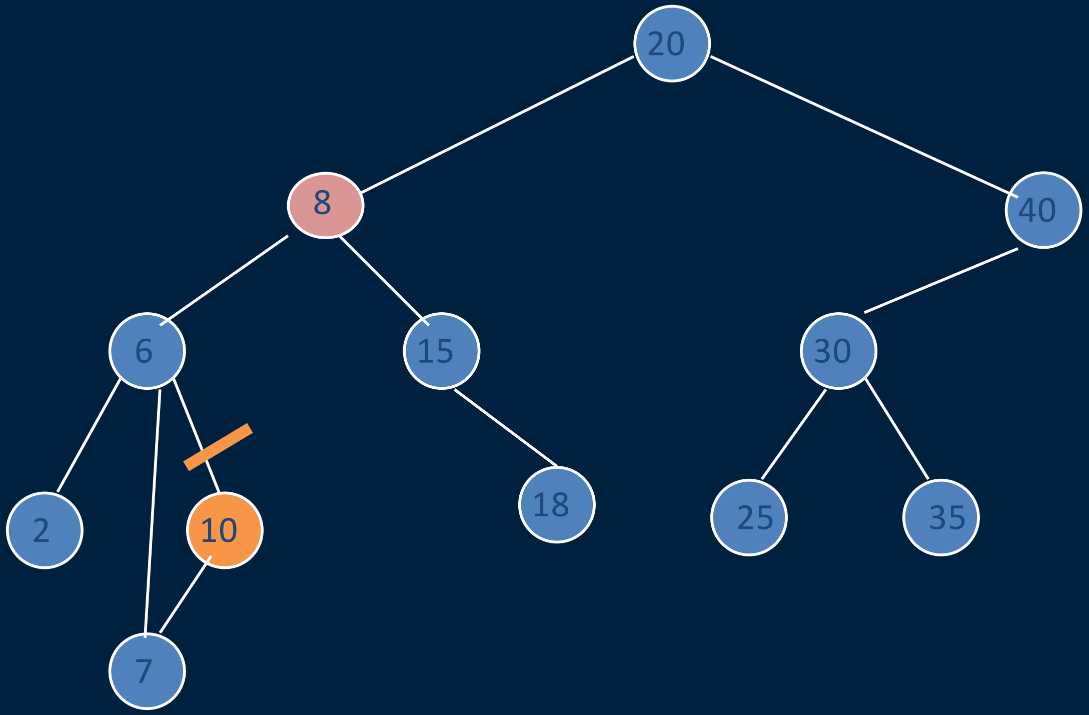
Left subtree

Most right node
This is the inorder predecessor

Delete using inorder predecessor

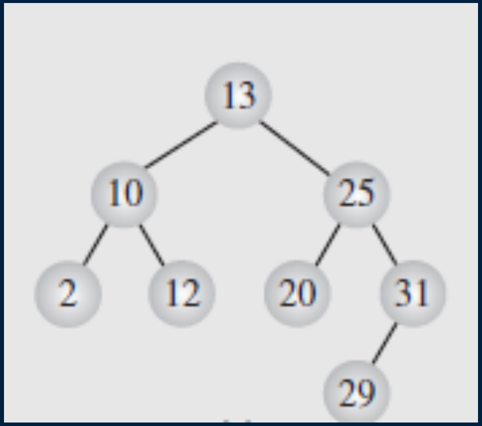


Delete using inorder predecessor



4

Complexity of Search in BST



Consider the following BST:

To determine the complexity of the search we will consider the number of comparisons to be done

- If we search for node 13 only one comparison is required
- If we search for node 29 then maximum of 4 comparisons required
- If we search for non-existing node (node 30), also 4 comparisons are needed
- In general:

  The complexity is given by the number of nodes along the path

The complexity as we have seen in the previous example is determined by 2 factors:

- The shape of the tree
- The position of the node we are searching for

In the shape of the tree analysis, we will study best, average, and worst case

In the position of the node analysis we will study worst case analysis. This implies searching for a node that is on level h of the tree.

Hence we will focus on the shape of the tree with the node always in the bottom at level h

Worst Case Tree Shape: All internal nodes have one child only. Thus, the tree is effectively a linked list.
The number of comparisons required is equal to N, hence
Thus search is O(N).
Best Case Tree Shape: The height of the tree is minimized. This case is a bit hard to analyze, so we assume that the tree is complete (requires $N = 2^{h+1} - 1$). The height of the tree is h = Log(N). Thus, the search is O(Log N).
Average tree shape: This one is more difficult. However, it can be shown that the average height of a randomly built BST is: h = O(Log N). Thus, search is O(Log N) for such a BST tree.

THE UNIVERSITY OF BRITISH COLUMBIA