CSC 172– Data Structures and Algorithms

Lecture #16

Spring 2018

Please put away all electronic devices


a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]
0
1
8
7
5
4
9
6

Announcement

- Extra credit opportunity:
  - Sorting dance
    - http://www.cs.rochester.edu/courses/172/spring2018/sorting_dance.html
    - (Any sorting algorithm)
    - You need to have 10+ students
      - Multiple workshop teams together?!
    - You have to upload the video to YouTube and provide us the link by April 23 (include the list of students participated)
    - Also, 2 bonus points (divided among the number of participants) for the best dance!

How it will impact

- It will heal 3 pts of your overall project or exam score.
- Example:
- Your Exam Score:
  - 20 out of 35
- Your Project Score:
  - 15 out of 30
- Your overall score = 20+15 = 35 out of 65

Sorting Dance will heal by replacing
=min (3 * 20/35, 3 * 15/30)
= min(1.71, 1.5)
= 1.5 pts
with 3 pts.

After the dance, your total score for Projects and Exams will be:
20+15 -1.5 + 3 = 36.5

Extremely useful data structure

Special cases include

- Huffman tree
- Expression tree
- Decision tree (in machine learning)
- Heap data structure (later lecture)

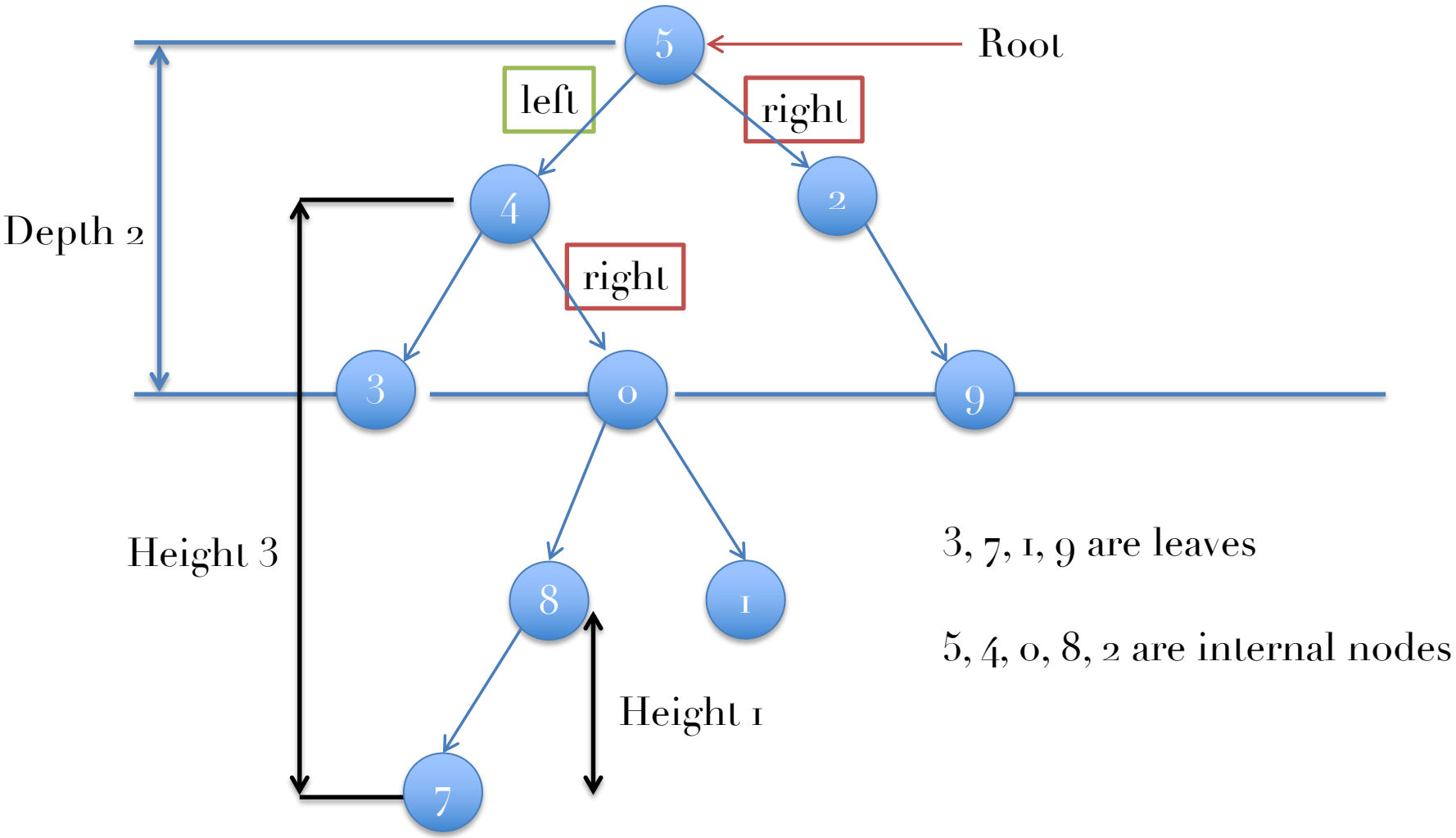
BINARY TREES

Definition

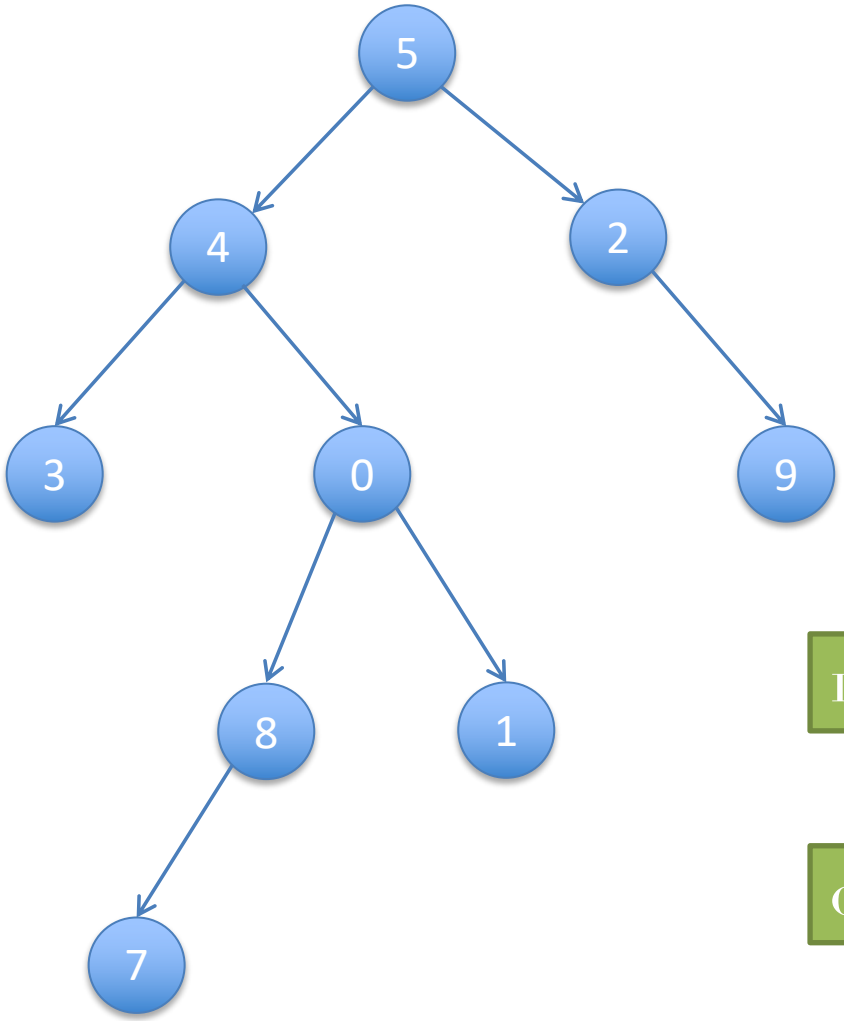
This is not a binary tree:

Neither is this

Binary Trees



Ancestors and Descendants

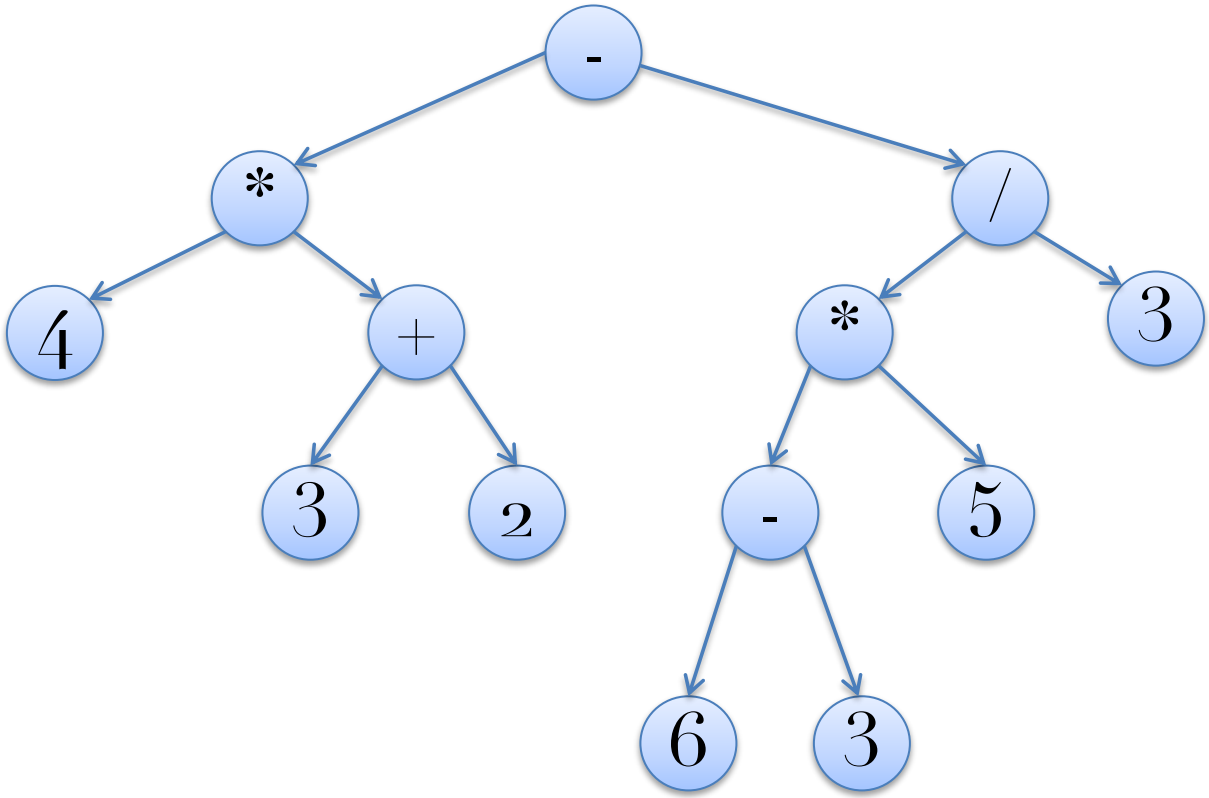


1, 0, 4, 5 are ancestors of 1

0, 8, 1, 7 are descendants of 0

Application: Expression Trees

```
4*(3+2) - (6-3)*5/3
```



How to construct Expression Trees?

Infix

```
4*(3+2) - (6-3)*5/3
```

Shunting Yard Algorithm

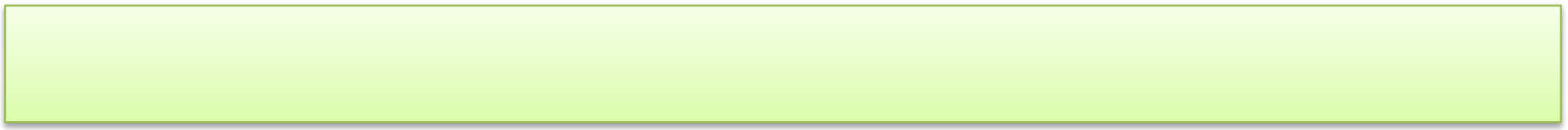
Postfix

```
4 3 2 + * 6 3 - 5 * 3 / -
```

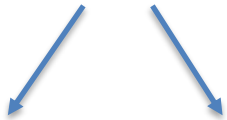
How to construct Expression Trees?

4 3 2 + * 6 3 – 5 * 3 / -

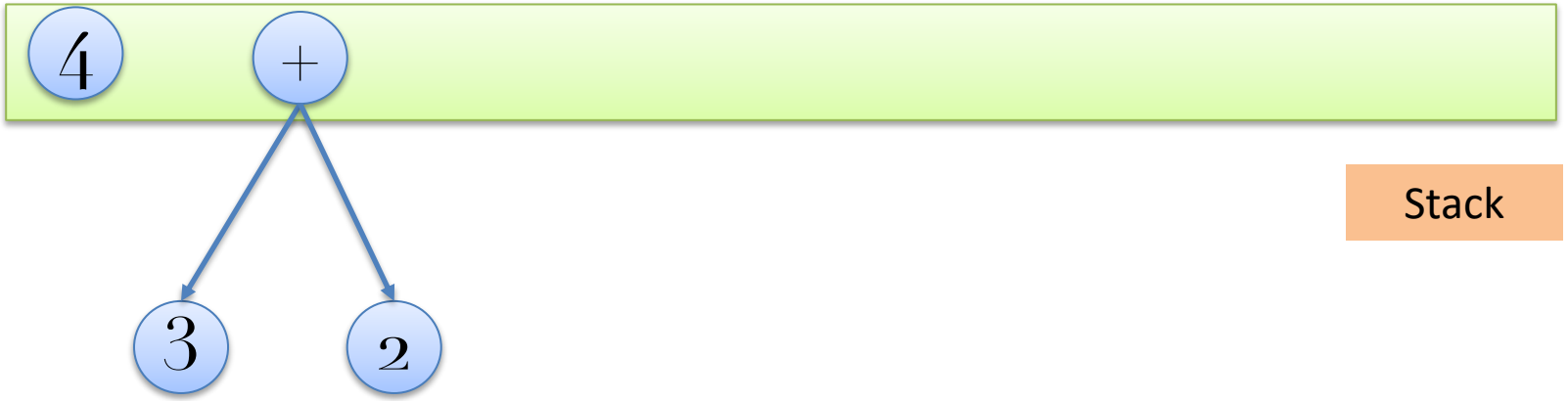
How to construct Expression Trees?



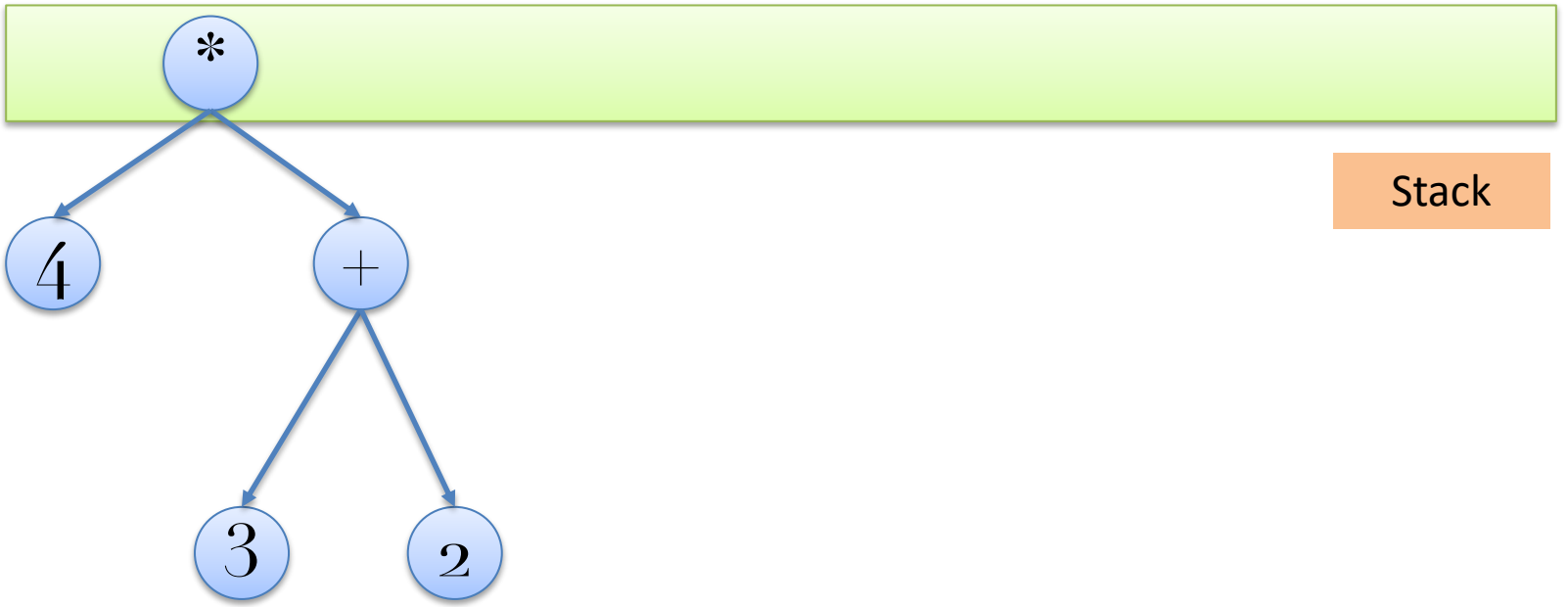
Stack



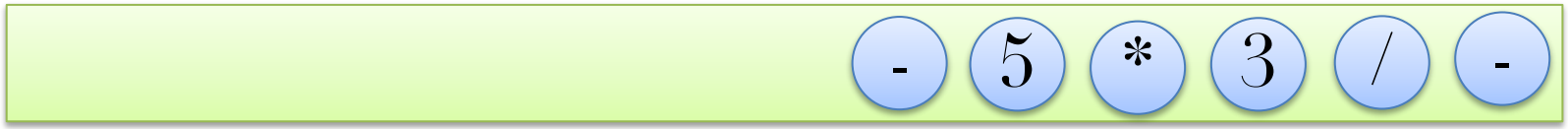
How to construct Expression Trees?

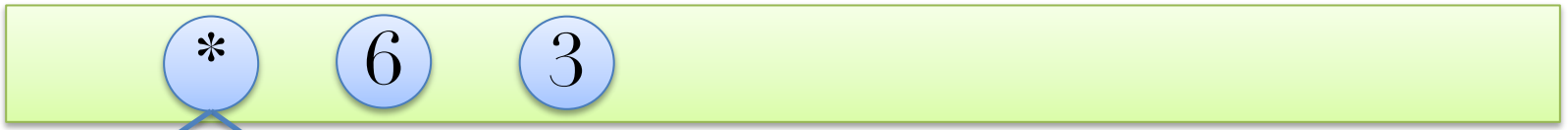


How to construct Expression Trees?

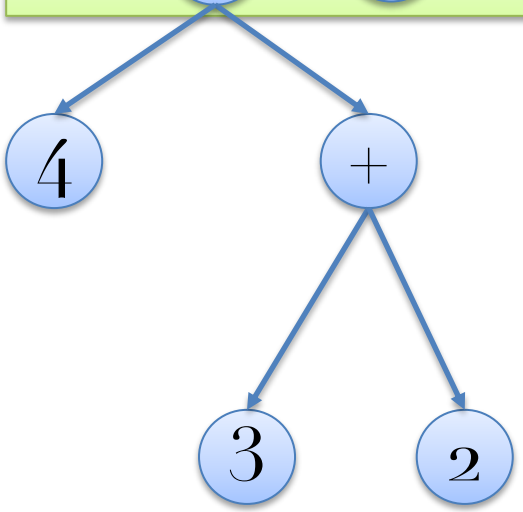


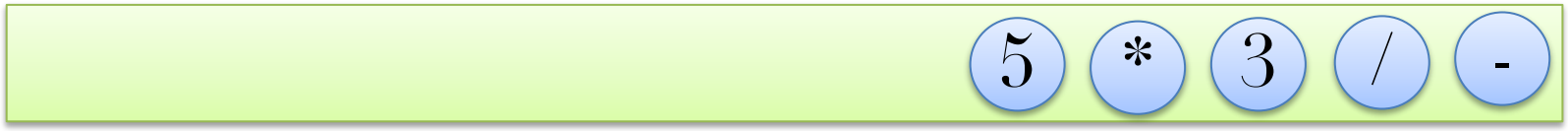
How to construct Expression Trees?

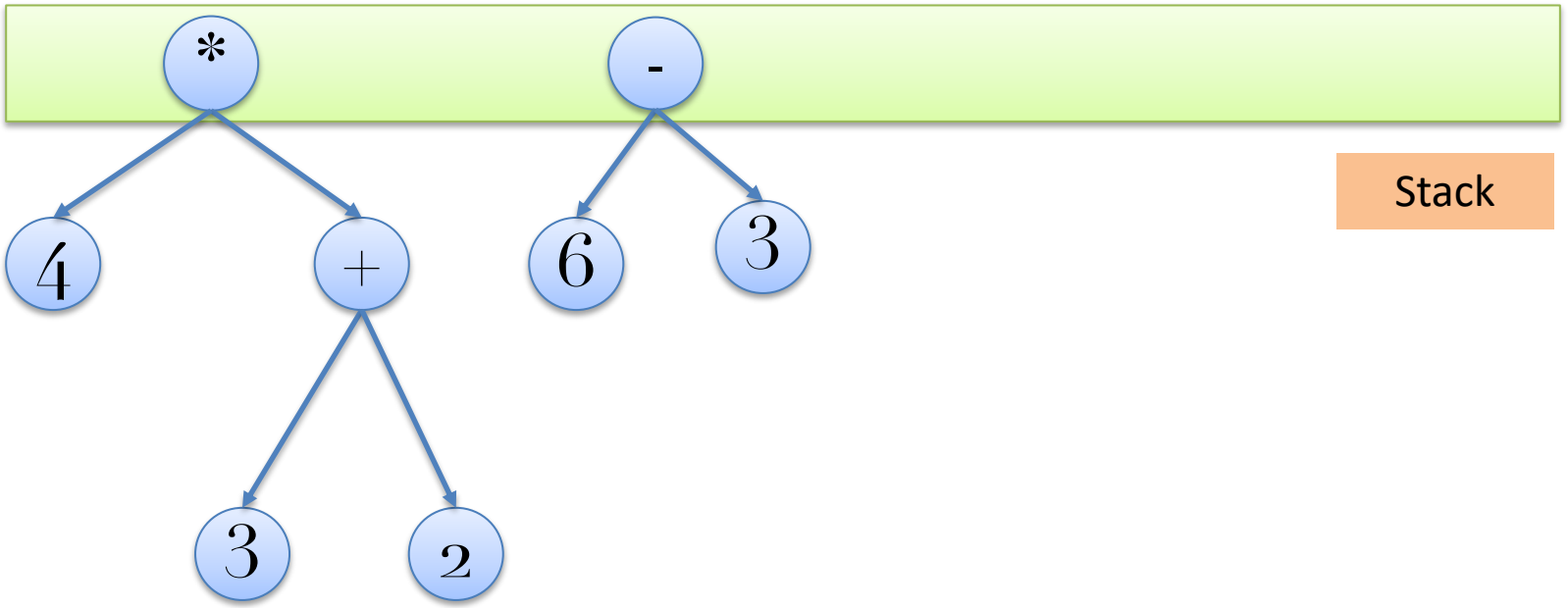




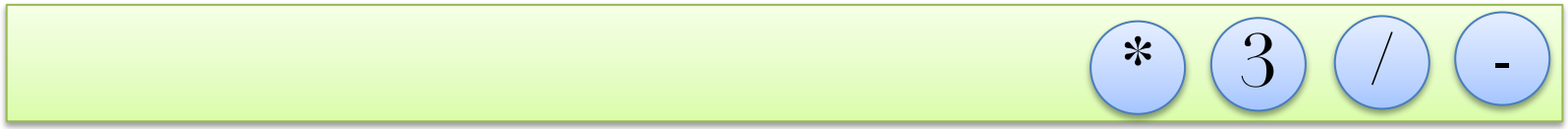
Stack

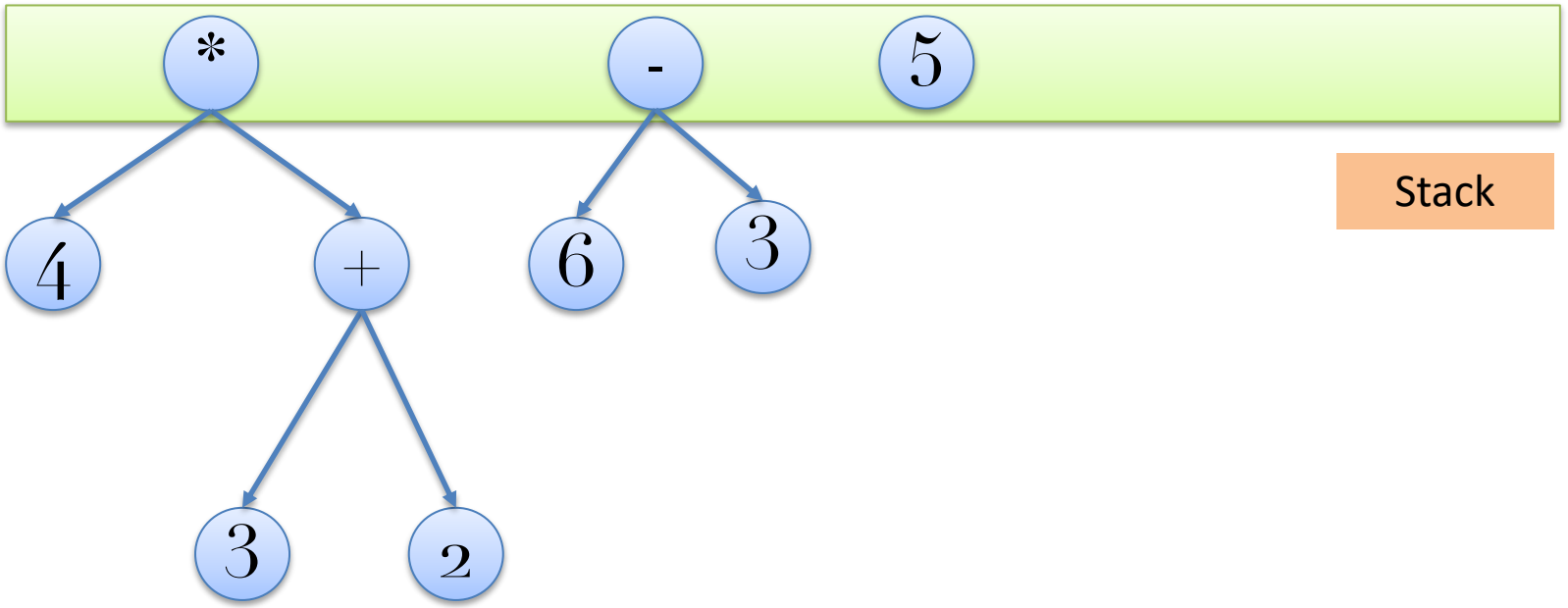



5
*
3
/
-

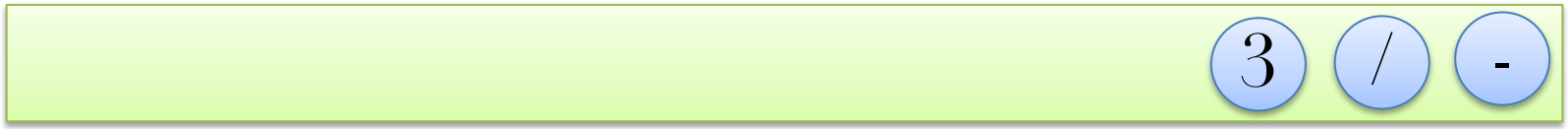

*
-
4
+
6
3
3
2
Stack

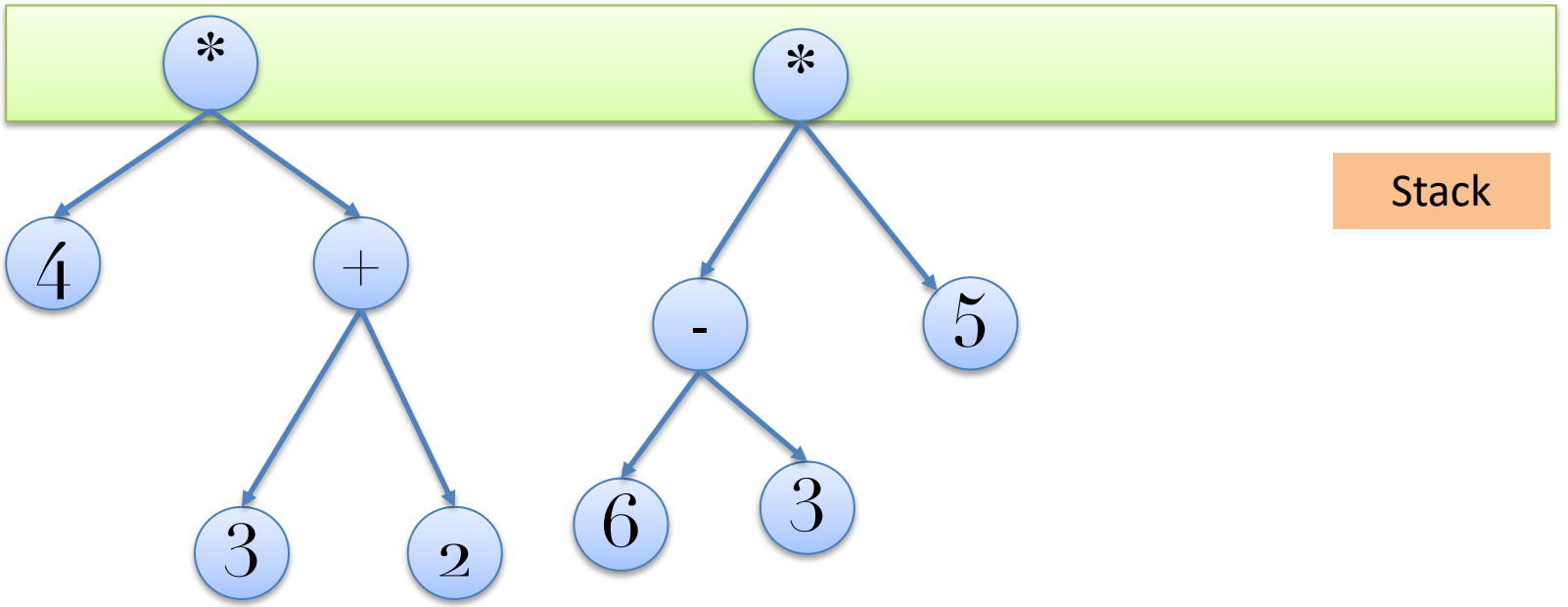
How to construct Expression Trees?



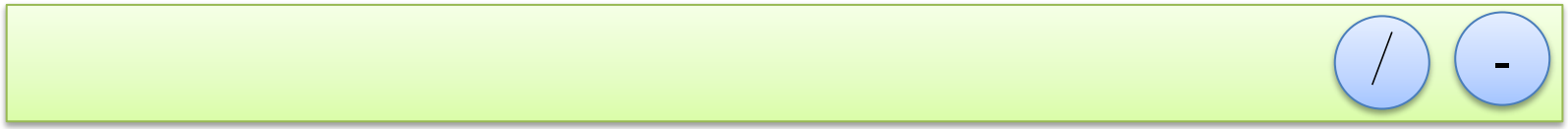


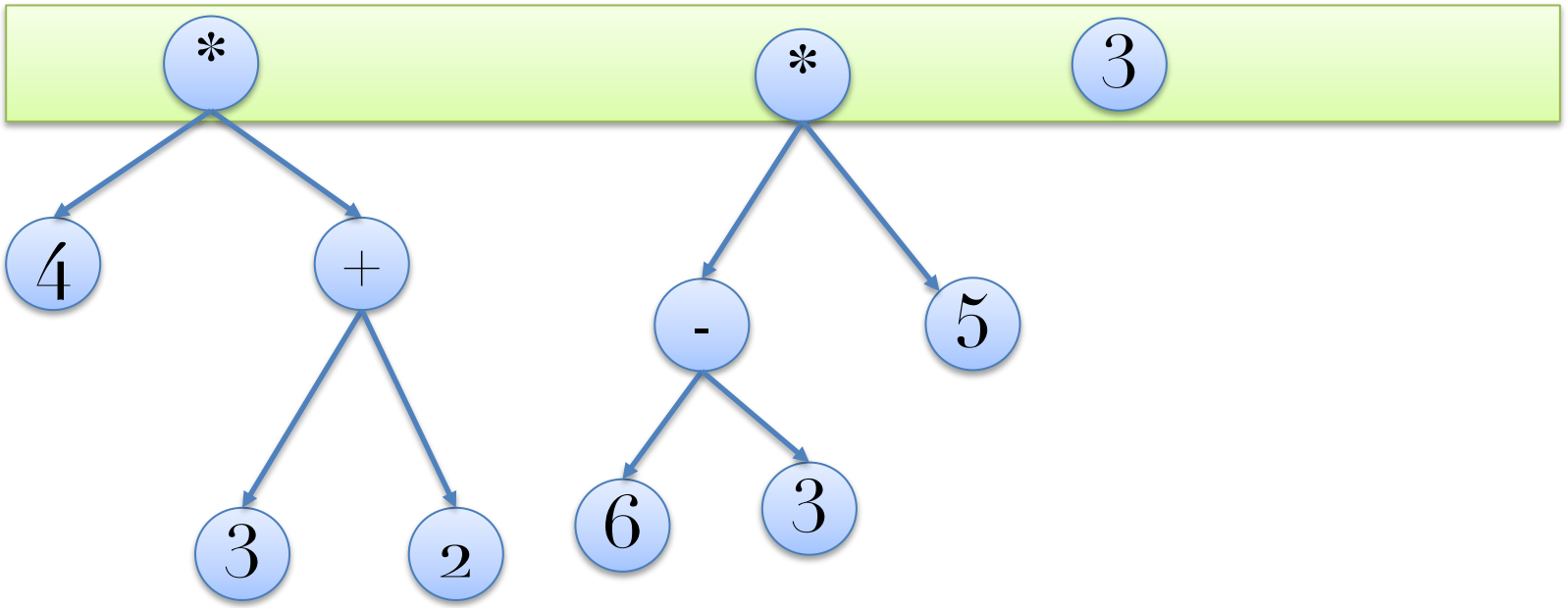
How to construct Expression Trees?



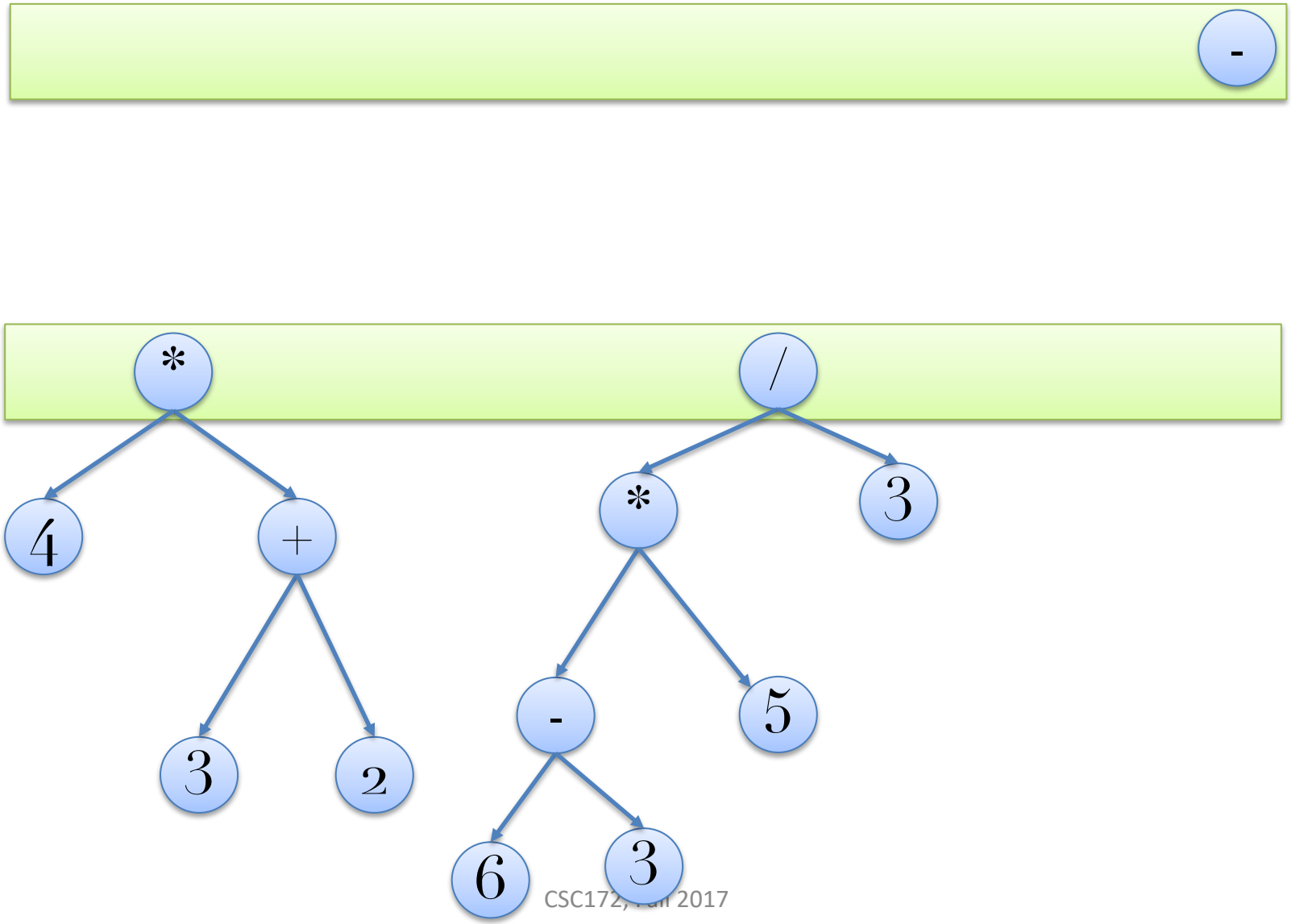


How to construct Expression Trees?

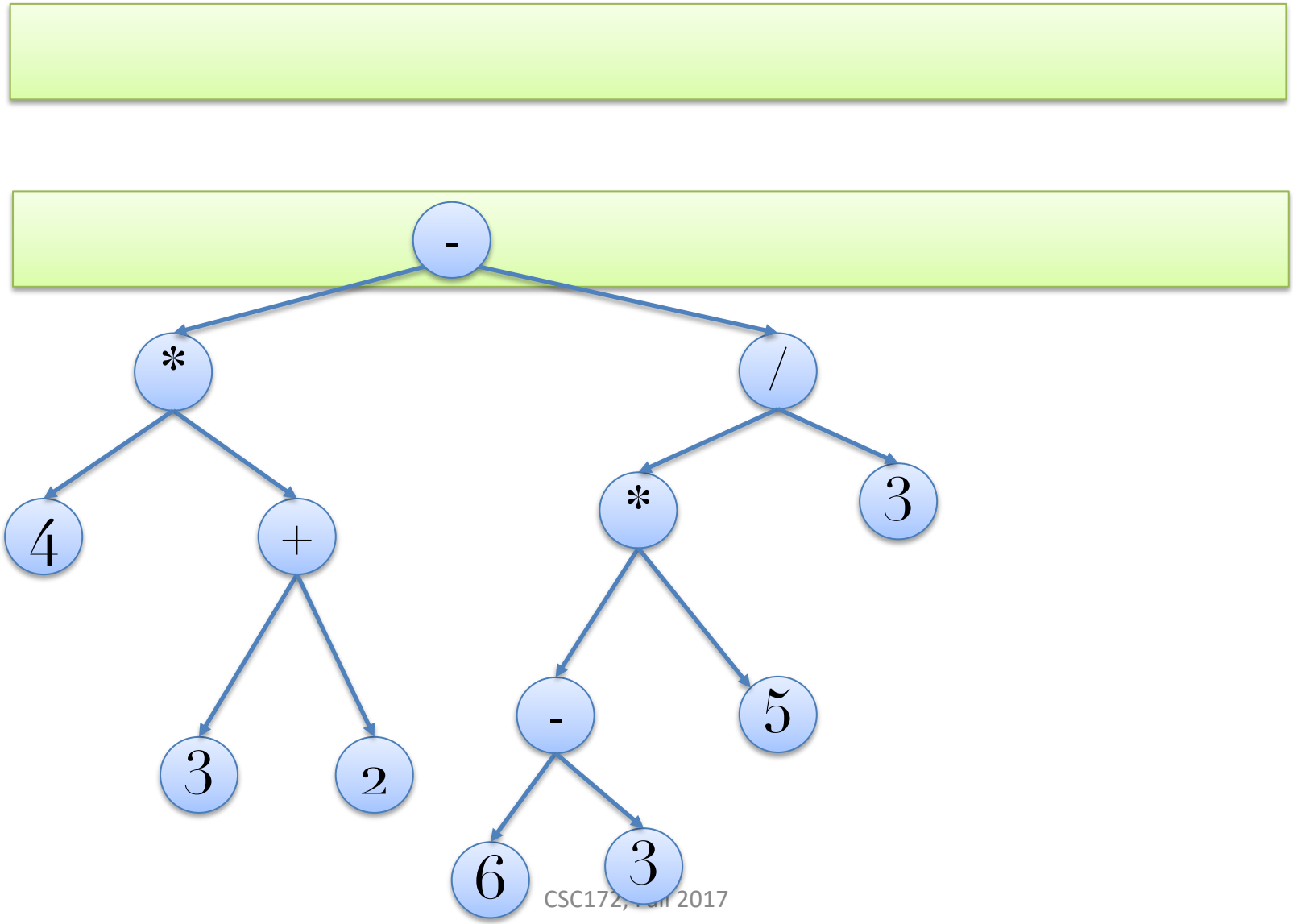




How to construct Expression Trees?

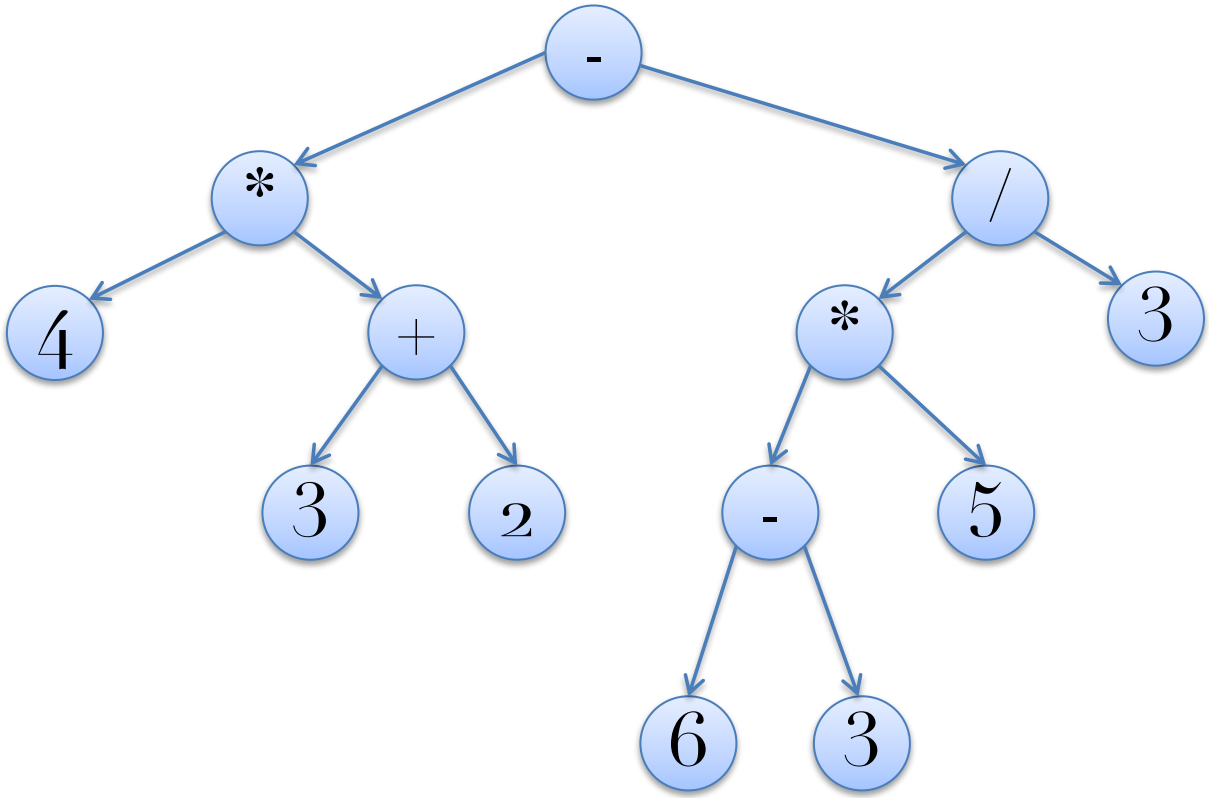


How to construct Expression Trees?



Finally!

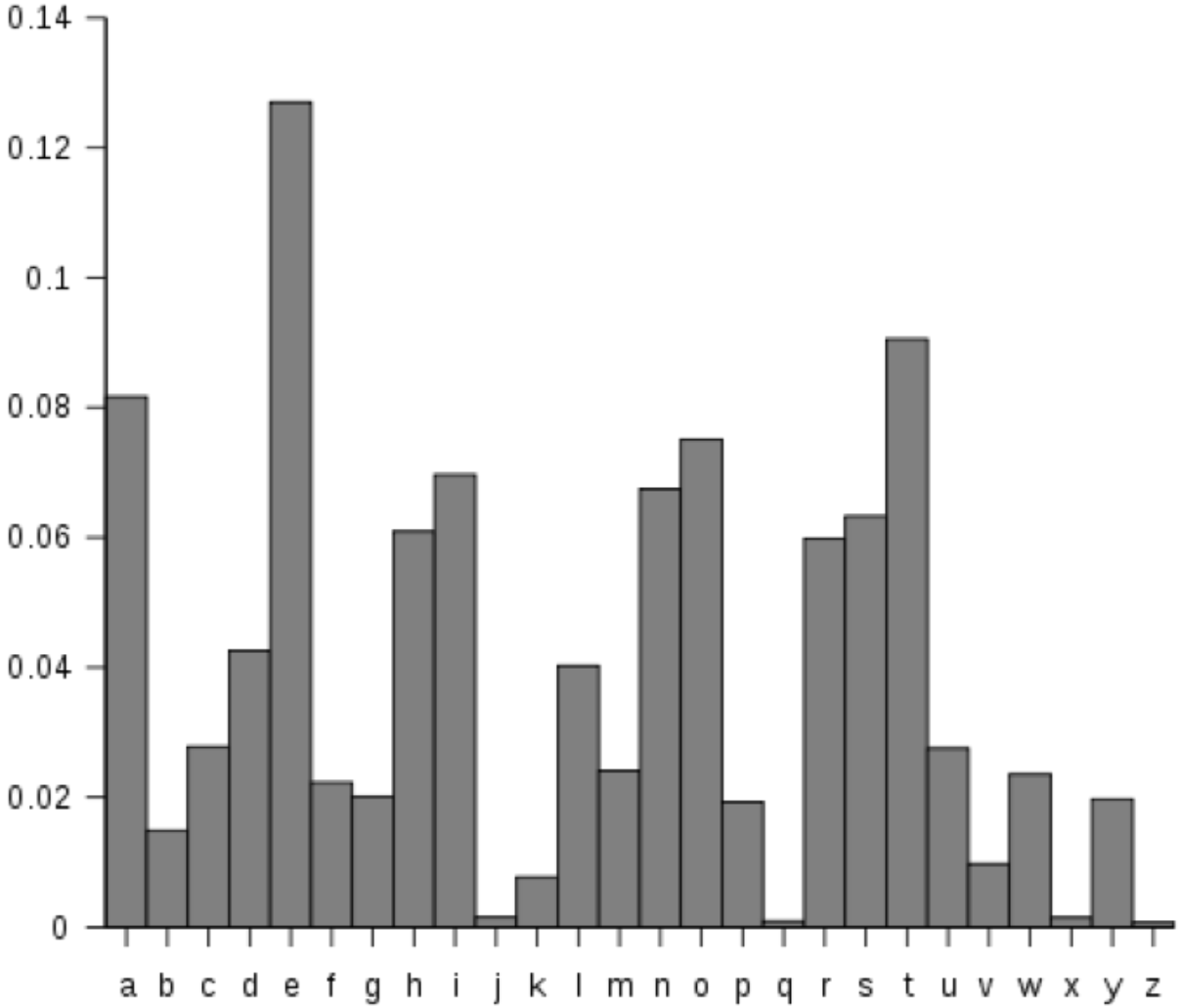
```
4*(3+2) - (6-3)*5/3
```



Another Application: Character Encoding

- UTF-8 encoding:
  - Each character occupies 8 bits
  - For example, ‘A’ = 0x41
- A text document with 10^9 characters is 10^9 bytes long
- But characters were not born equal

English Character Frequencies



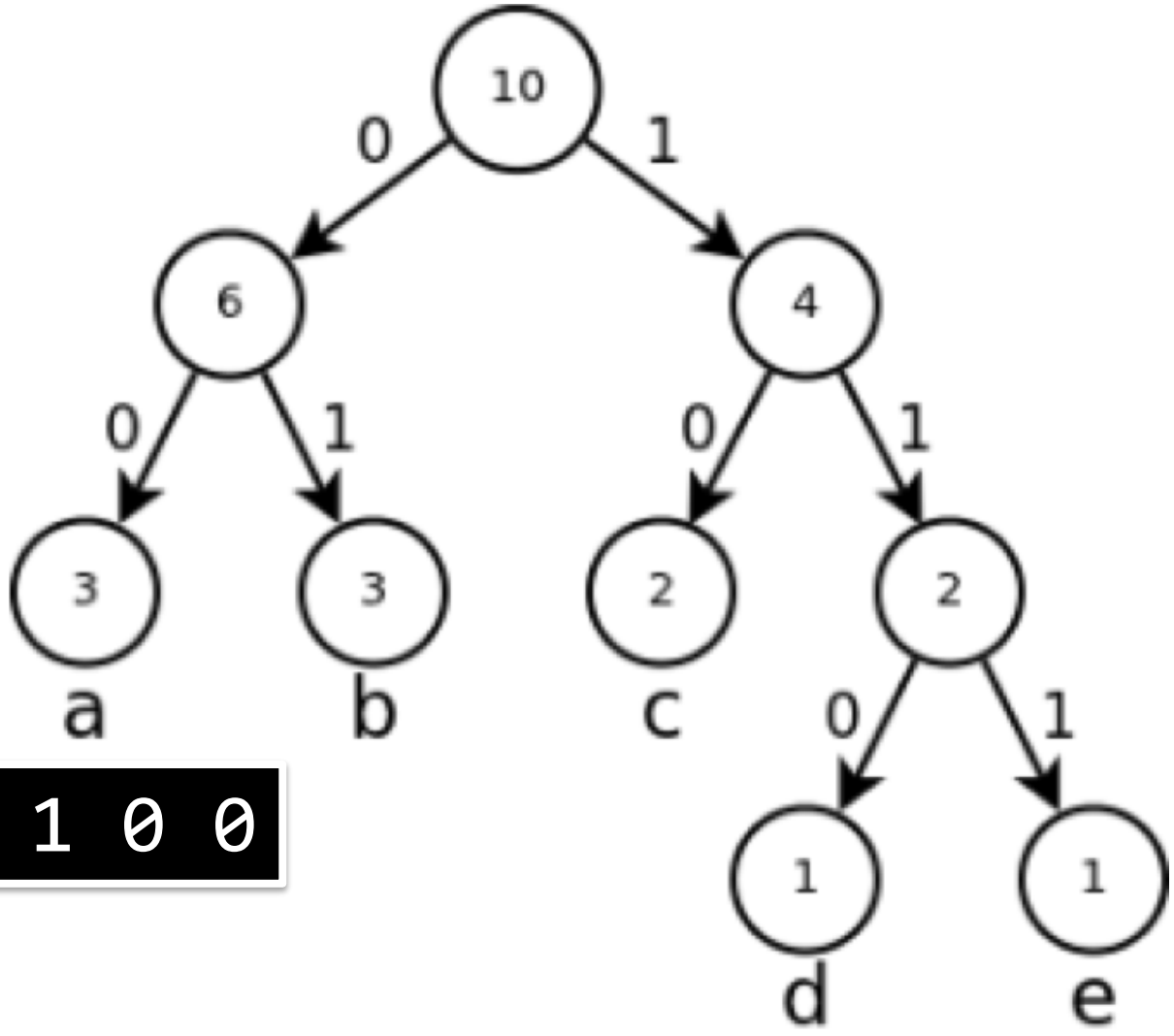
Variable-Length Encoding: Idea

- Encode letter E with fewer bits, say b_E bits
- Letter J with many more bits, say b_J bits
- We gain space if
- $$b_E \cdot f_E + b_J \cdot f_J < 8f_E + 8f_J$$

where ***f*** is the frequency vector

- Problem: how to decode?

One Solution: Prefix-Free Codes

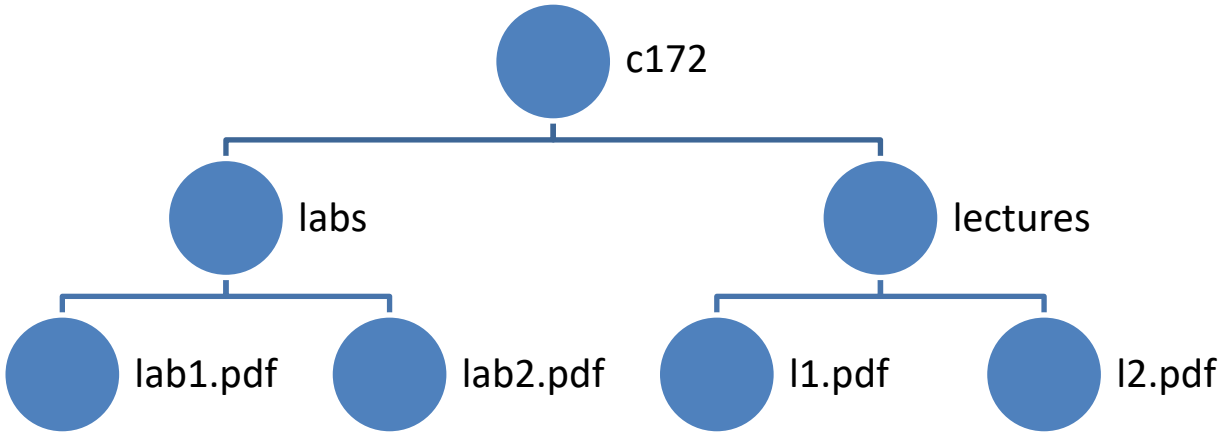



```
1 0 1 1 1 0 1 0 0
```

```
c e b a
```

Printing a Hierarchy

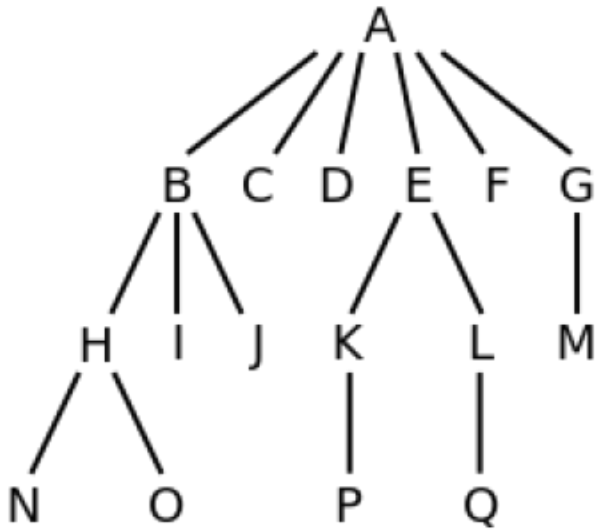
- The easiest way of printing a hierarchy (tree)

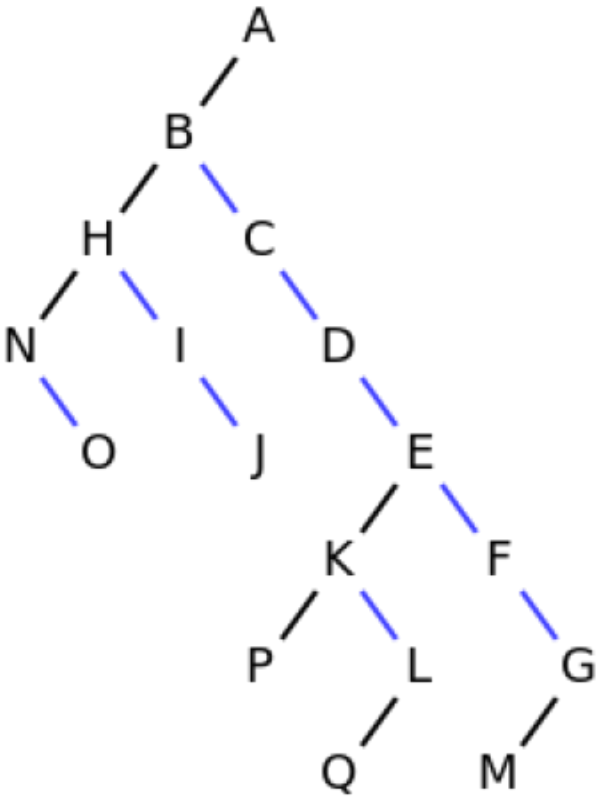



```
c172
├── bootstrap.html
├── calendar.html
├── demo.html
├── hide.js
├── images
│   └── textbook.jpg
├── index.html
├── index.html.backup
├── index.html.backup2
├── jq.js
├── labs
│   ├── lab10.pdf
│   ├── lab11.pdf
│   ├── lab1.pdf
│   ├── lab2.pdf
│   ├── lab3.pdf
│   ├── lab4.pdf
│   ├── Lab4.zip
│   ├── lab5.pdf
│   ├── lab6.pdf
│   ├── lab7.pdf
│   ├── lab7.tex
│   ├── Lab7.zip
│   ├── lab8.pdf
│   ├── lab9.pdf
│   ├── URList.java
│   └── URNode.java
├── Labs.html
├── lectures
│   ├── l10.pdf
│   ├── l11.pdf
│   ├── l12.pdf
│   ├── l13.pdf
│   ├── l14.pdf
│   ├── l15.pdf
│   ├── l16.pdf
│   ├── l17.pdf
│   ├── l18.pdf
│   ├── l19.pdf
│   ├── l1.pdf
│   ├── l20.pdf
│   ├── l21.pdf
│   ├── l22.pdf
│   ├── l23.pdf
│   ├── l24.pdf
│   ├── l25.pdf
│   ├── l26.pdf
│   ├── l27.pdf
│   ├── l2.pdf
```

WHY ONLY BINARY TREE?

Any Tree can be “Encoded” as a Binary Tree





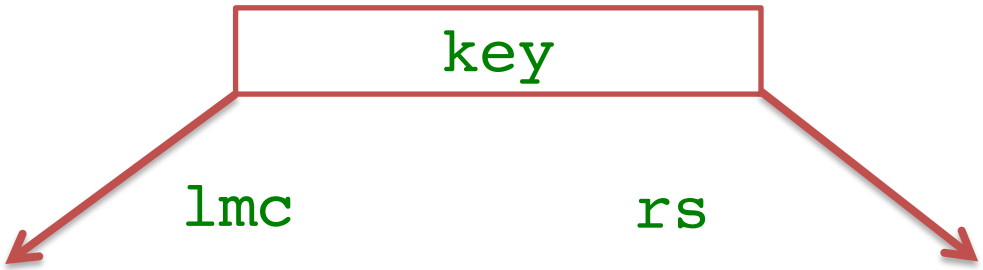
LMC-RS Representation

- In this representation, every node has two pointers:
  - LMC (Left-most-child)
  - RS (Right Sibling)

LMC-RS Representation

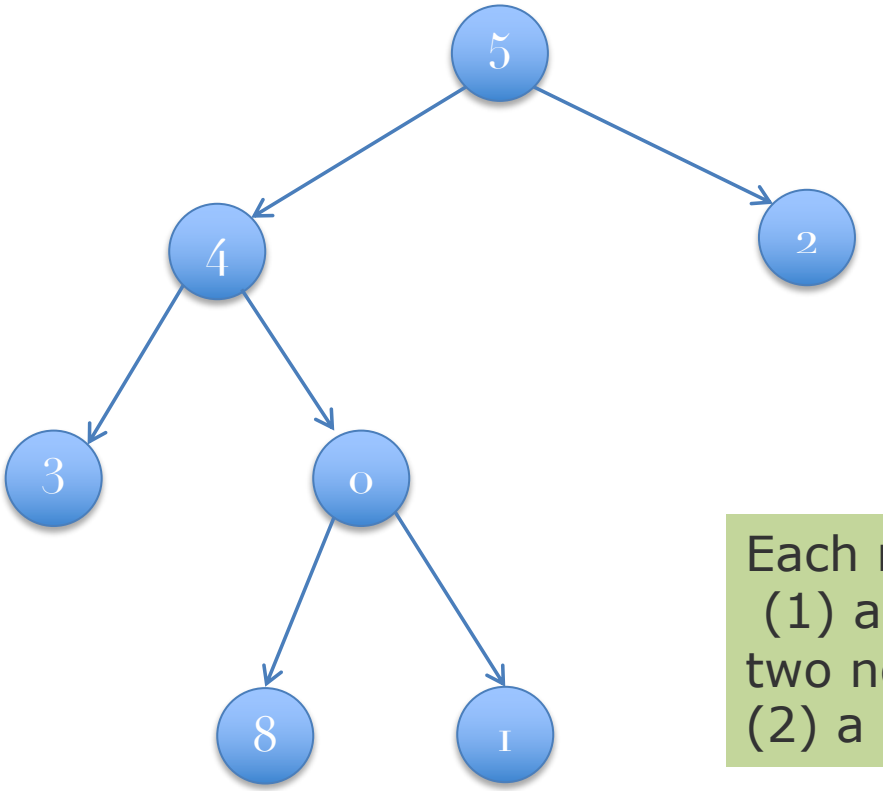
```
public class Node
{
    public int key;
    public Node lmc, rs;

    public Node(int item)
    {
        key = item;
        lmc= rs= null;
    }
}
```



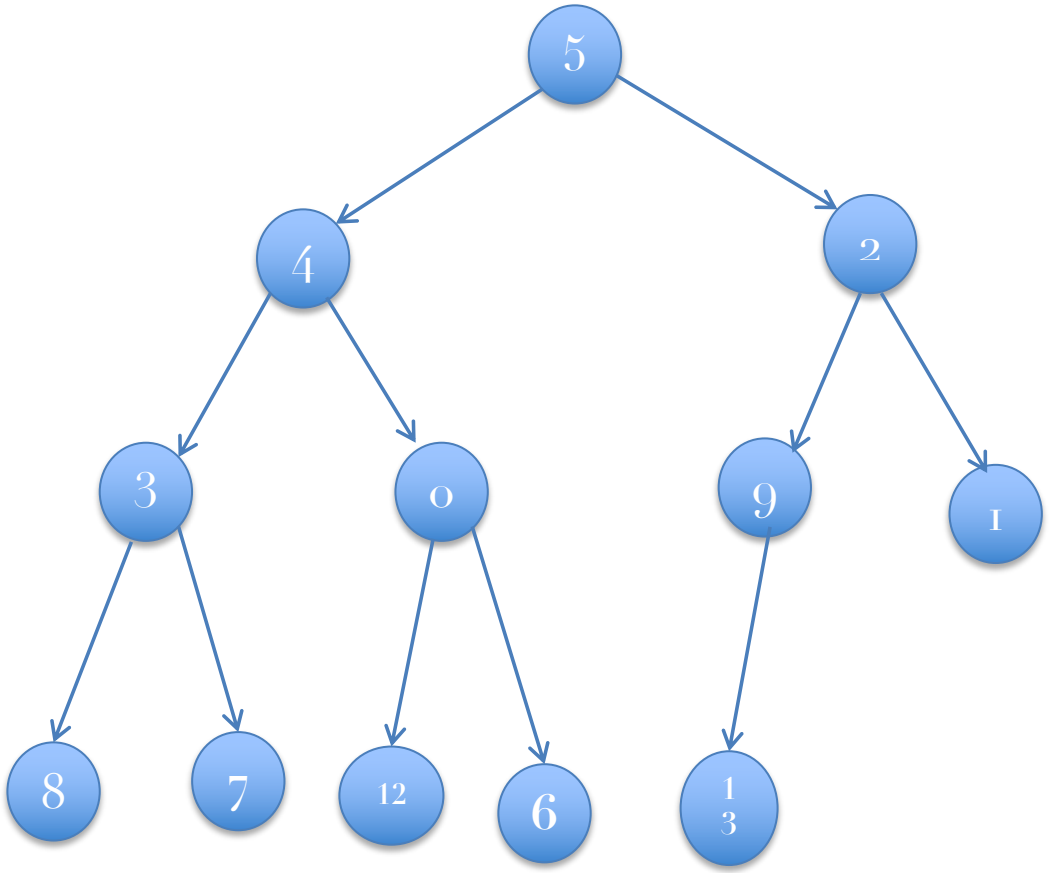
FULL VS. COMPLETE BINARY TREE

Full Binary Tree



Each node is either
(1) an internal node with exactly two non-empty children or
(2) a leaf

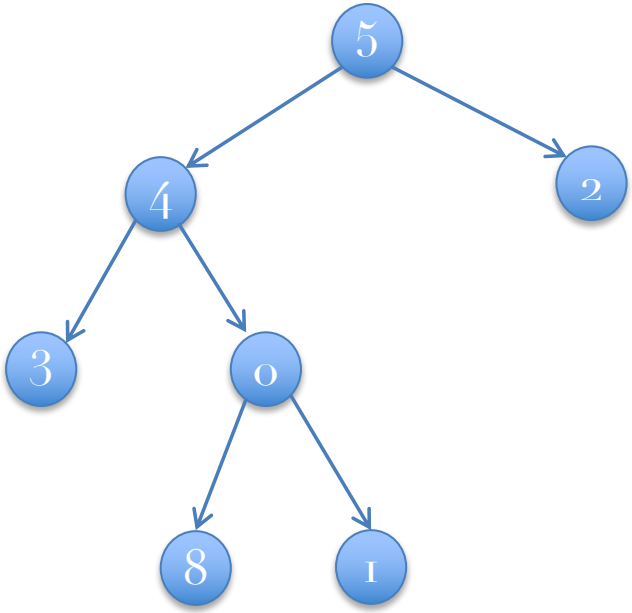
Complete Binary Tree



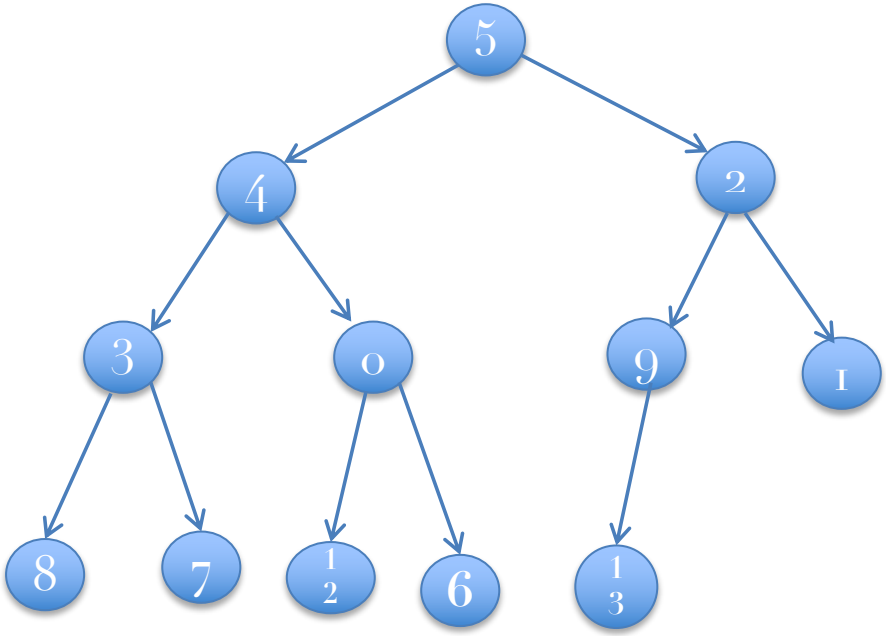
- Has a restricted shape obtained by starting at the root and filling the tree by levels from left to right.
- In the complete binary tree of height d, all levels except possibly level d−1 are completely full.
- The bottom level has its nodes filled in from the left side.

Let's see the examples again

Full vs. Complete Binary Tree



Full but not complete



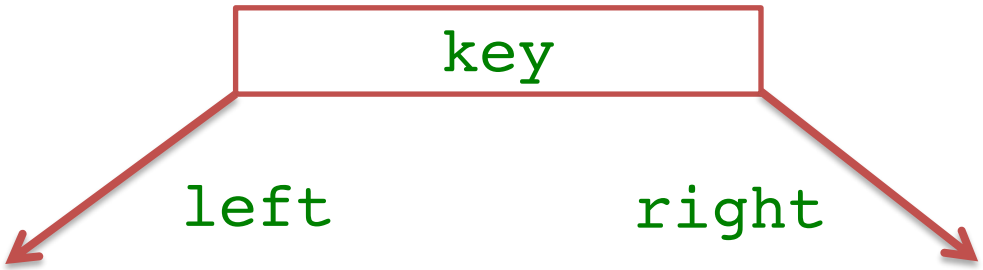
Complete but not full

TREE USING JAVA

A BTNode in Java

```
public class Node
{
    public int key;
    public Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
```



TREE WALKS/TRAVERSALS

There are many ways to traverse a binary tree

- (reverse) In order
- (reverse) Post order
- (reverse) Pre order
- Level order = breadth first

Inorder Traversal

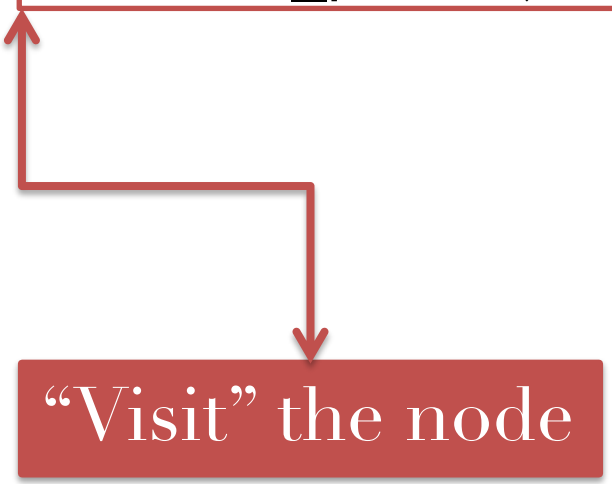
```
Inorder-Traverse(BTNode root)

- Inorder-Traverse(root.left)
- Visit(root)
- Inorder-Traverse(root.right)
```

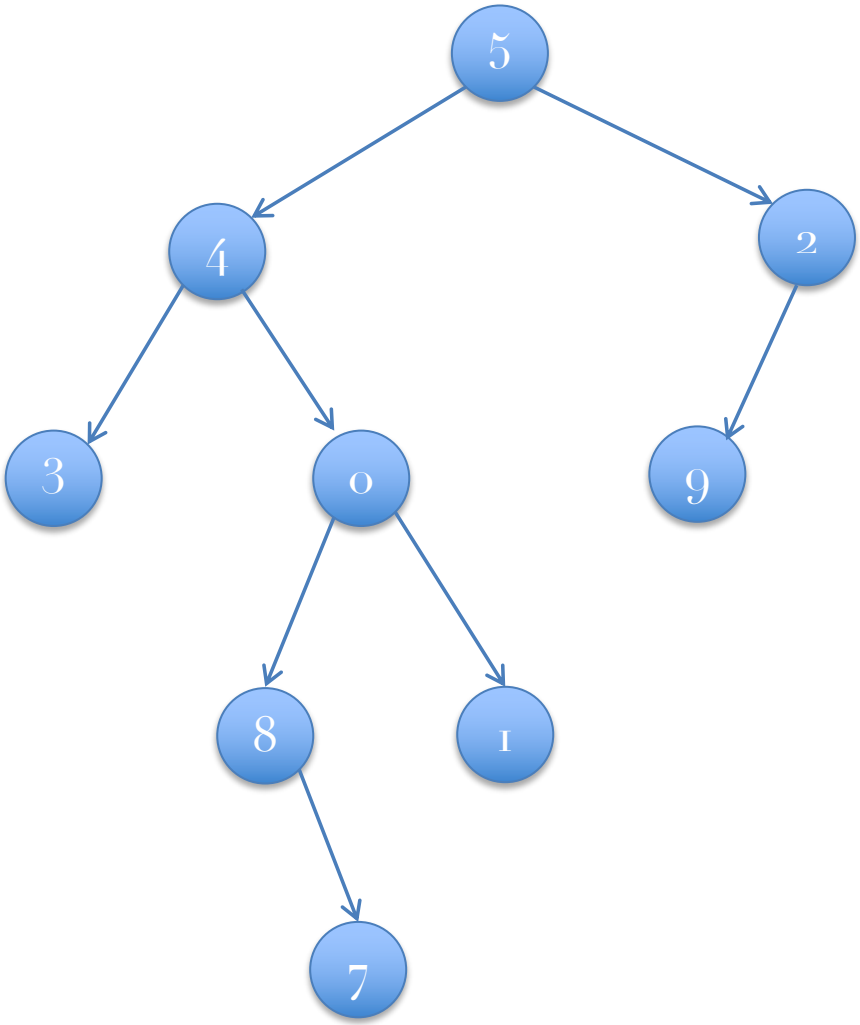
Also called the *(left, node, right)* order

Inorder Printing in C++

```
void inorder_print(BTNode root)
{
    if (root != null) {
        inorder_print(root.left);
        printNode(root);
        inorder_print(root.right);
    }
}
```



In Picture



3
4
8
7
0
1
5
9
2

Run Time

- Suppose "visit" takes O(1)-time, say c sec
  - n_l = # of nodes on the left sub-tree
  - n_r = # of nodes on the right sub-tree
  - Note: $n - 1 = n_l + n_r$
- $T(n) = T(n_l) + T(n_r) + c$
- Induction: $T(n) \leq cn$, i.e. $T(n) = O(n)$
- $T(n) \leq cn_l + cn_r + c$
  $= c(n-1) + c$
  $= cn$

Reverse Inorder Traversal

- `RevInorder-Traverse(root.right)`
- `Visit(root)`
- `RevInorrder-Traverse(root.left)`

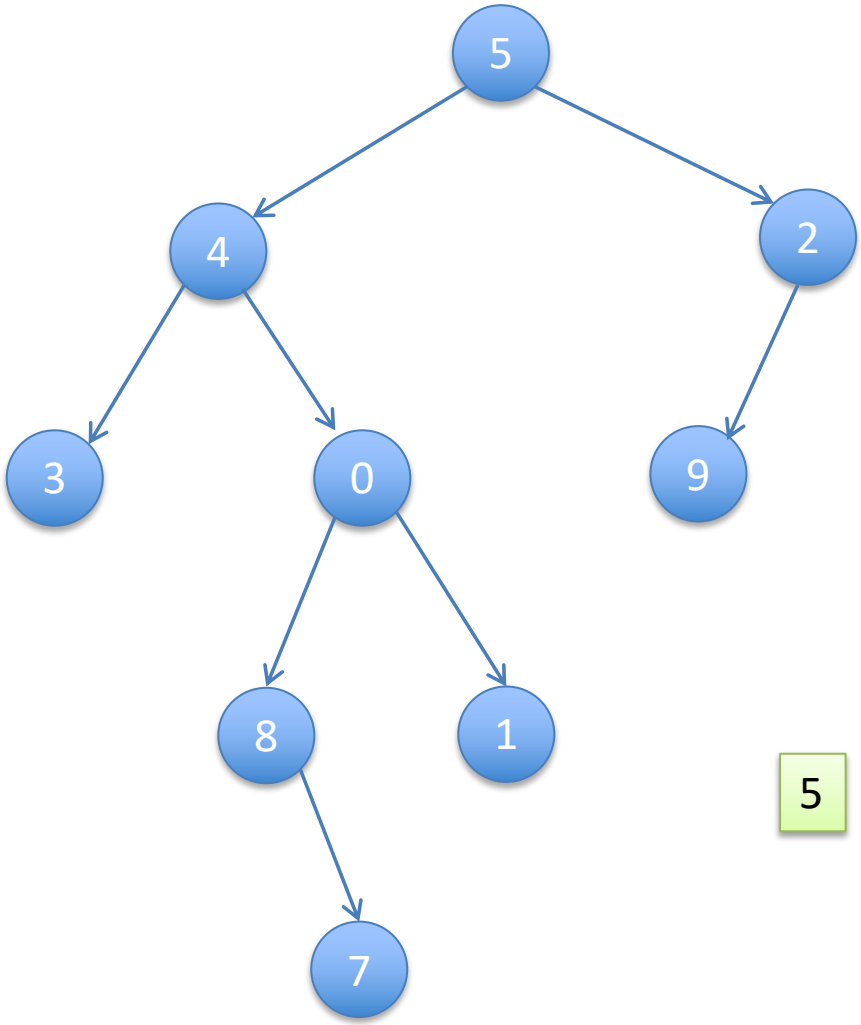
The (right, node, left) order

The other 4 traversal orders

- Preorder: (node, left, right)
- Reverse preorder: (node, right, left)
- Postorder: (left, right, node)
- Reverse postorder: (right, left, node)

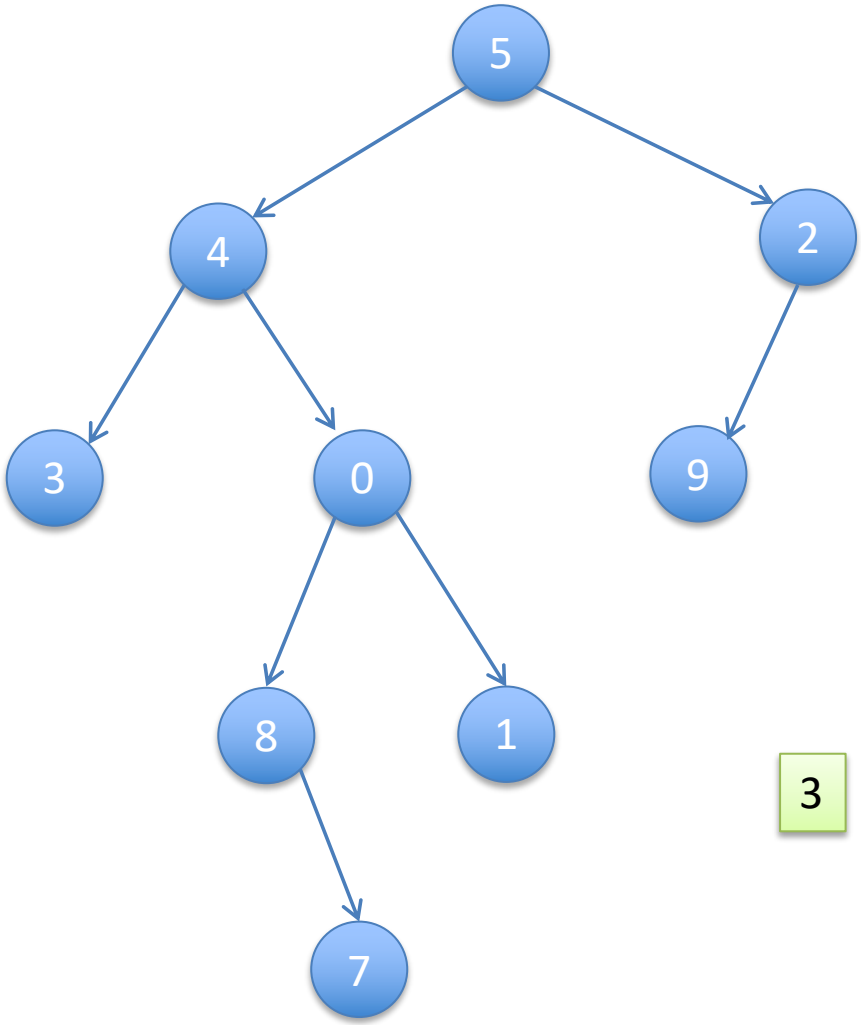
We'll talk about level-order later

What is the preorder output for this tree?



5	4	3	0	8	7	1	2	9

What is the postorder output for this tree?



3	7	8	1	0	4	9	2	5

Questions to Ponder

```
void inorder_print(BTNode root) {
    if (root != NULL) {
        inorder_print(root.left);
        printNode(root);
        inorder_print(root.right);
    }
}
```

Write the above routine without the recursive calls?

Use a stack

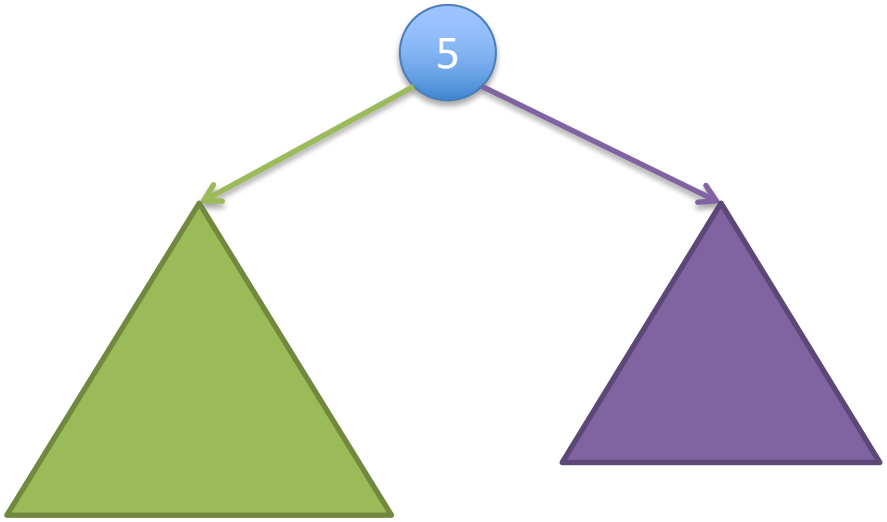
Don't use a stack

Try this for fun! Not required for Exam or Quiz

Reconstruct the tree from inorder+postorder

Inorder	3	4	8	7	0	1	5	9	2

Preorder	5	4	3	0	8	7	1	2	9



Questions to Ponder

- Can you reconstruct the tree given its postorder and preorder sequences?
- How about inorder and reverse postorder?
- How about other pairs of orders?
- How many trees are there which have the same in/post/pre-order sequence? (suppose keys are distinct)

Number of trees with a given inorder sequence

Catalan numbers:
Not required for Exam or Quiz

Catalan numbers

$$C_n = \sum_{i=1}^{n} C_{i-1} C_{n-i}$$

$$C_0 = 1$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

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

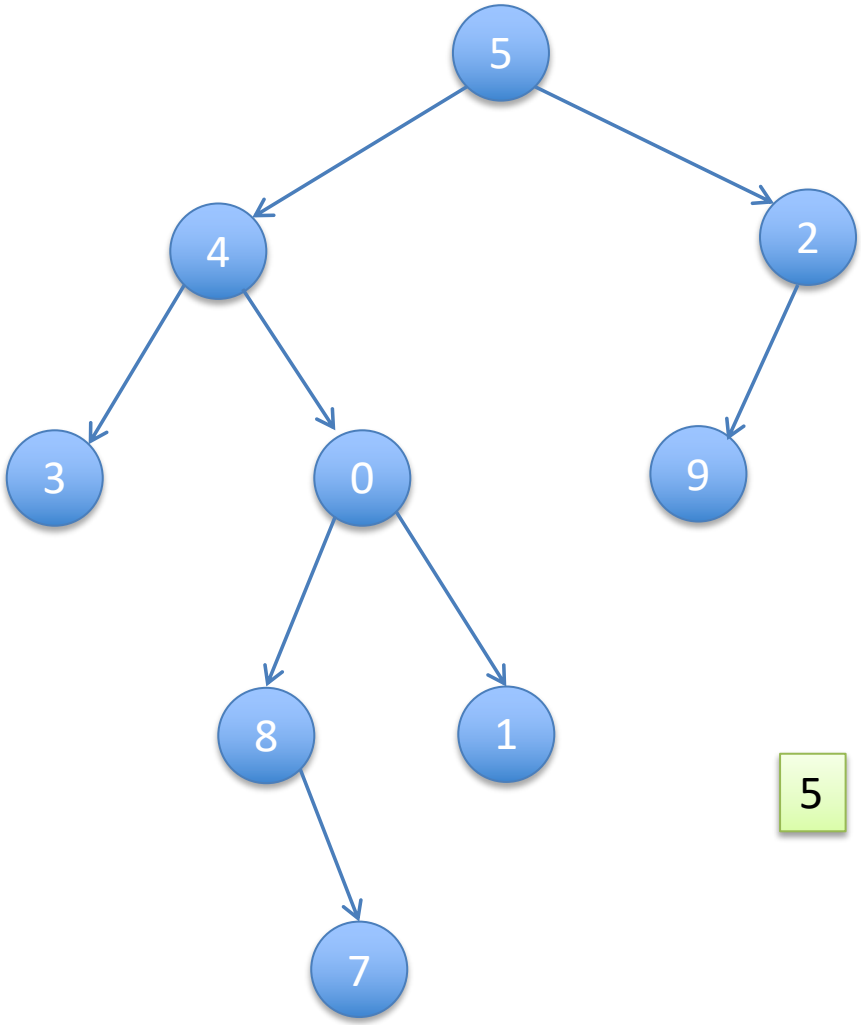
What is a traversal order good for?

- Many things

- E.g., Evaluate(root) of an expression tree
  - If root is an operand, return the operand
  - Else
    - A = Evaluate(root.left)
    - B = Evaluate(root.right)
    - Return A root.key B
      - root.key is one of the operators

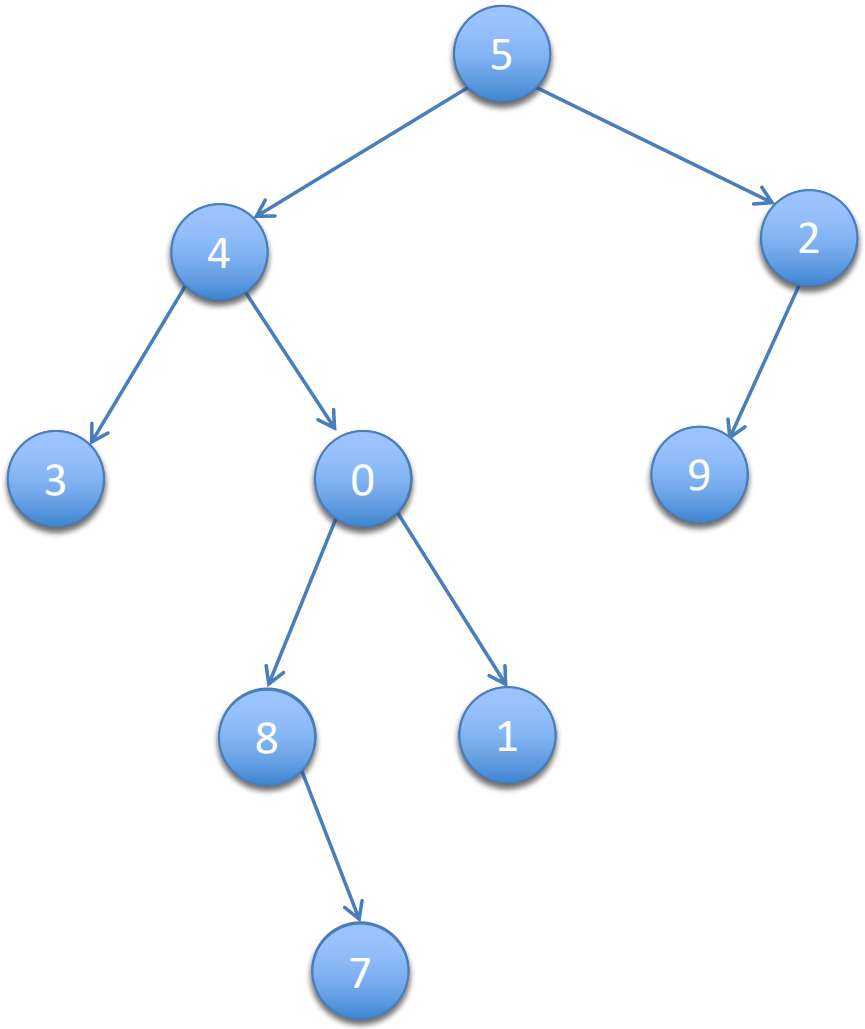
- What traversal order is the above?

Level-Order Traversal



5	4	2	3	0	9	8	1	7

How to do level-order traversal?



A (FIFO) Queue

Level-Order Print in Java

```
void levelorder_print(BTNode root) {




// Implement




}
```

Level-Order Print in Java

```
void printLevelOrder()
    {
        Queue<BTNode> queue = new LinkedList<BTNode>();
        queue.offer(root);
        while (!queue.isEmpty())
        {

            BTNode currNode = queue.poll();
            System.out.print(currNode.getPayLoad() + " ");

            if (currNode.left != null) {
                queue.offer(currNode.left);
            }

            if (currNode.right != null) {
                queue.offer(currNode.right);
            }
        }
    }
```

What if we change the Queue into a Stack

```
void print__________Order()
    {
        Stack<BTN
        stack.pus
        while (!s
        {

            BTNod
            Syste

            /*Enq
            if (t
                s
            }

            /*Enq
            if (t
            stack.push(tempNode.right);
            }
        }
    }
```

What if we change the Queue into a Stack

```
void print__________Order()
    {
        Stack<BTNode> stack = new Stack<BTNode>();
        stack.push(root);
        while (!stack.isEmpty())
        {

            BTNode tempNode = stack.pop();
            System.out.print(tempNode.getPayLoad() + " ");

            /*Enqueue left child */
            if (tempNode.left != null) {
                stack.push(tempNode.left);
            }

            /*Enqueue right child */
            if (tempNode.right != null) {
            stack.push(tempNode.right);
            }
        }
    }
```

What traversal is this?

Reverse Preorder

hat traversal is this?

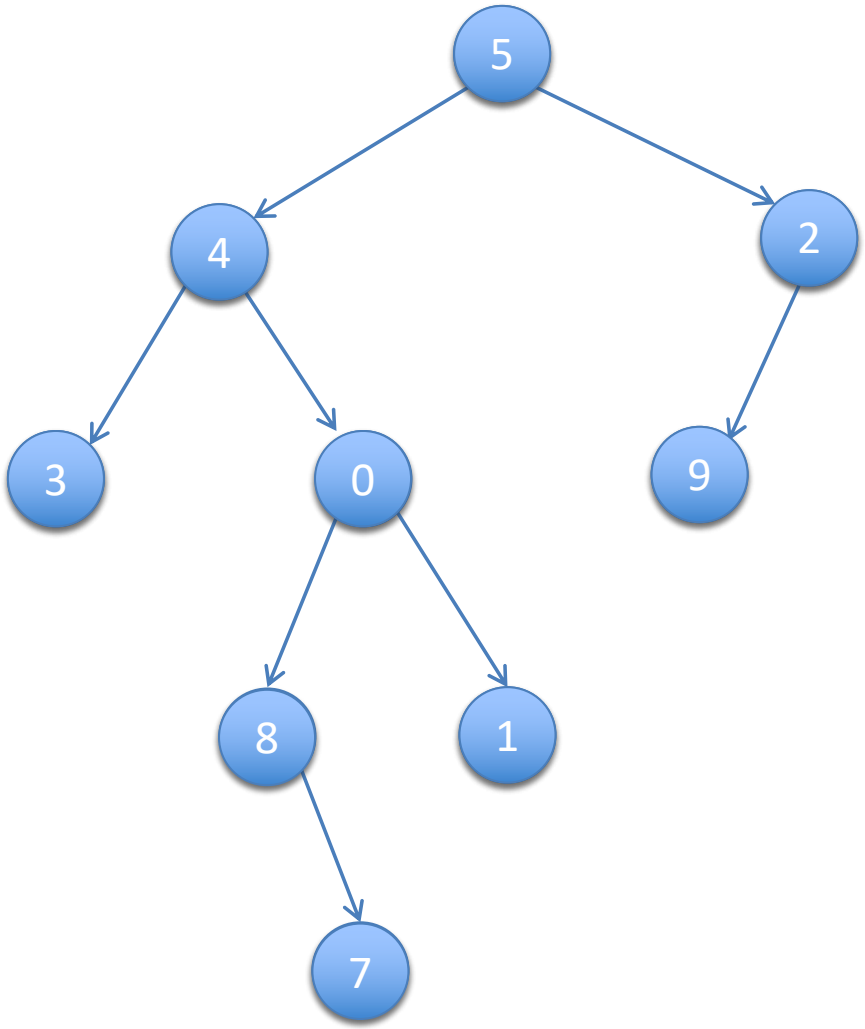
;

verse Preorder

CSC 172, Fall 2017

ANOTHER EXAMPLE WITHOUT RECURSION

How to do in-order traversal? (without recursion)

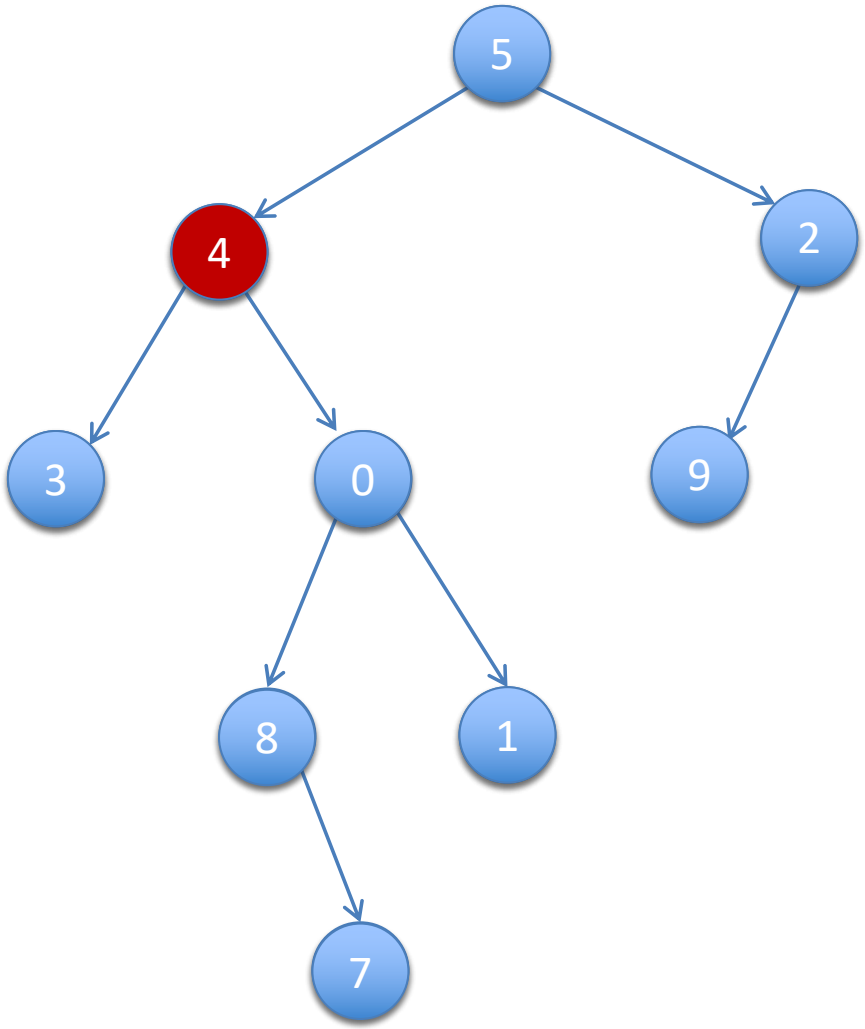


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

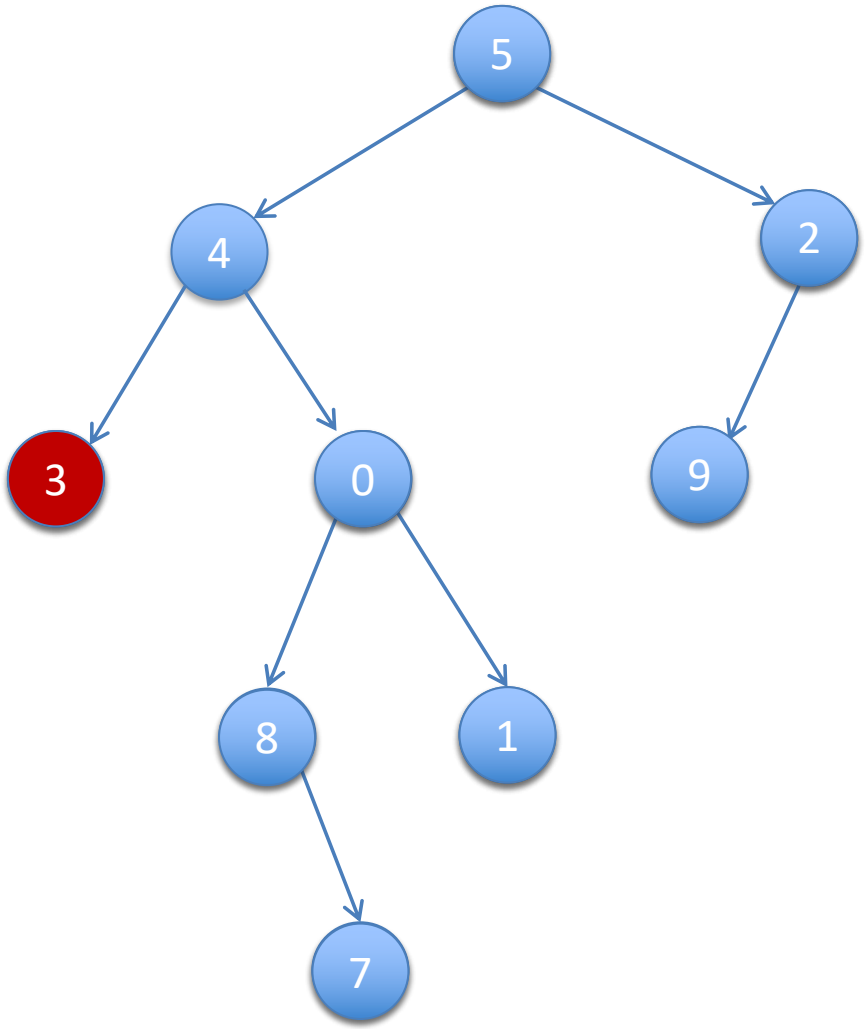


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

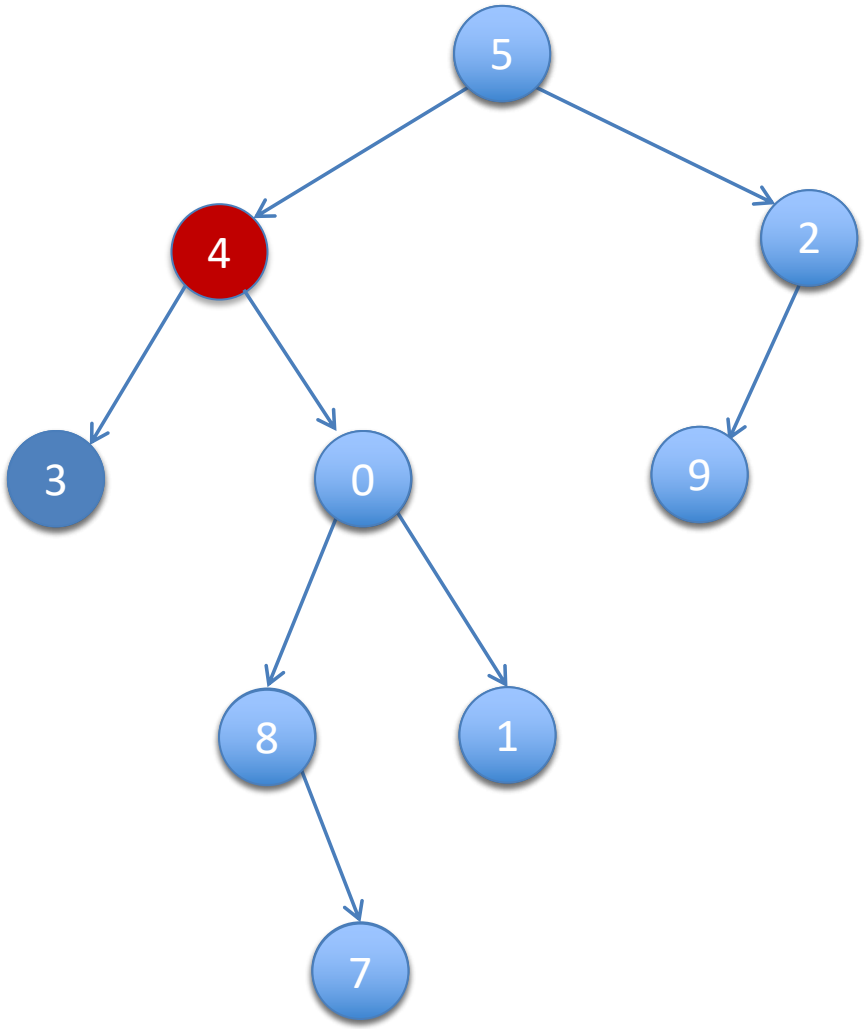


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

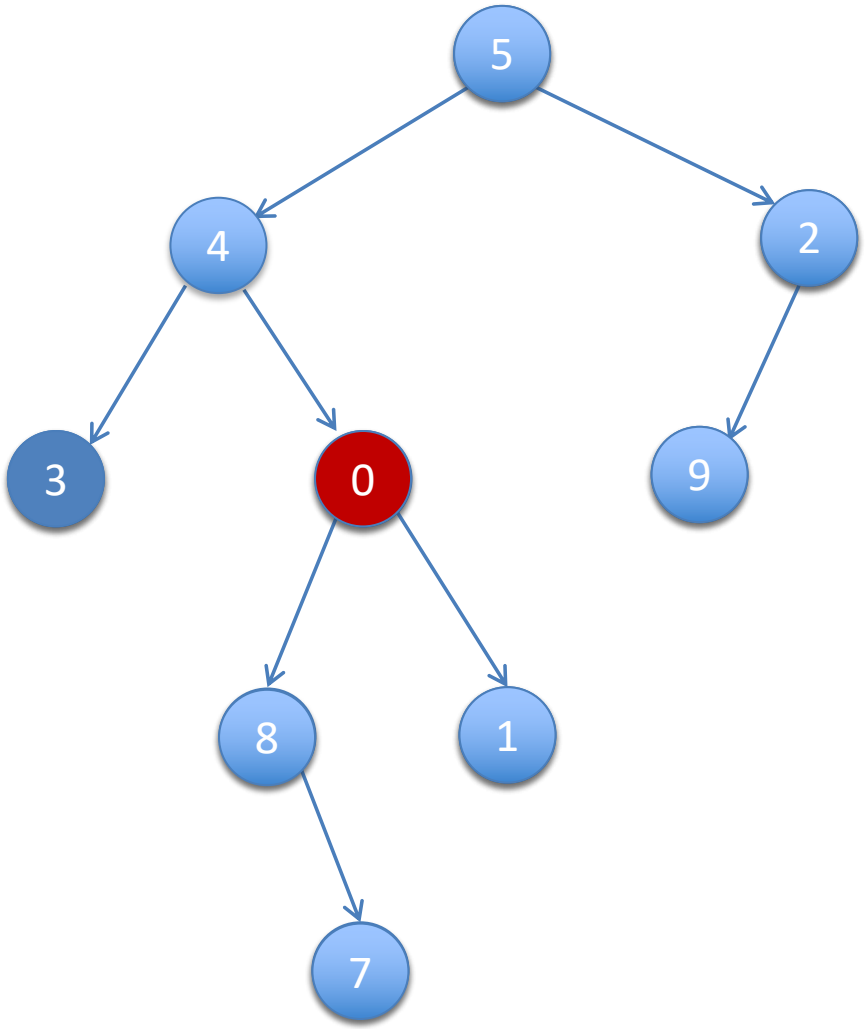


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

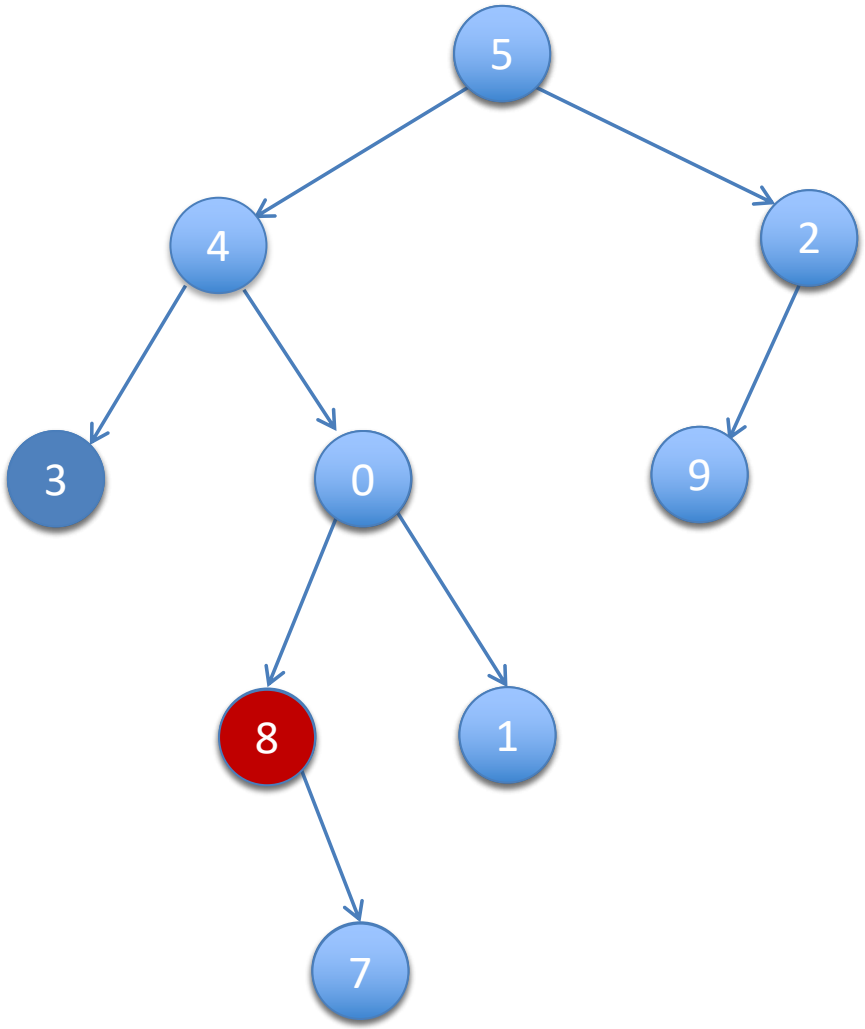


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

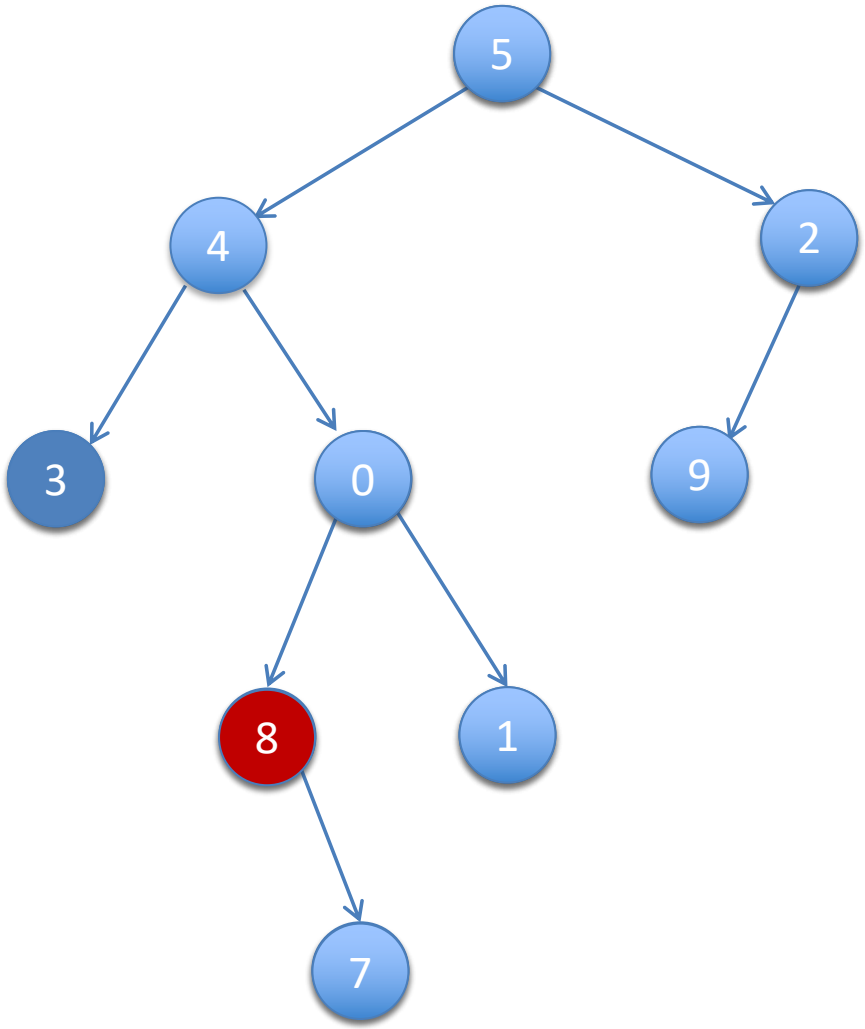


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

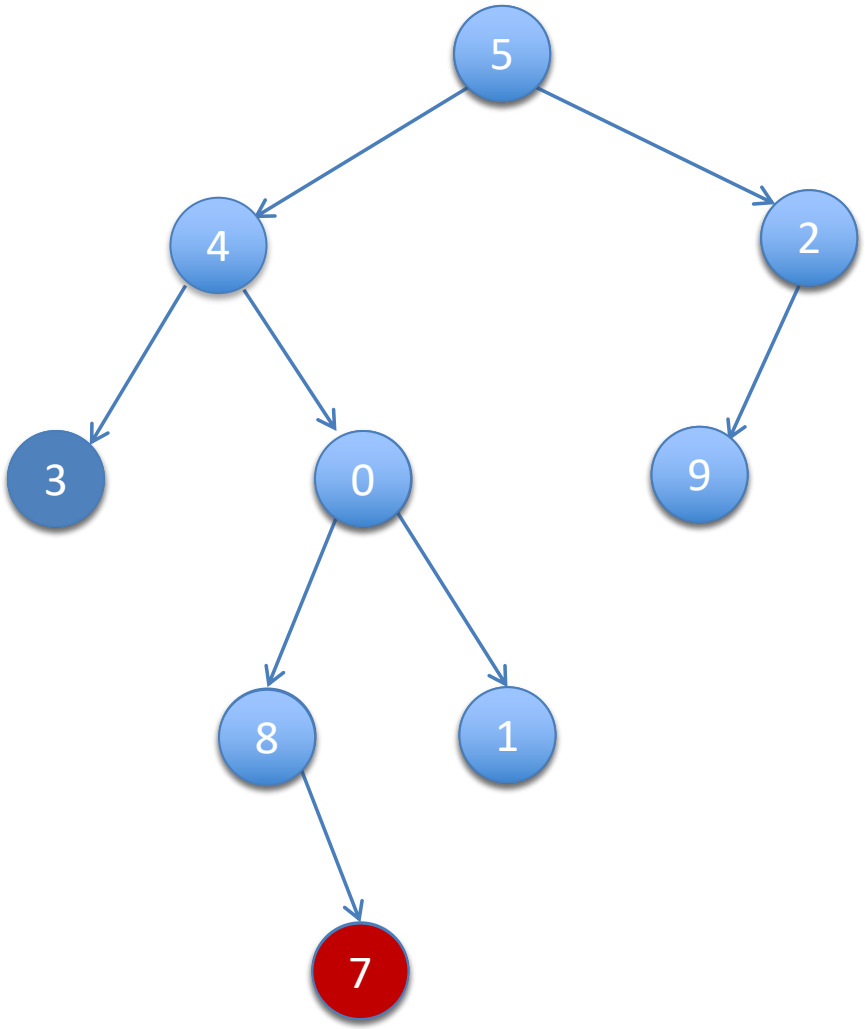


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

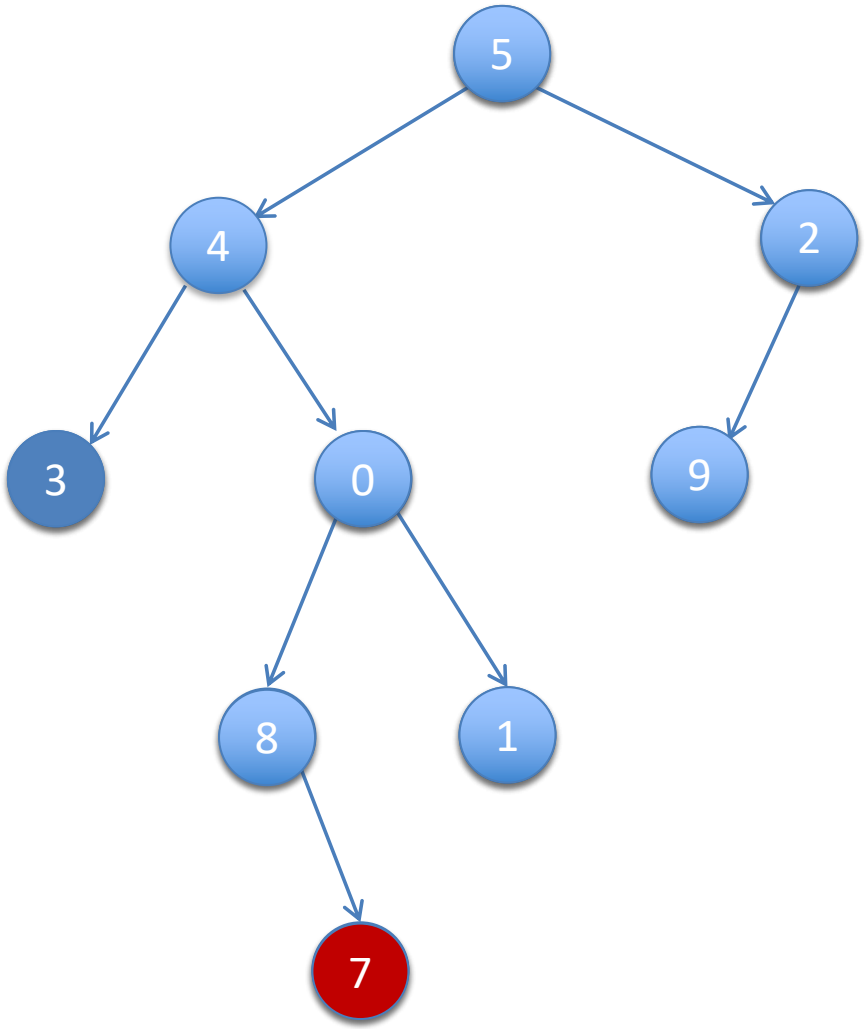


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

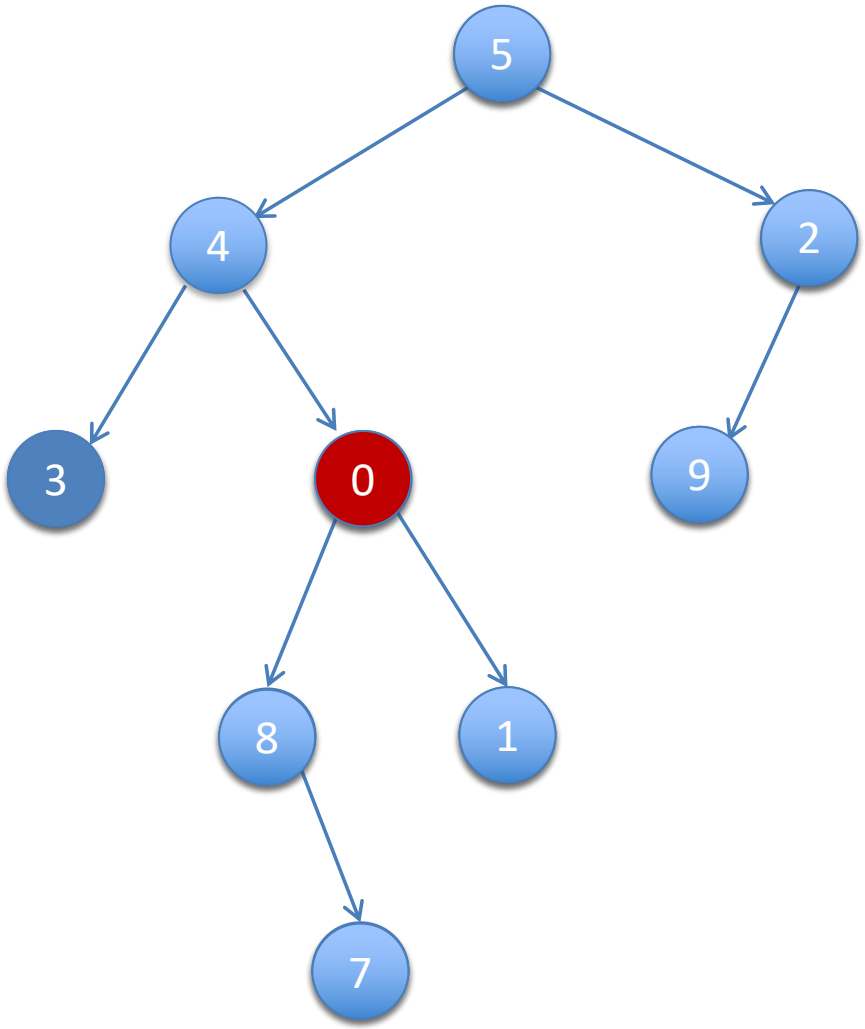


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

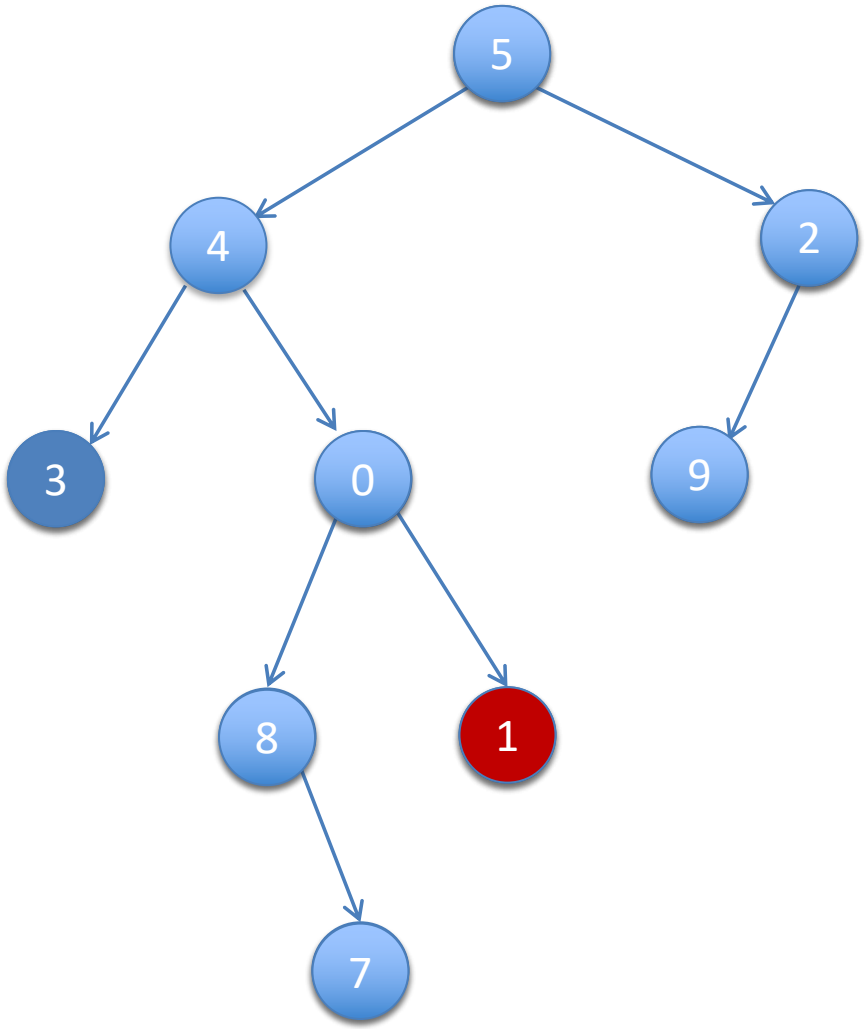


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

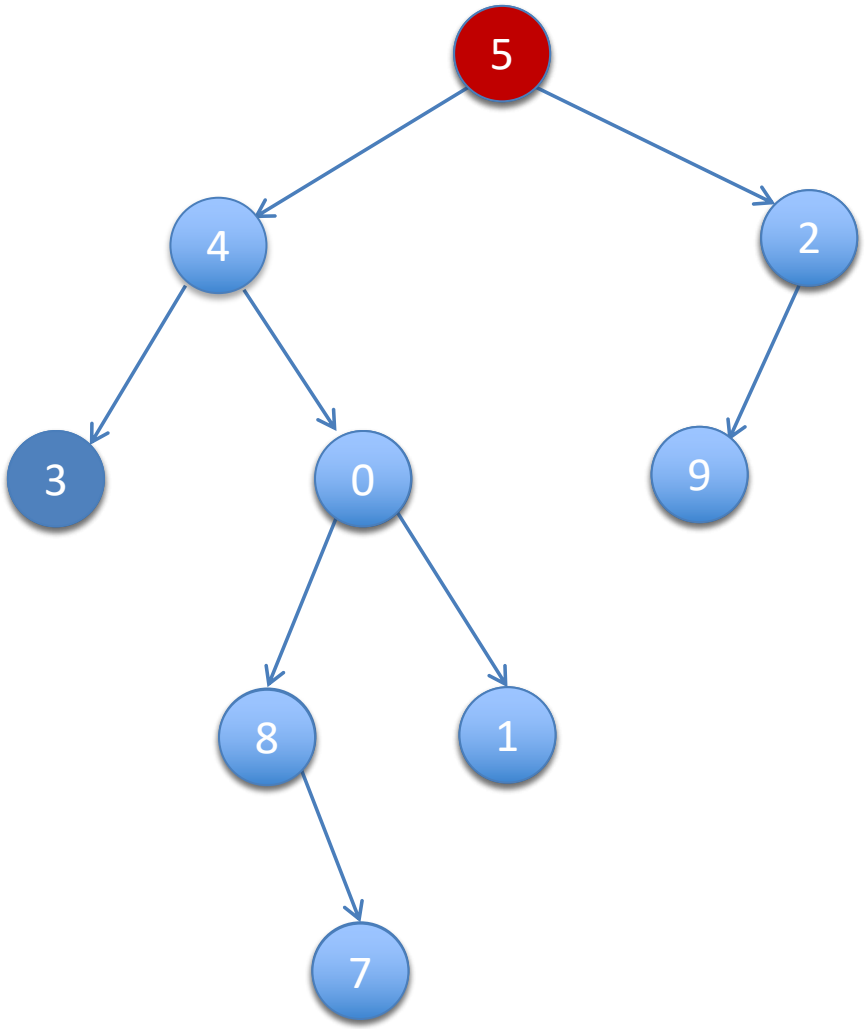


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

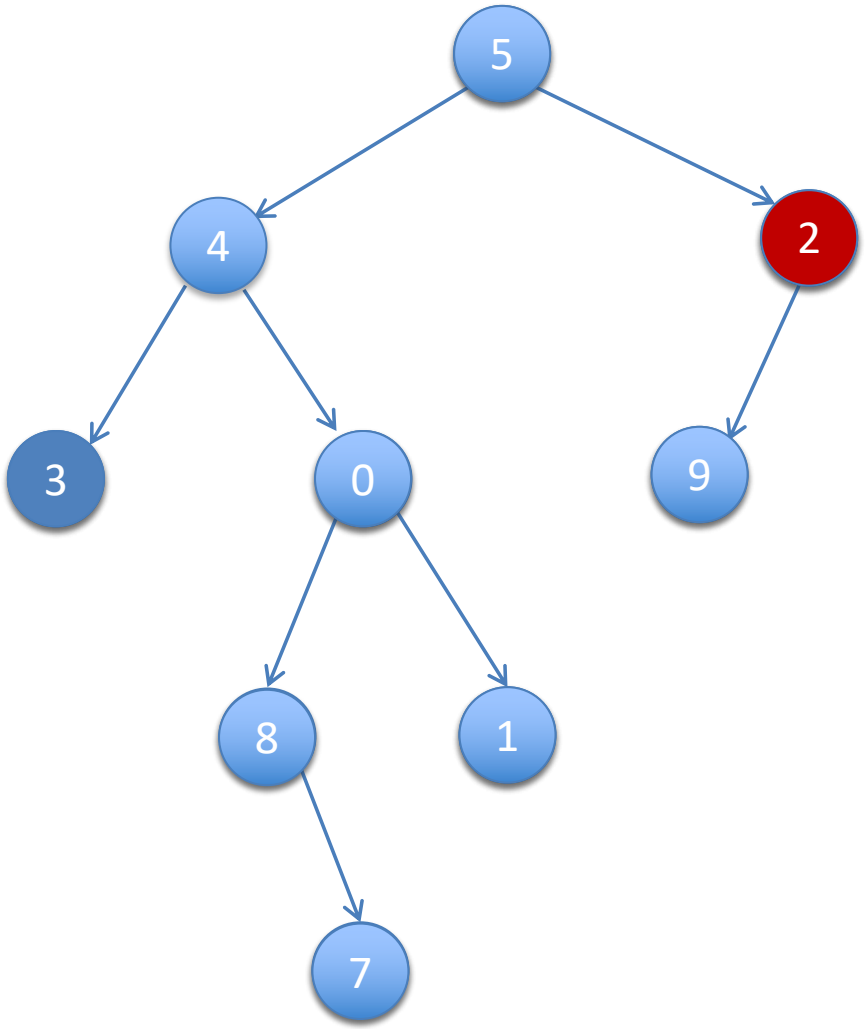


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

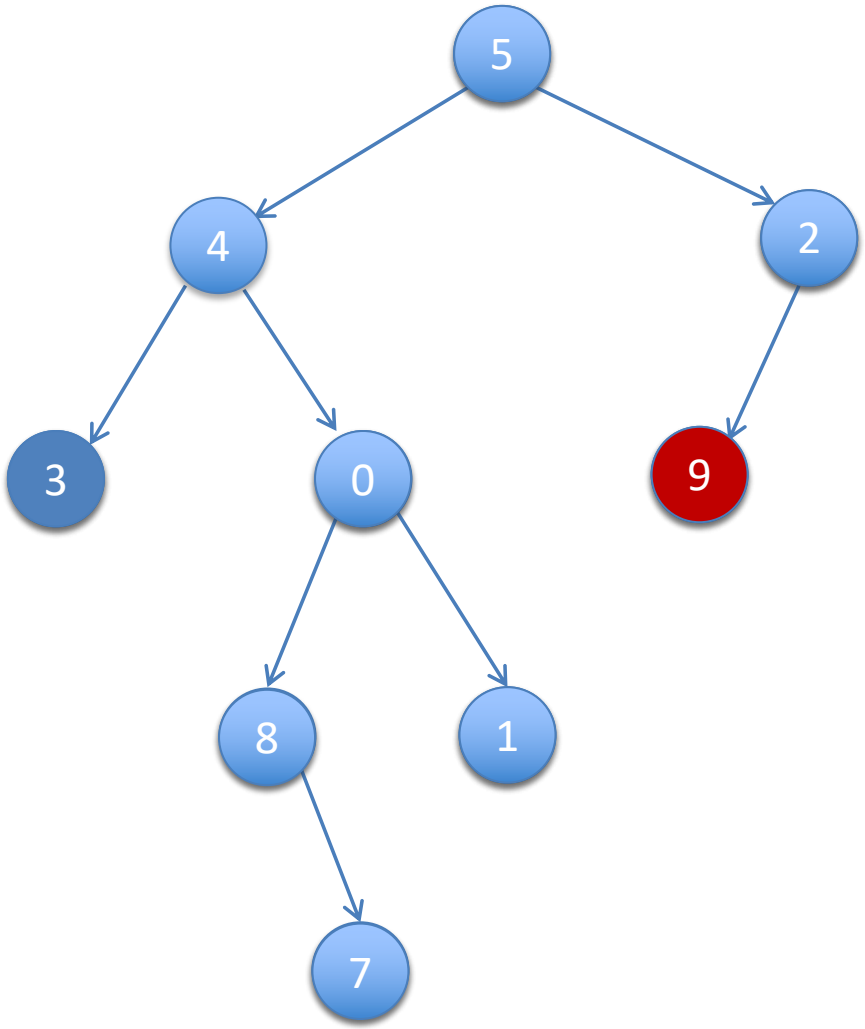


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

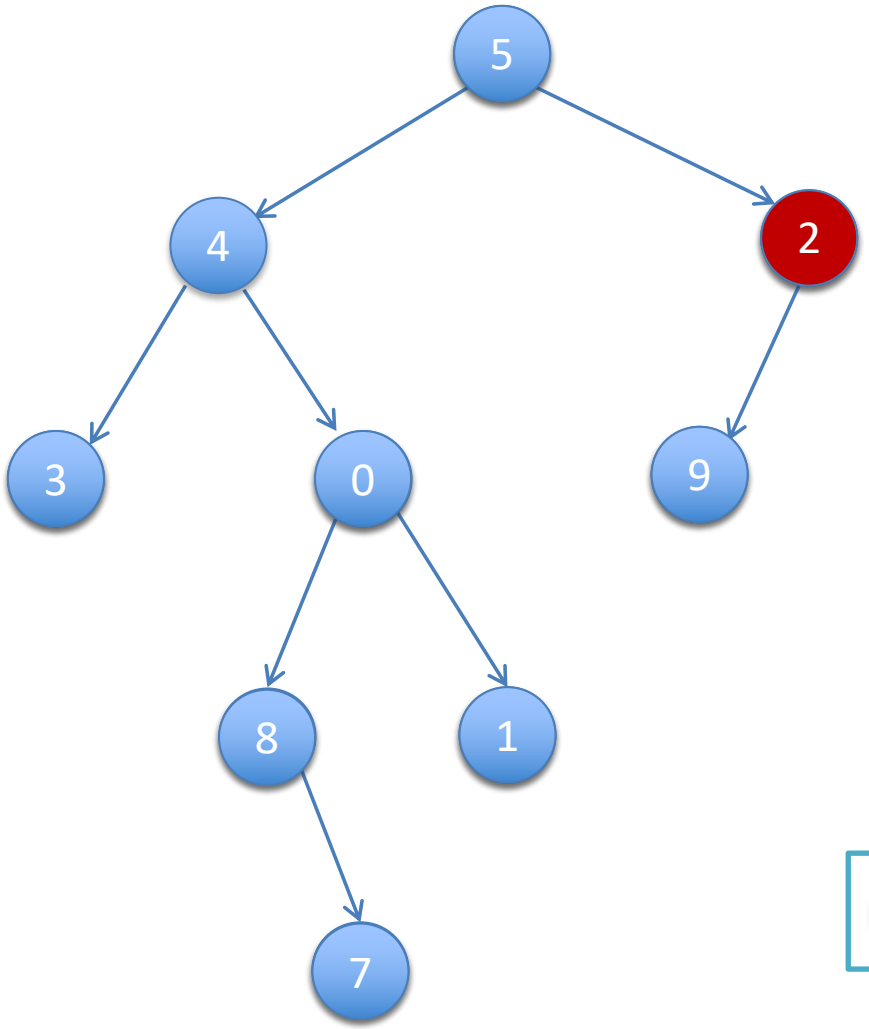


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)

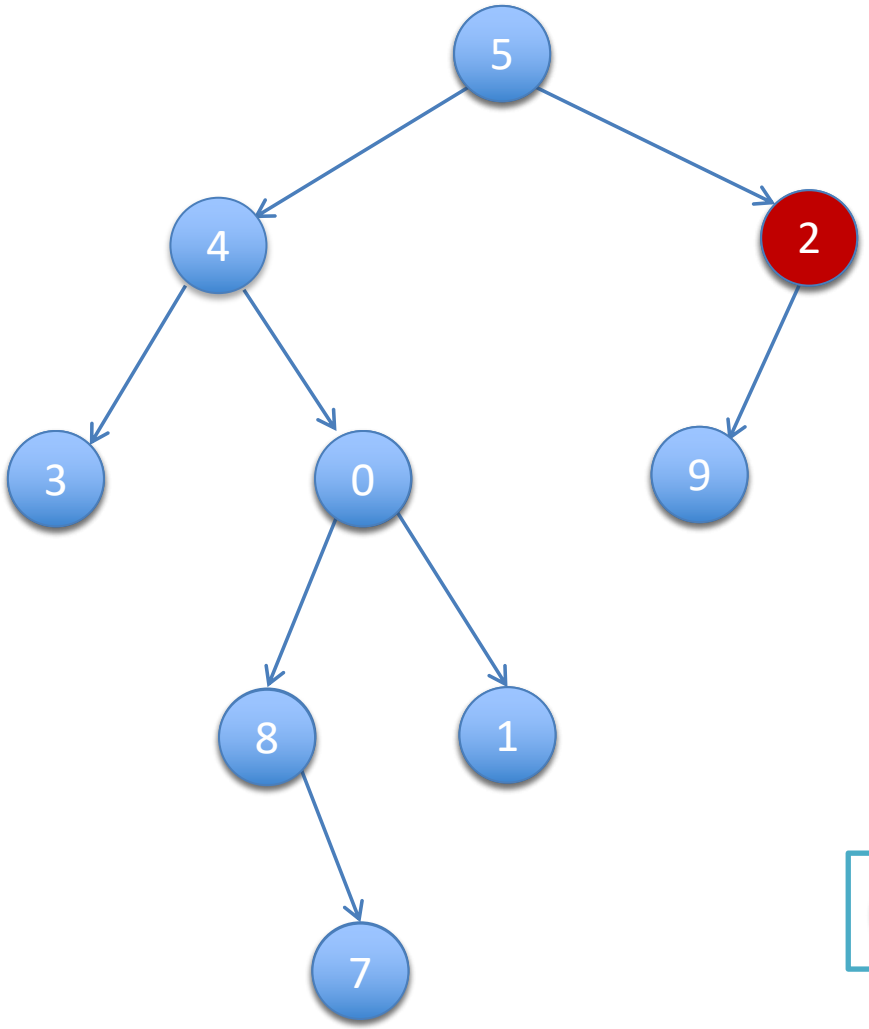


Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

How to do in-order traversal? (without recursion)



Using a Stack!
Just use a Stack instead of a Queue

A Stack

Output

Summary

We have covered:

- Definition of a tree data structure and its components
- LMC-RS vs 2-Children Tree
- Concepts of:
  - Root, internal, and leaf nodes
  - Parents, children, and siblings
  - Ancestors and descendants
  - Full vs Complete Tree
  - Tree Traversal

Acknowledgement

- Douglas Wilhelm Harder.
  - Thanks for making an excellent set of slides for ECE 250 *Algorithms and Data Structures* course

- Prof. Hung Q. Ngo:
  - Thanks for those beautiful slides created for CSC 250 (Data Structures) course at UB.

- Many of these slides are taken from these two sources.