CS 106 INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON

HAVERFORD COLLEGE

ADMIN

- Make use of chat!
  - Can ask/answer questions publicly or privately
- Questionnaire about online learning coming soon
- Lab tomorrow:
  - NO FORM, just join the zoom meeting
  - Will help with both Lab 4 and Lab 5
  - Priority given to core of the lab (i.e. not extra credit, extensions, etc)
  - Still fill out the google sheet to sign in

Remind me to RECORD

REVISED TA/OFFICE HOURS

Sunday 7-9pm (Juvia)

Monday 8-midnight (Steve)

Tuesday 11:30-12:30pm (Lizzie)

Tuesday 4:30-6pm (Sara)

Wednesday 8-midnight (Steve)

Thursday 11:30-12:30pm (Lizzie)

Thursday 9-11pm (Will)

Friday 8-10pm (Gareth)

Today/Tomorrow

Saturday 4-6pm (Will)

Saturday 8-10pm (Gareth)

MAR 26 OUTLINE

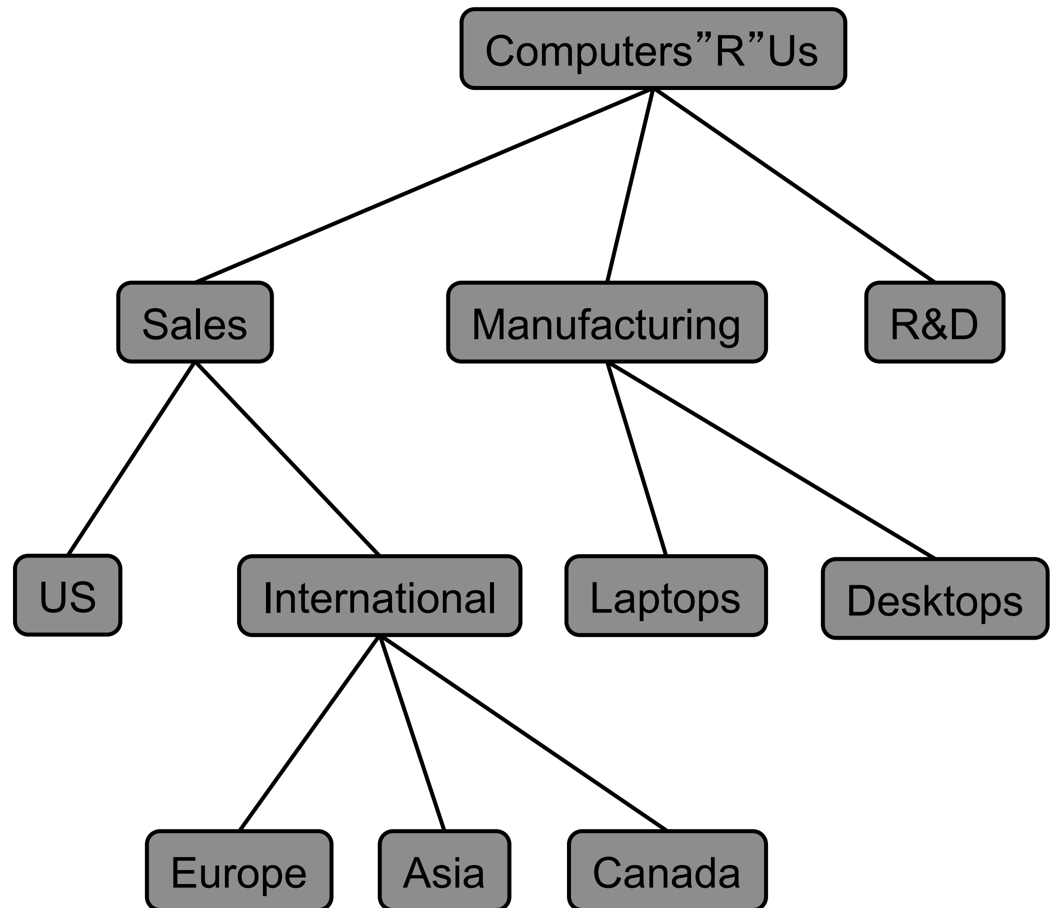
- Recap tree data structure
- Tree traversal practice
- Tree implementations
- Tree algorithms and operations

MAR 26 OUTLINE

- **Recap tree data structure**
- Tree traversal practice
- Tree implementations
- Tree algorithms and operations

TREE DATA STRUCTURE

- Trees are **acyclic graphs**
- Nodes/vertices have a **parent-child** relationship



TERMINOLOGY

Root: node with no parent (caveat, all nodes are roots of their subtree)

- A

Leaf node: node with no children:

- E, I, J, K, G, H, D

Internal node: node with at least one child

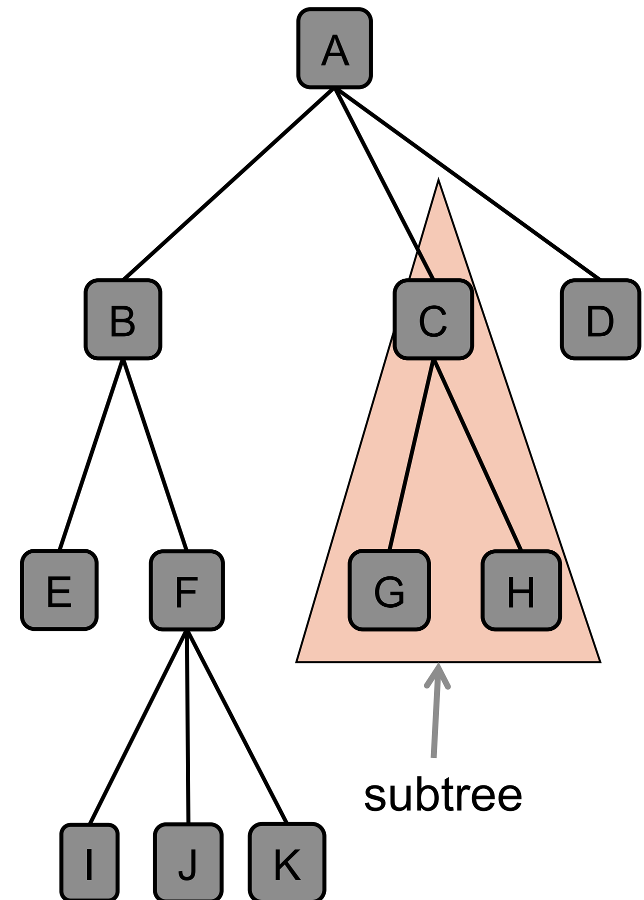
- A, B, C, F

Parent / Child relationships: two nodes connected by an edge. The node closer to the root is the parent.

- E.g., B (parent) and F (child)

Ancestor / Descendent relationships: ancestors of node X lie on the path from the root to X

- E.g., B (ancestor) and J (descendent)



TERMINOLOGY

Depth of a node: length of the path (num edges) from the root to that node

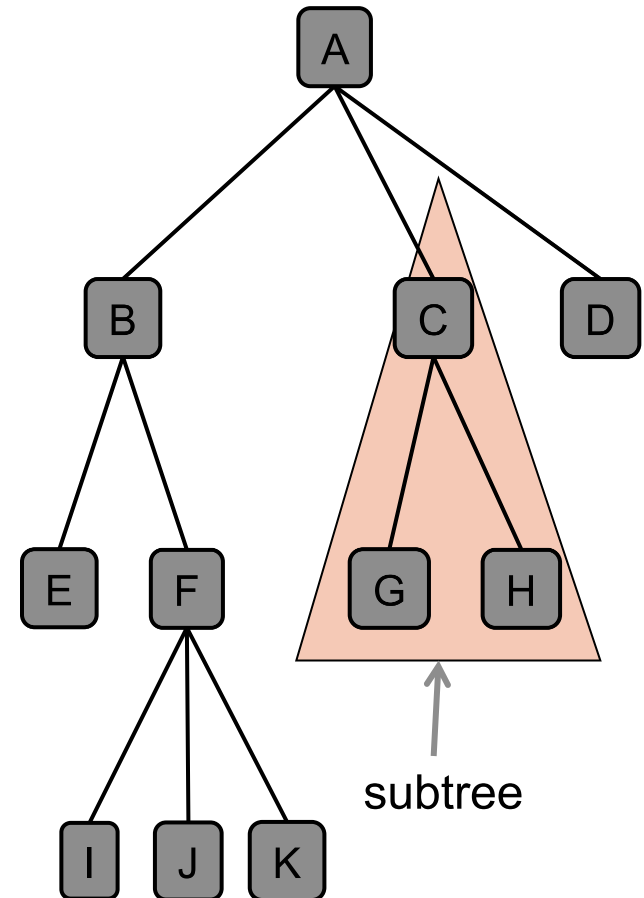
- e.g., depth of F = 2

Height: the maximum depth in the tree

- the height is 3

Subtree: a tree consisting of a node and its descendants

- the orange subtree with root C



If we allow more than one parent, could cause a cycle (see below)

Mute Stop Video Manage Participants 31 New Share Pause Share Annotate Remote Control More

ID: 928-616-103 Stop Share

TERMINOLOGY

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save

Depth of a node: length of the path (num edges) from the root to that node

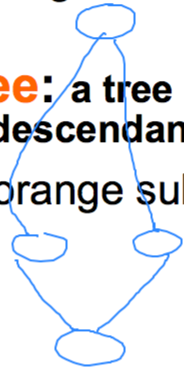
- e.g., depth of F = 2

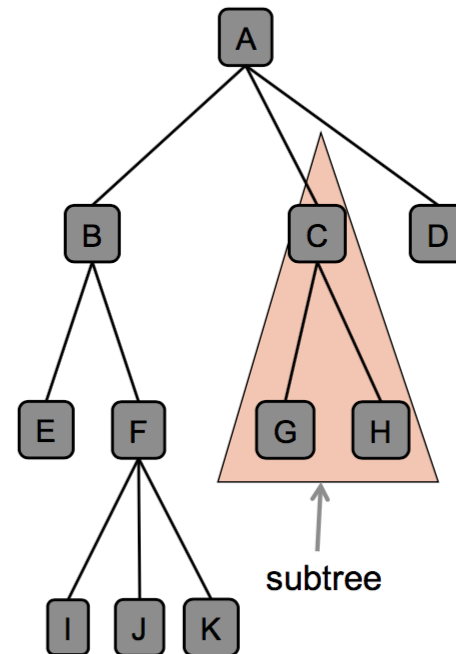
Height: the maximum depth in the tree

- the height is 3

Subtree: a tree consisting of a node and its descendants

- the orange subtree with root C



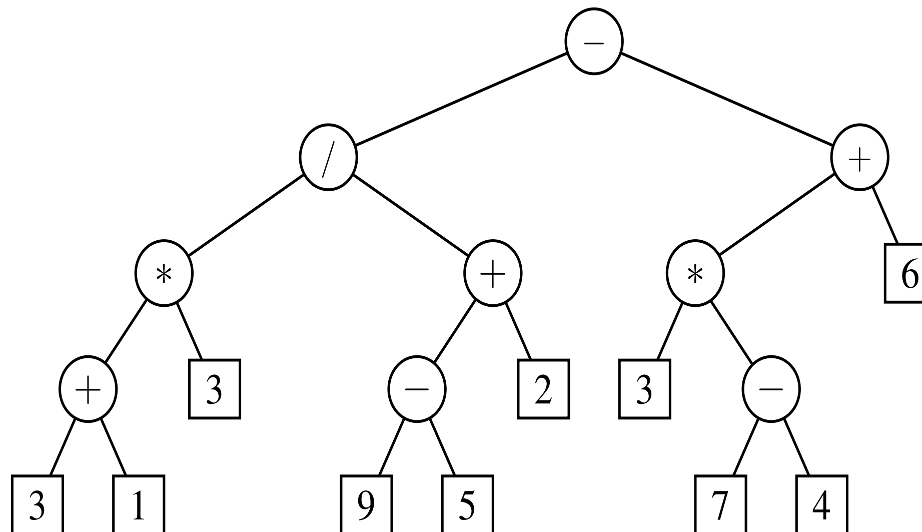


BINARY TREE

An ordered tree with every node having at most two children – left and right

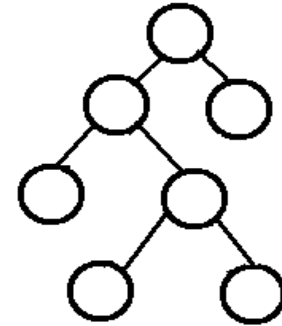
Recursive definition:

- base case: empty tree
- recursion: root with two subtrees



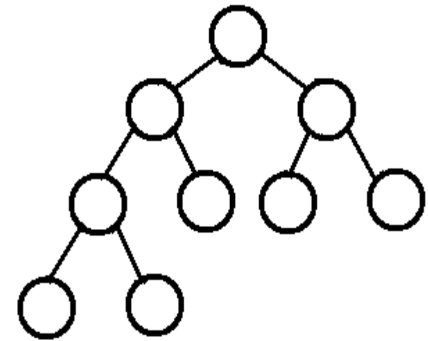
TYPE OF BINARY TREES

A binary tree is **proper** (or **full**) if each node has zero or two children

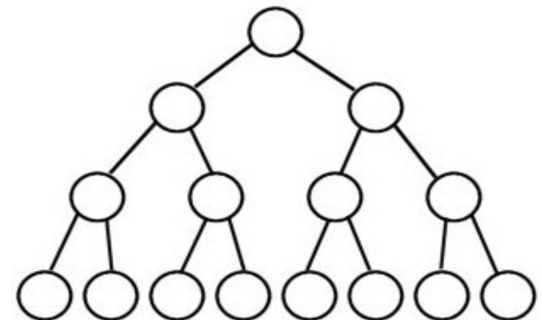


A binary tree is **complete** if every level (except possibly the last) is filled

Edit: if the last level is not filled, nodes should be as far left as possible!

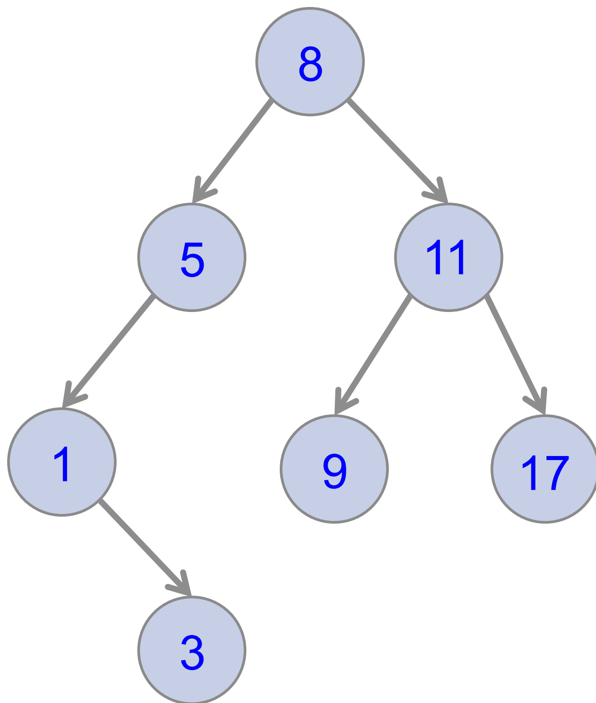


If a complete binary tree is filled at every level, it is **perfect**



INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

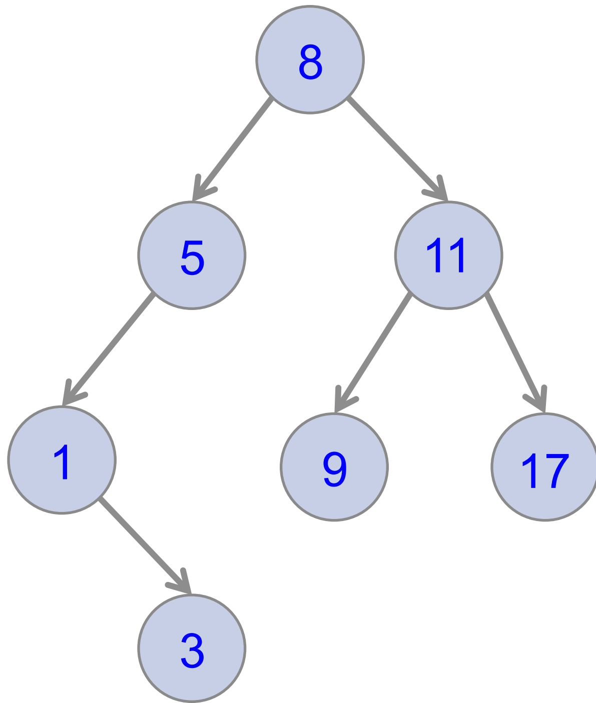


- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

d = number of levels or number of comparisons

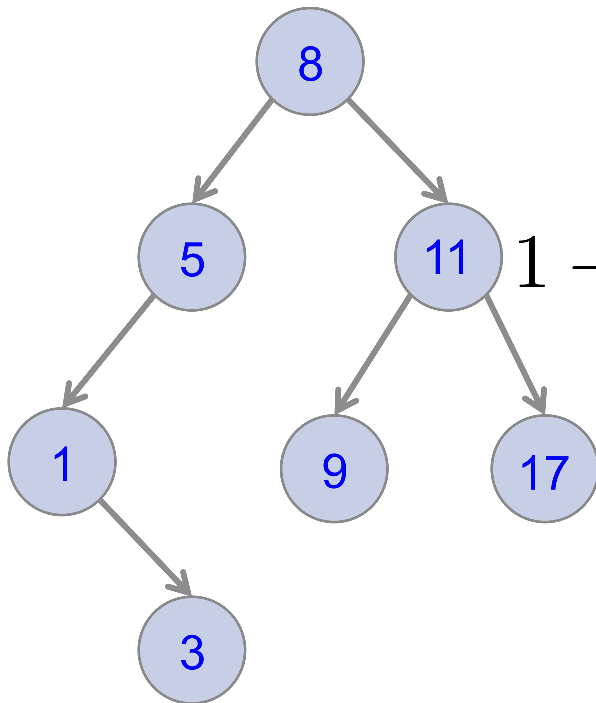


- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

d = number of levels or number of comparisons



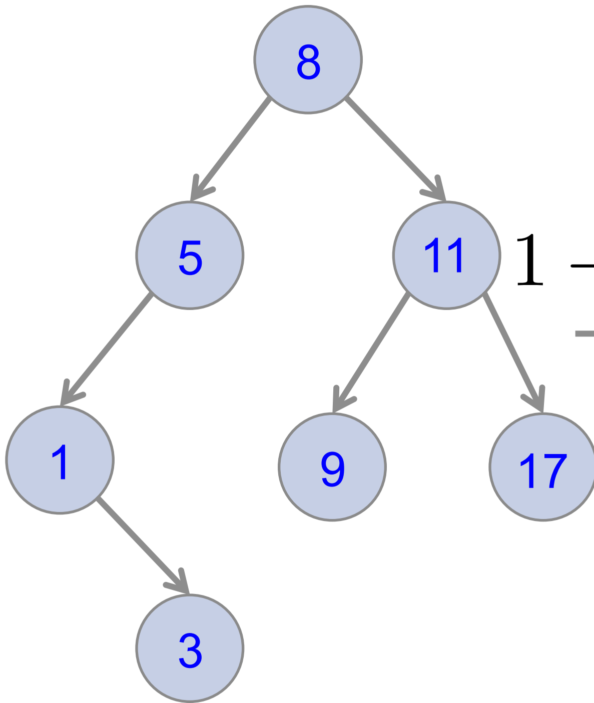
$$1 + 2 + 2^2 + 2^3 + \cdots + 2^{d-1} = n$$

- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

d = number of levels or number of comparisons



$$2 + 2^2 + 2^3 + \cdots + 2^{d-1} + 2^d = 2n$$

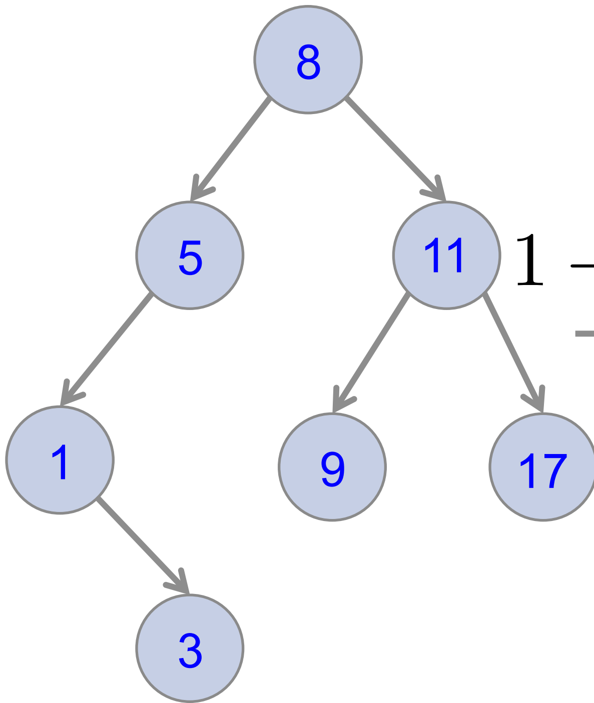
$$1 + 2 + 2^2 + 2^3 + \cdots + 2^{d-1} = n$$

- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

d = number of levels or number of comparisons



$$2 + 2^2 + 2^3 + \cdots + 2^{d-1} + 2^d = 2n$$

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^{d-1} = n$$

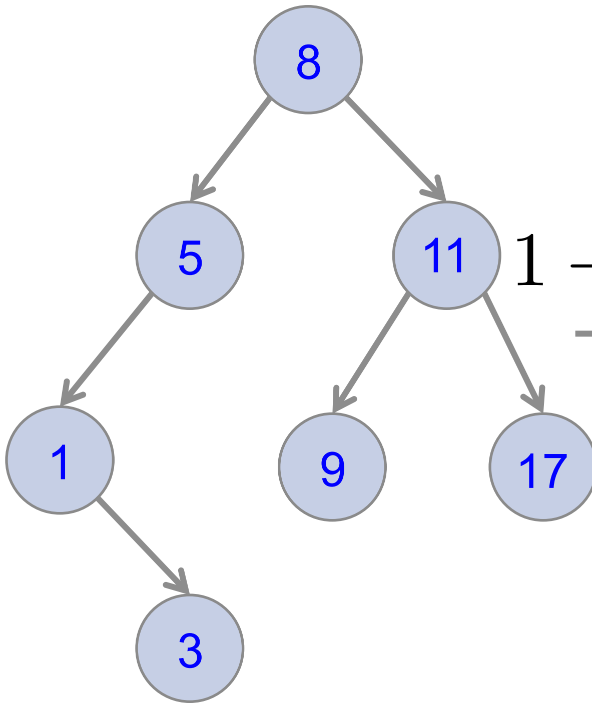
$$2^d - 1 = n$$

- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

d = number of levels or number of comparisons



$$2 + 2^2 + 2^3 + \cdots + 2^{d-1} + 2^d = 2n$$

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^{d-1} = n$$

$$2^d - 1 = n$$

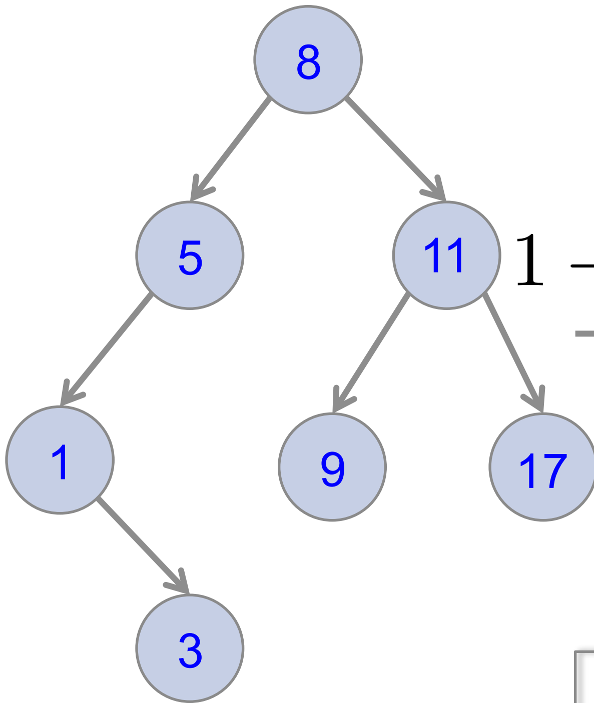
Therefore: $$d \approx \log(n)$$

- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

INSERTION SORT WITH TREES

Input: {8, 11, 5, 17, 1, 9, 3}

d = number of levels or number of comparisons



$$2 + 2^2 + 2^3 + \cdots + 2^{d-1} + 2^d = 2n$$

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^{d-1} = n$$

$$2^d - 1 = n$$

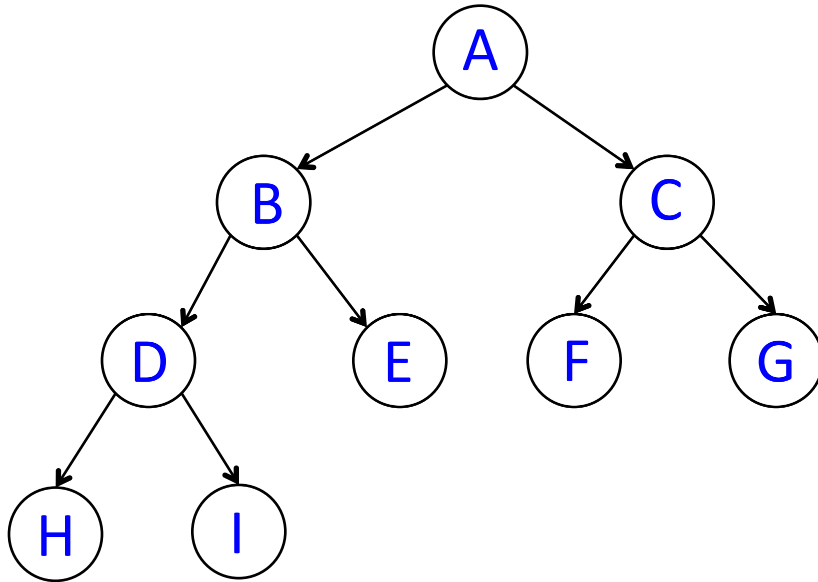
Therefore: $d \approx \log(n)$

Overall runtime: $O(n \log n)$

- No duplicates (for now)
- Everything to the left of each root is less than root data
- Everything to the right of each root is greater than root data

TREE TRAVERSALS: IN-ORDER

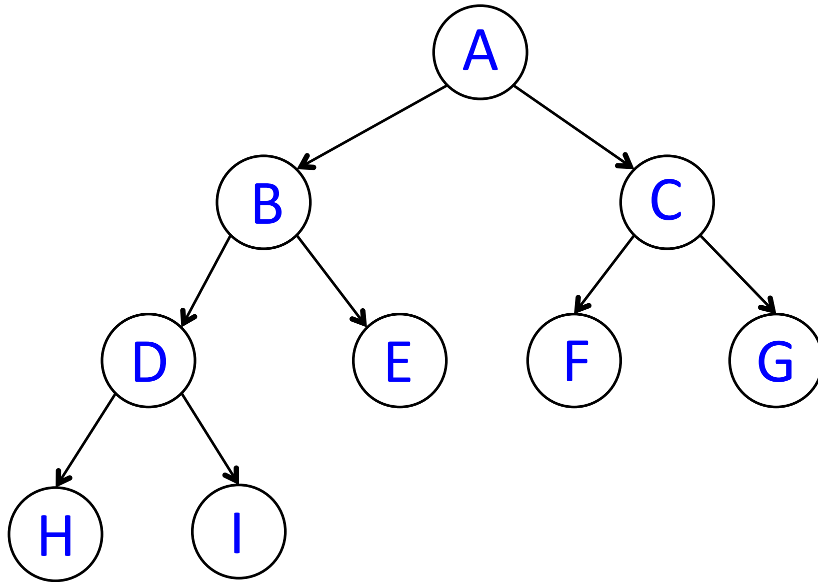
left – root – right



H D I B E A F C G

TREE TRAVERSALS: IN-ORDER

left – root – right

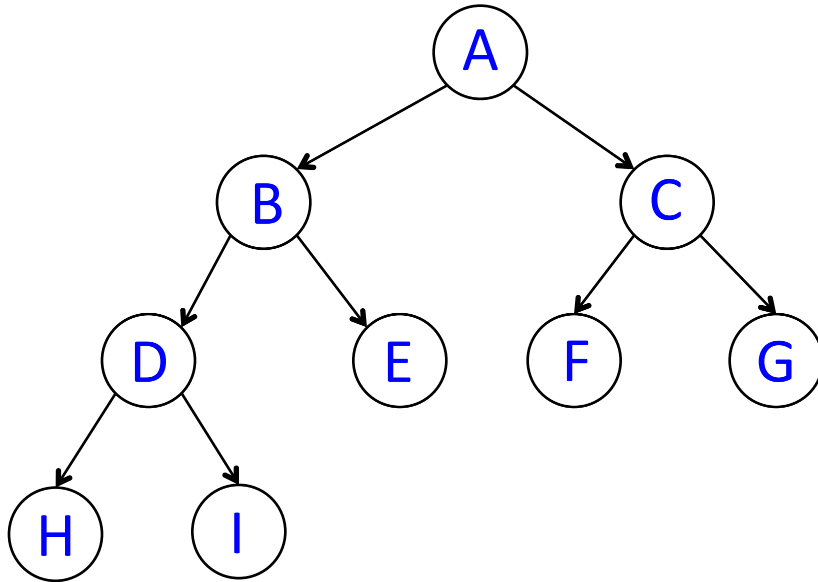



```
inOrder(root):
  if root != null:
    inOrder(root.left)
    visit(root)
    inOrder(root.right)
```

H D I B E	A	F C G

TREE TRAVERSALS: PRE-ORDER

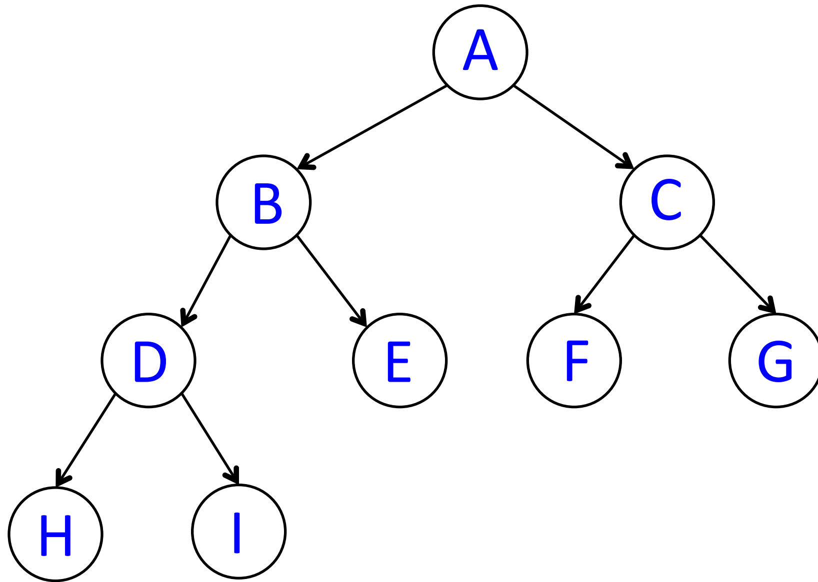
root – left – right



A | B D H I E | C F G

TREE TRAVERSALS: POST-ORDER

left – right – root

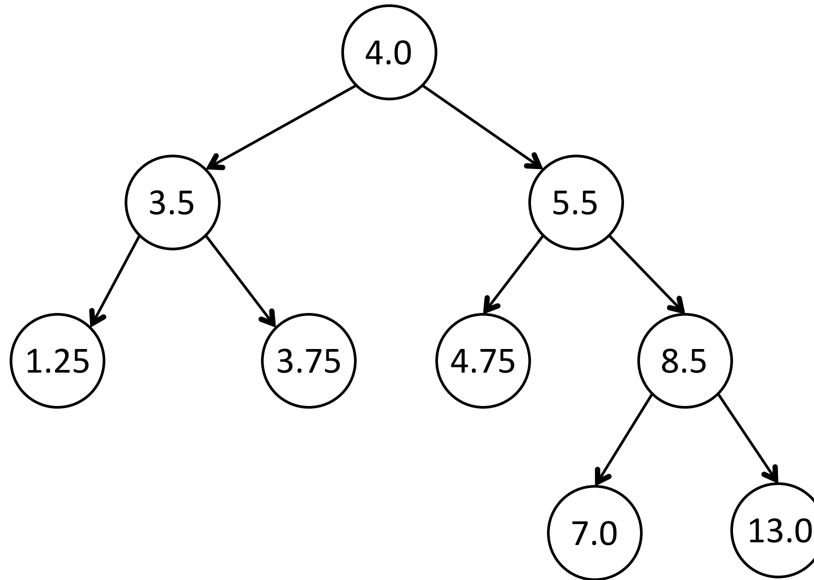


H I D E B F G C A

MAR 26 OUTLINE

- Recap tree data structure
- Tree traversal practice
- Tree implementations
- Tree algorithms and operations

HANDOUT (PAIR EXERCISE)



* Actually not complete! (see edit to slide with these definitions)
* But it is full
* It also has the property that nodes to the left of each root are less than the root and nodes to the right of each root are greater than the root.

HANDOUT (PAIR EXERCISE)

full &
complete

4.0
3.5
5.5
1.25
3.75
4.75
8.5
5
7.0
13.0
28
15
72
51
100

all right!

in:
1.25 3.5 3.75 4.0 5.5

pre:
4.0 3.5 5.5

post
3.5 5.5 4.0

MAR 26 OUTLINE

- Recap tree data structure
- Tree traversal practice
- Tree implementations
- Tree algorithms and operations

INTERFACE (NOT THE ONLY ONE)

```
public interface BinaryTree<E extends
Comparable<E>> {

  E getRootElement();

  int size();

  boolean isEmpty();

  void insert(E element);

  boolean contains(E element);

  boolean remove(E element);

  String toStringInOrder();
  String toStringPreOrder();
  String toStringPostOrder();

}
```

Mute Stop Video Manage Participants 31 New Share Pause Share Annotate Remote Control More
ID: 928-616-103 Stop Share
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save

INTERFACE (NOT THE ONLY ONE)

```
public interface BinaryTree<E extends
Comparable<E>> {
   E getRootElement();
   int size();
   boolean isEmpty();
   void insert(E element);
   boolean contains(E element);
   boolean remove(E element);
   String toStringInOrder();
   String toStringPreOrder();
   String toStringPostOrder();
}
```

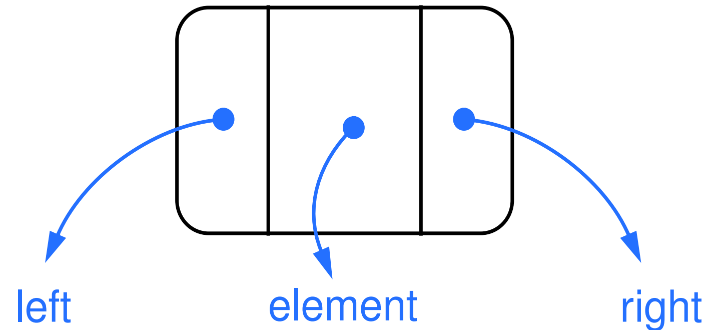
Lab 5 and recursive

IMPLEMENTATION

Fully recursive data structure!

A tree is kind of like a Node...

* data
* left (also a BinaryTree)
* right (also a BinaryTree)



MAR 26 OUTLINE

- Recap tree data structure
- Tree traversal practice
- Tree implementations
- Tree algorithms and operations

REMOVE

```
boolean remove(E element);
```

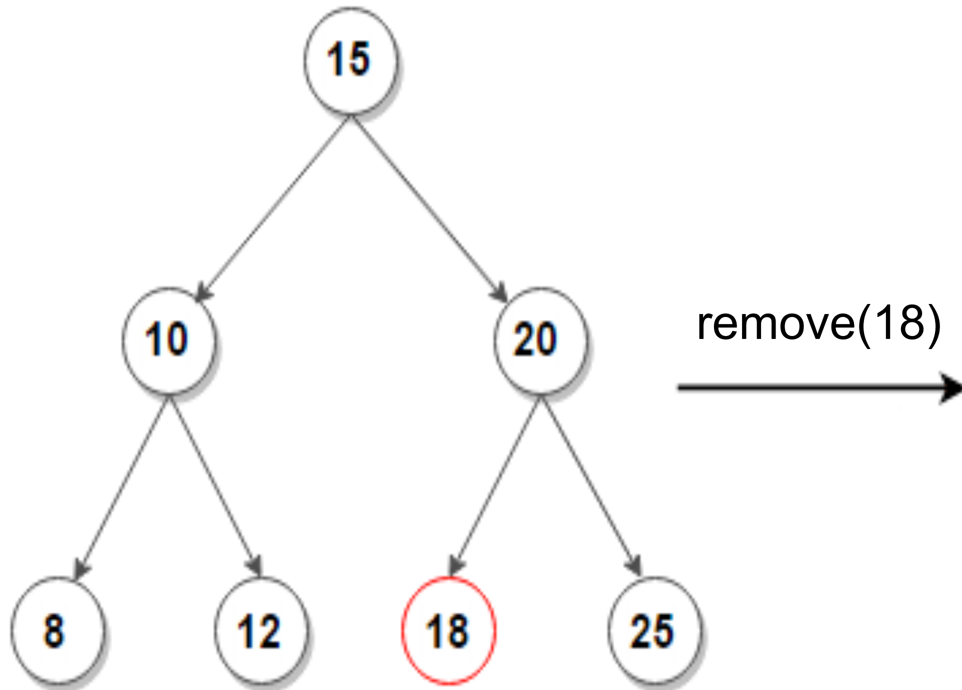
returns true if element existed and was removed and false otherwise

Cases

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

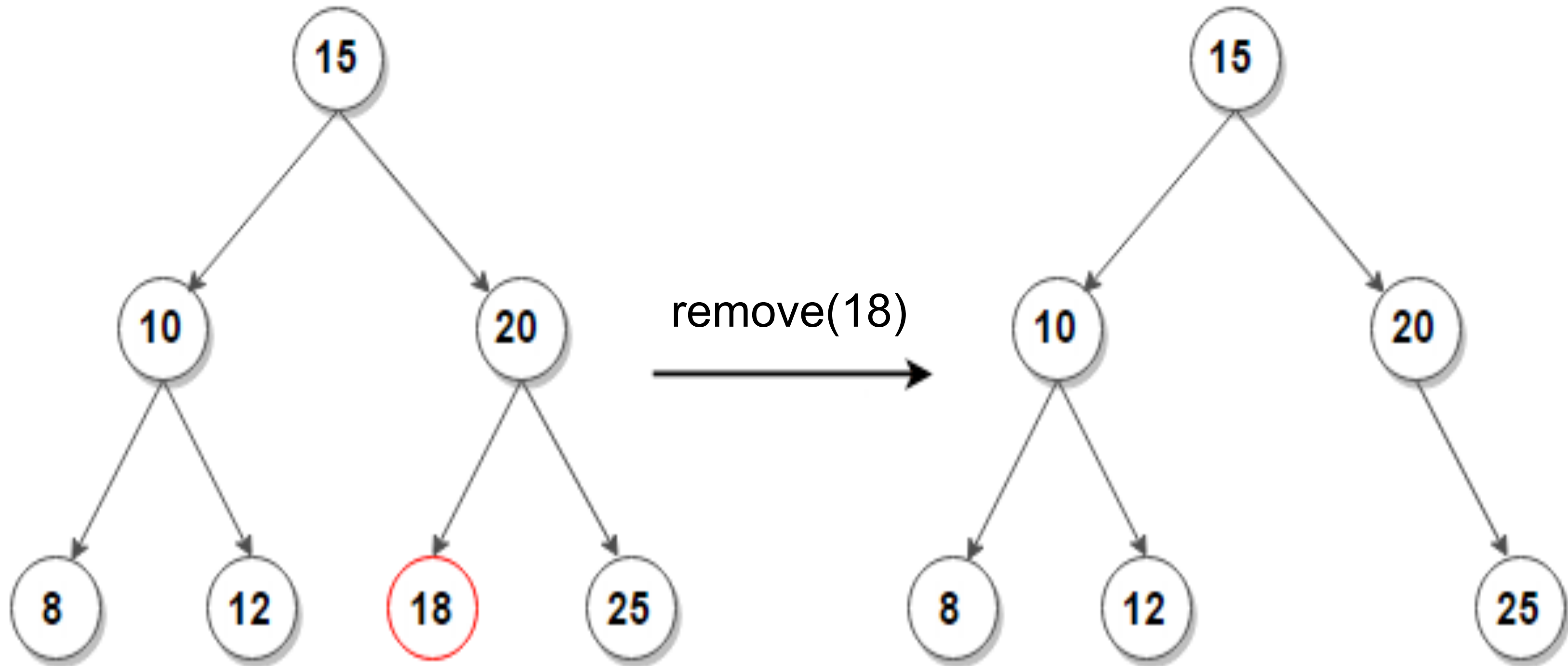
LEAF

Just delete



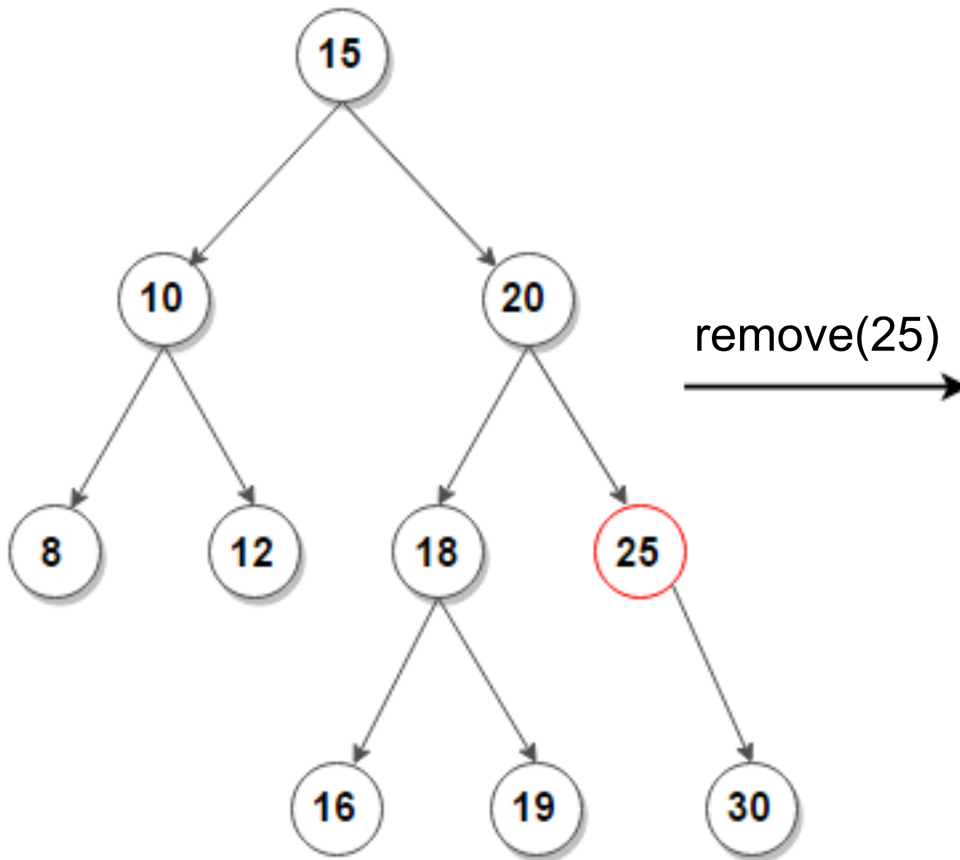
LEAF

Just delete



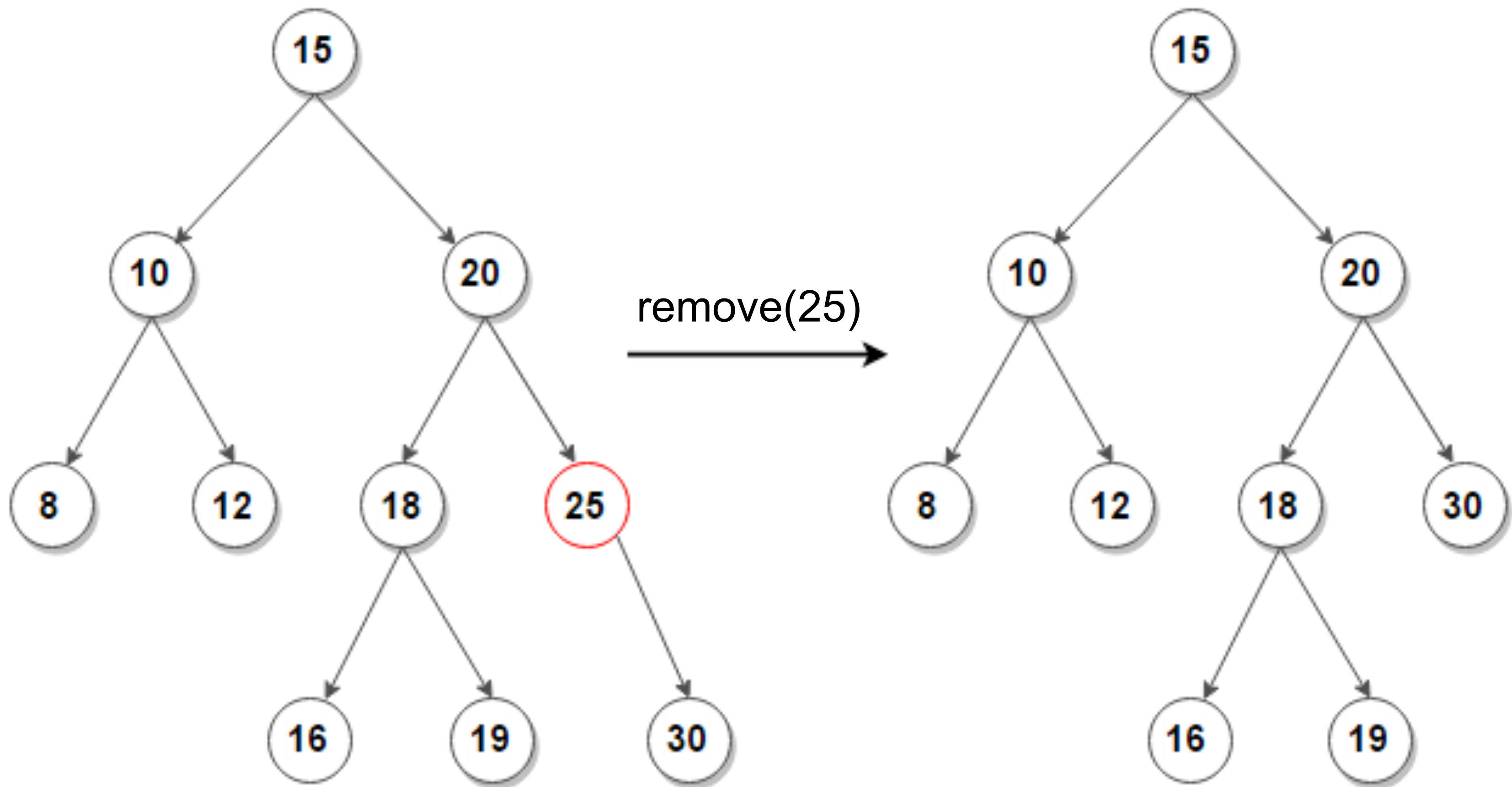
ONE CHILD

Replace with child



ONE CHILD

Replace with child



TWO CHILDREN

Replace with in-order predecessor or in-order successor

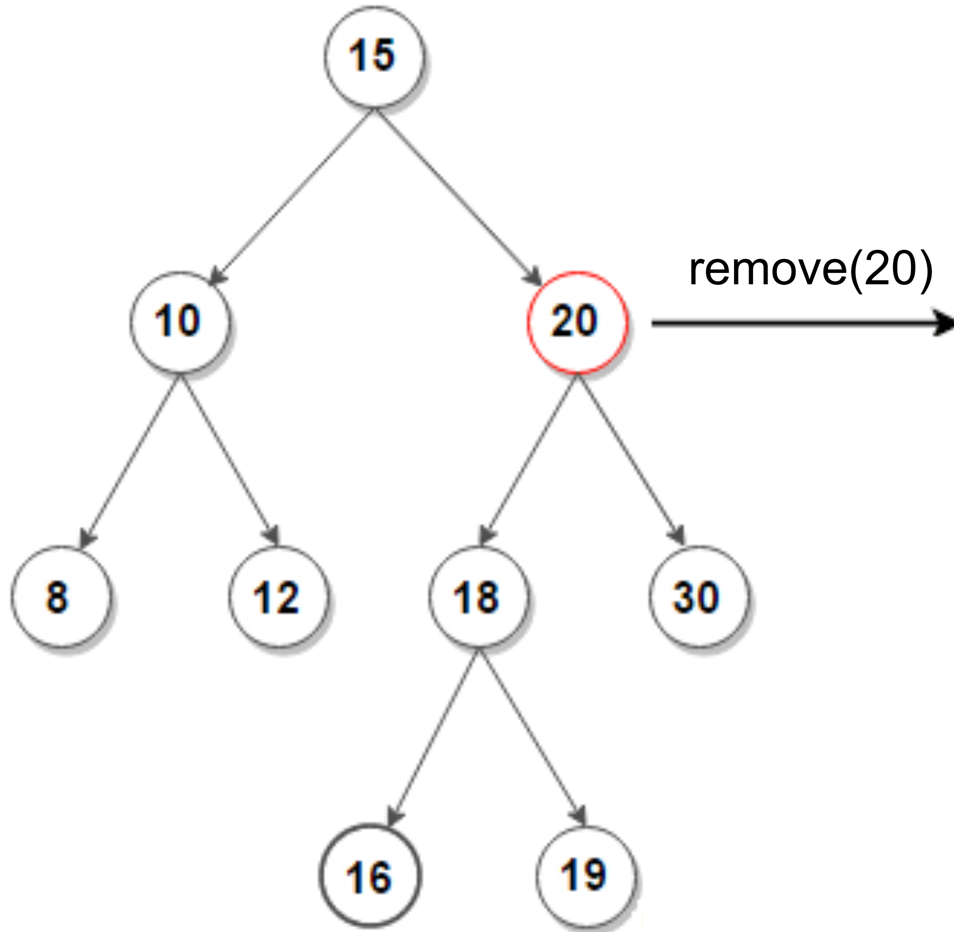
in-order predecessor

- rightmost child in left subtree
- max-value child in left subtree

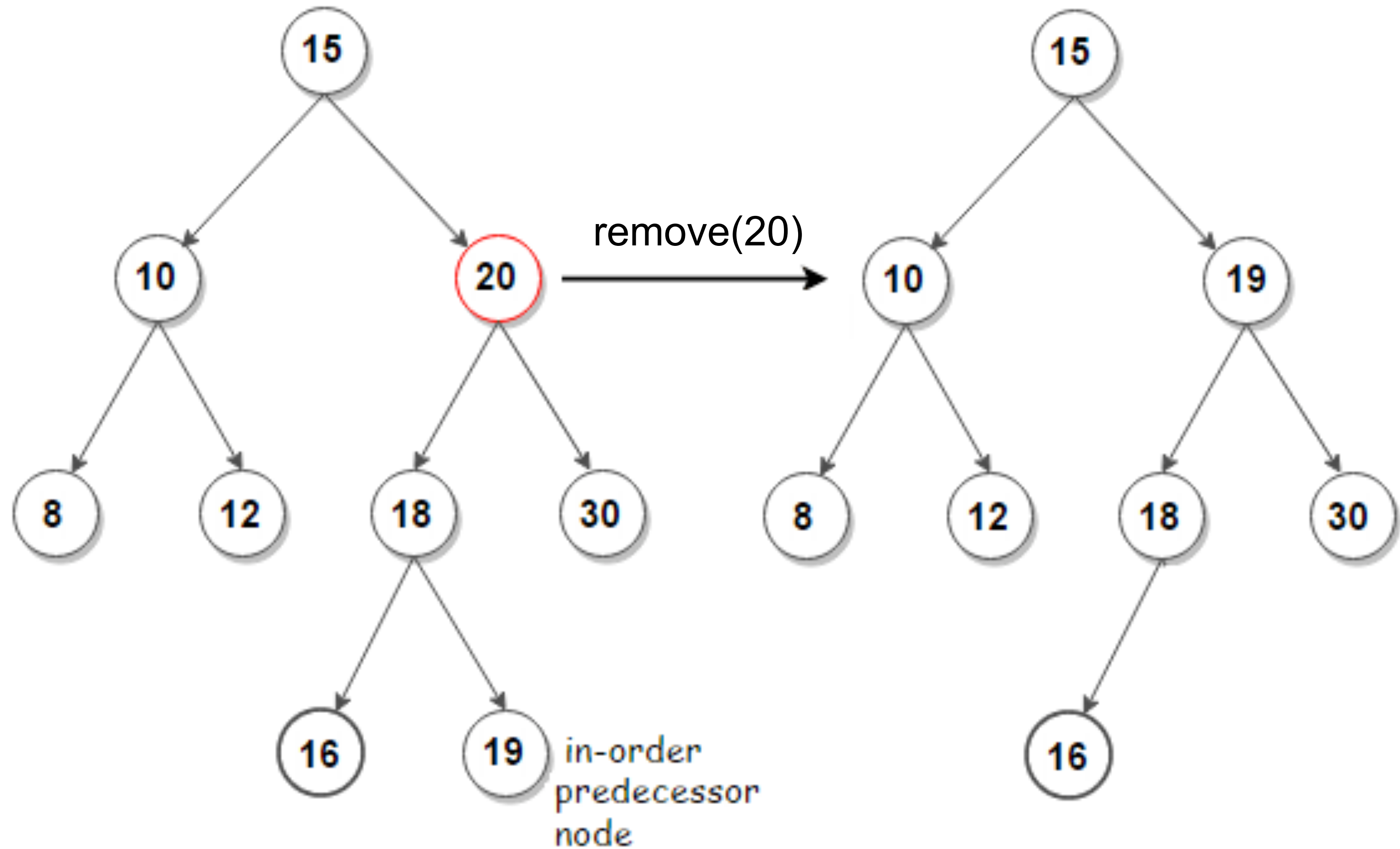
in-order successor

- leftmost child in right subtree
- min-value child in right subtree

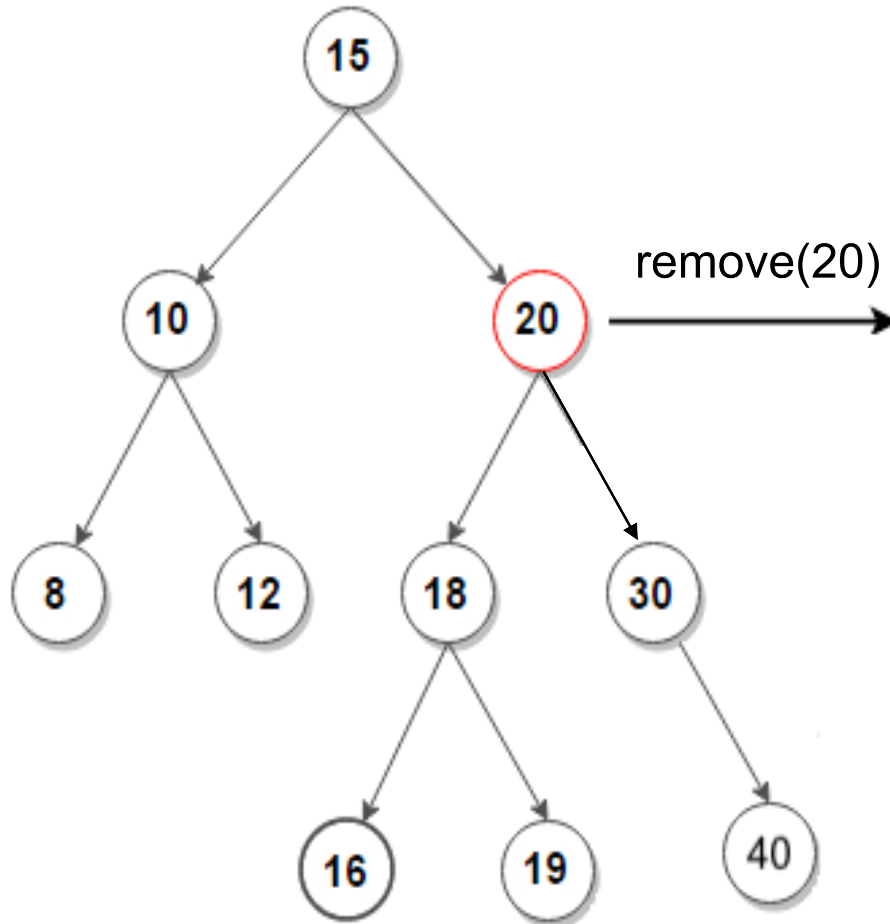
REPLACE WITH PREDECESSOR



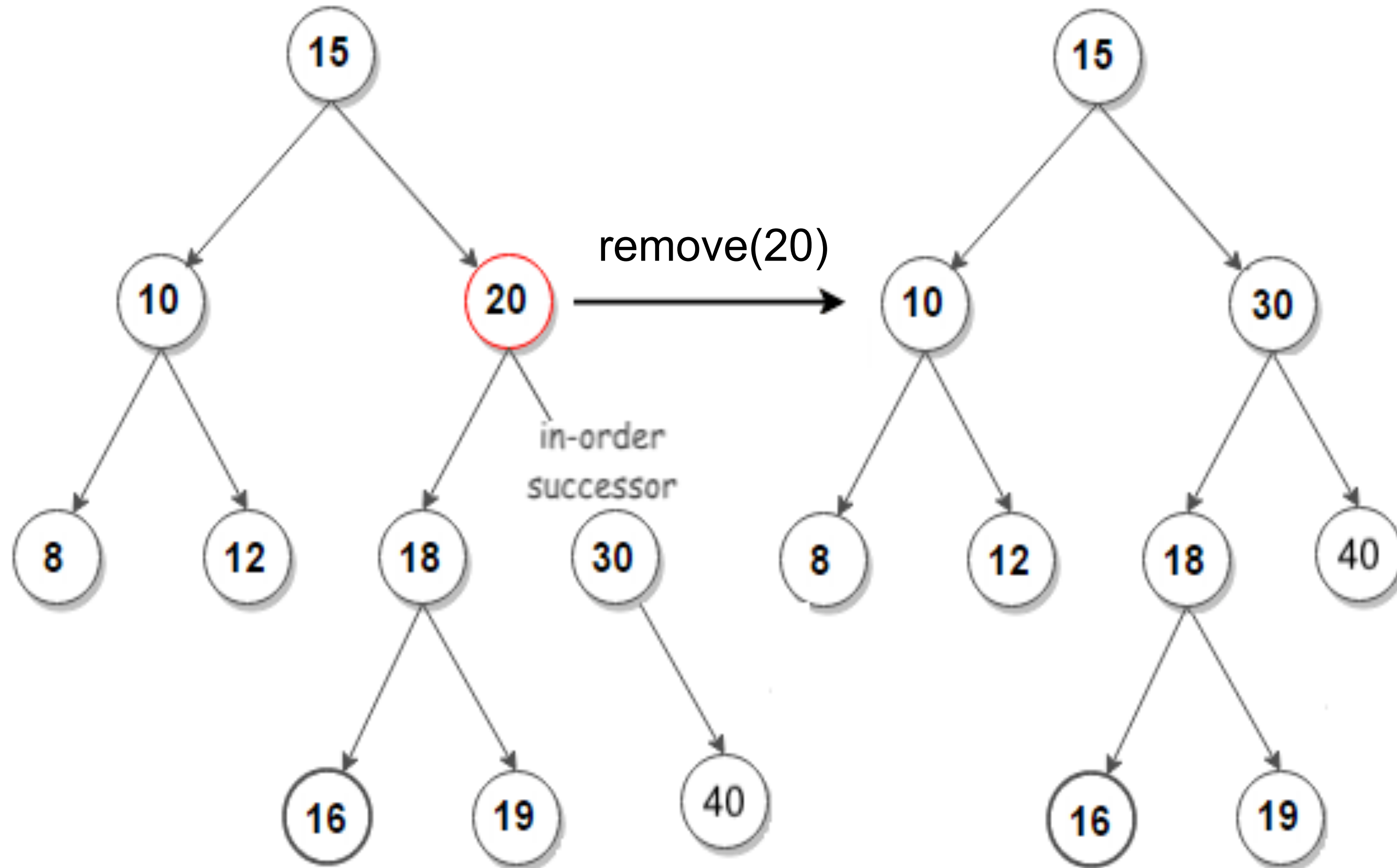
REPLACE WITH PREDECESSOR



REPLACE WITH SUCCESSOR



REPLACE WITH SUCCESSOR



Note: this is a special case when successor is also child (replace entire node, just just data)

HUFFMAN CODING

Goal: convert words/letters/symbols to binary code

HUFFMAN CODING

Goal: convert words/letters/symbols to binary code

freq:	0.5	0.3	0.2
word:	a	the	and
code:	0	1	10

1010

and, the, a

and, and

HUFFMAN CODING

Outline:

Step 1: sort by frequency and make a queue (low -> high)

Step 2: merge first two elements off the queue to form a root, then add root back into queue (respecting frequency)

Repeat until queue is empty.

Step 3: label resulting branches of tree with 0's for left branches and 1's for right branches

HUFFMAN CODING

Example

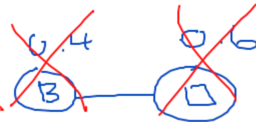
HUFFMAN CODING

Example

	A	B	C	D	
code	110	0	111	10	
frequency	0.1	0.4	0.2	0.3	= 1.0
alphabet	A	B	C	D	

① sort

② merge



③ code