Binary Trees

MSCI 240: Algorithms & Data Structures

next class:

course evaluations

bring device (phone, laptop, etc.) to complete in class

lecture summary

binary tree terminology

`IntTreeNode` and `IntTree`

traversal & recursion with binary trees

Topic	Building Java Programs	Algorithms (Sedgewick)
classes, ADTs	chapter 8	1.2
arrays	chapter 7	
ArrayList<T>	chapter 10	1.3
Stack/Queue	chapter 14, (11)	1.3
LinkedList	chapter 16	1.3
Complexity	chapter 13	1.4
Searching		pp. 46-47
Sorting		chapter 2.1-2.3
Recursion	chapter 12	1.1 (p. 25)
Binary Trees	chapter 17	chapter 3.1-3.2
Dictionaries	chapter 18.1	chapter 3.4
Graphs	N/A (Wikipedia good)	chapter 4.1
Heaps/Priority Queues	chapter 18.2	chapter 2.4

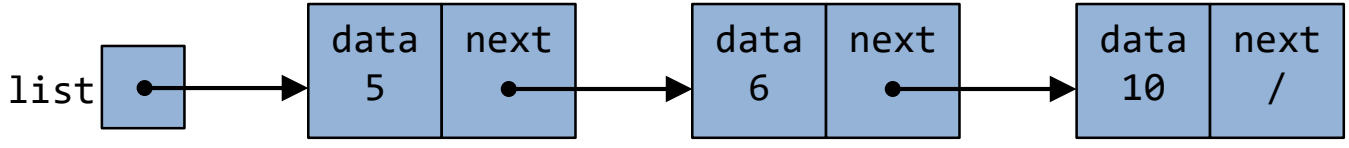
recall: `ListNode`

```
public class ListNode {
    int data;
    ListNode next;
}
```

each `ListNode` object stores:

- one piece of integer data
- a reference to another list node

`ListNode`s can be "linked" in chains to store a list of values:

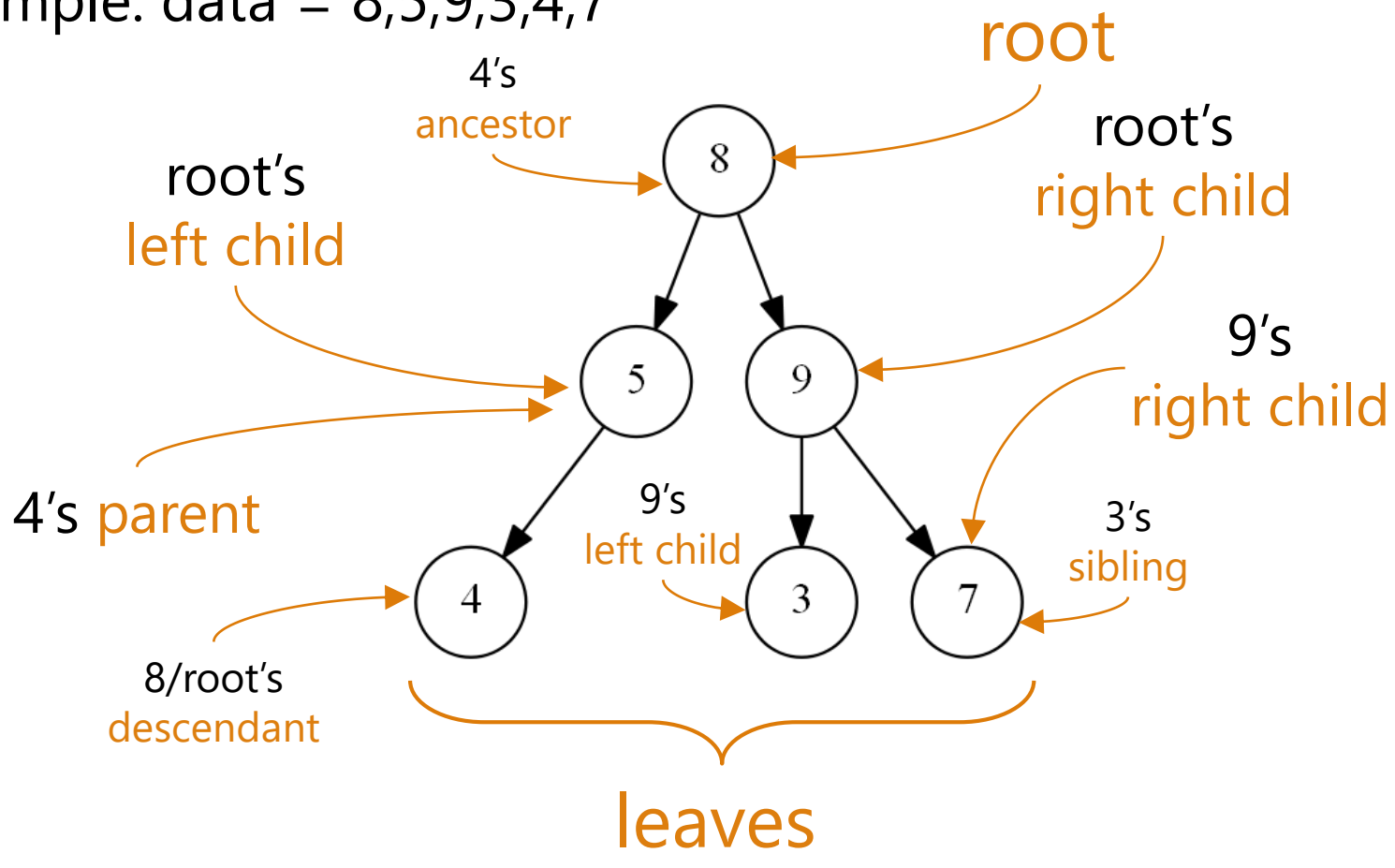


each `TreeNode` object stores:

- one piece of data
- a reference to up to two children: left & right

a binary tree is a collection of nodes that starts at a root node, and each node is linked to up to two children (which are also nodes)

example: data = 8,5,9,3,4,7



example: data = 8,5,9,3,4,7

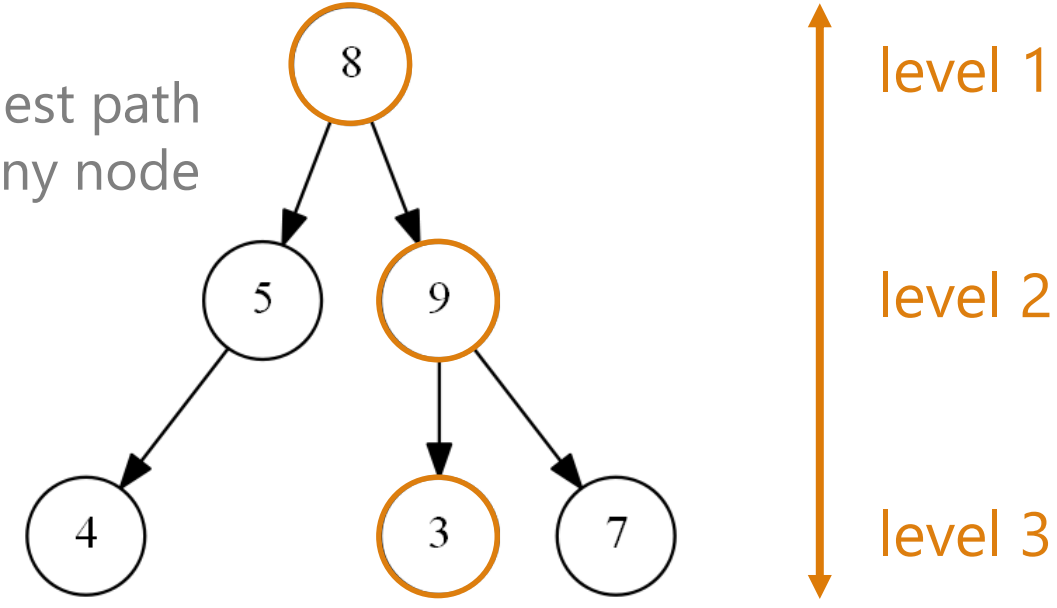
height = 3

height

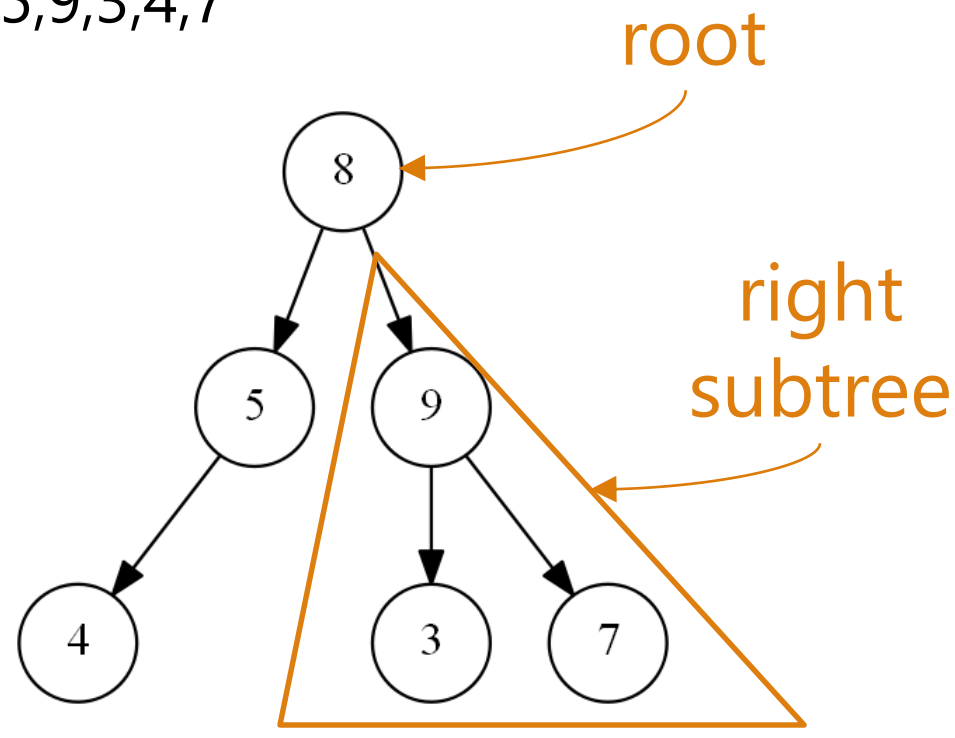
length of the longest path from the root to any node

level or **depth**

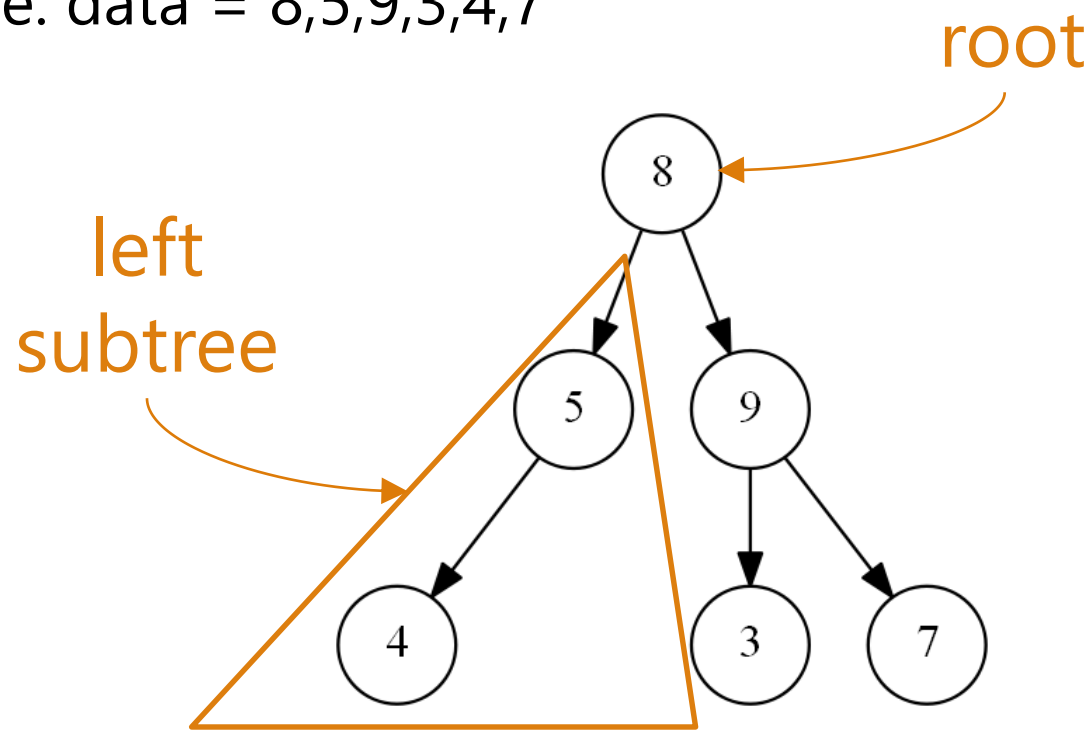
length of the path from a root to a given node



example: data = 8,5,9,3,4,7



example: data = 8,5,9,3,4,7



how many leaves are in this binary tree?

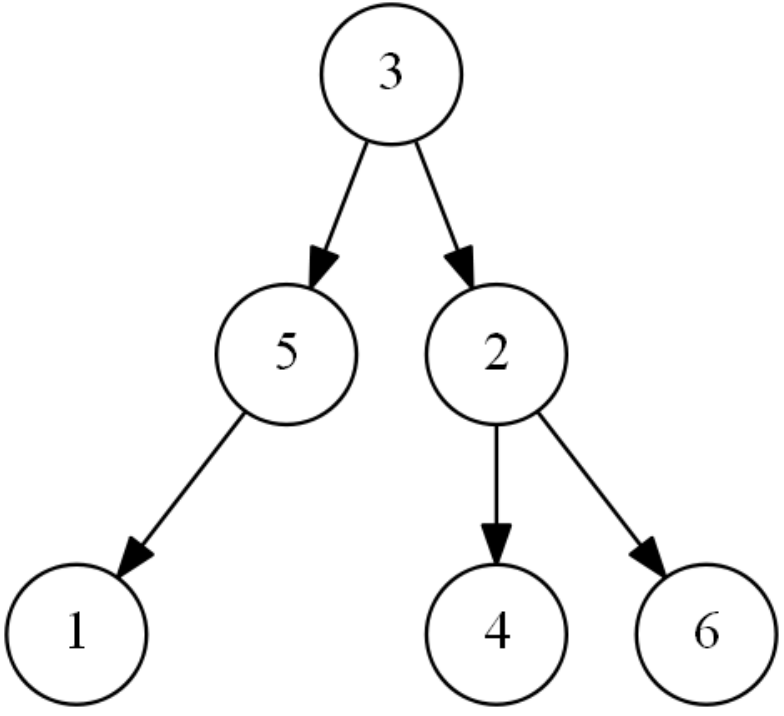
A. 2

B. 3

C. 4

D. 5

E. 6



programming with trees

trees are a mixture of linked lists and recursion

- considered very elegant (perhaps beautiful!)
- can be difficult for novices to master

common student remark #1:

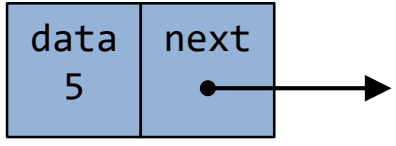
- “my code doesn’t work, and I don’t know why”

common student remark #2:

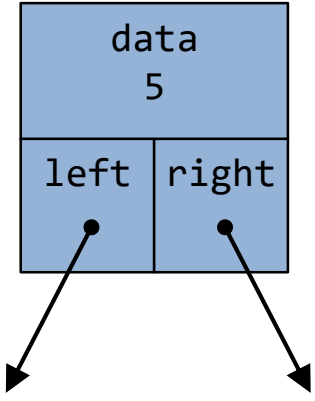
- “my code works, and I don’t know why”

recall: a node is a data structure that…

- holds a value
- holds a link to the next node
- (optional) holds a link to the previous node



node (binary tree)

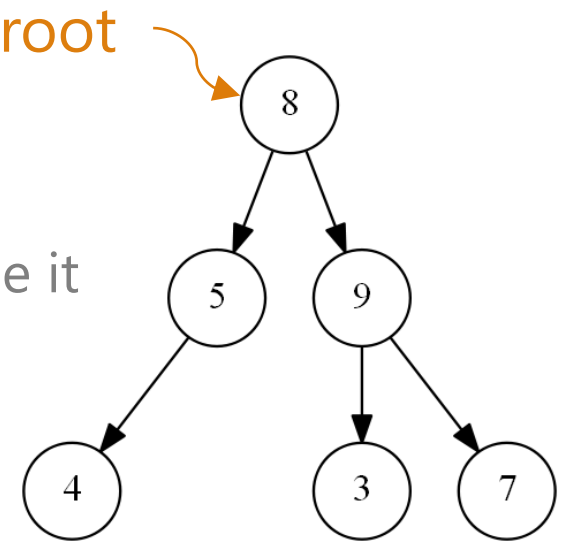


exercise: create `IntTreeNode` class

```java
// An IntTreeNode object is one node in a binary tree of ints.
public class IntTreeNode {
    public int data; // data stored at this node
    public IntTreeNode left; // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    // Constructs a leaf node with the given data.
    public IntTreeNode(int data) {
        this(data, null, null);
    }

    // Constructs a branch node with the given data and links.
    public IntTreeNode(int data, IntTreeNode left,IntTreeNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}
```

`IntTree` class

```
// an IntTree object represents an entire binary tree of ints
public class IntTree {
    private IntTreeNode root; // null for an empty tree

    // methods
}
```



client code

talks to `IntTree`, not to node objects inside it

methods of the `IntTree`

create/manipulate nodes, data and links

traversal

traversal: an examination of the elements of a tree

a pattern used in many tree algorithms and methods

common orderings for traversals:

- pre-order: process root node, then its left/right subtrees
- in-order: process left subtree, then root node, then right
- post-order: process left/right subtrees, then root node

to quickly generate a traversal:

trace a path around the tree

as you pass a node on the proper side, process it

pre-order (left side):

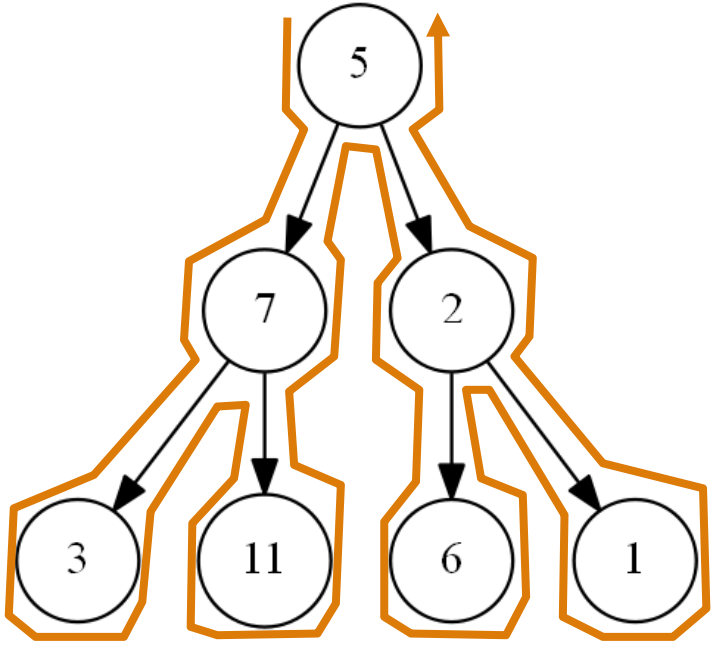
5 7 3 11 2 6 1

in-order (bottom):

3 7 11 5 6 2 1

post-order (right side):

3 11 7 6 1 2 5



the elements of the given tree would be in the following order using a pre-order traversal:

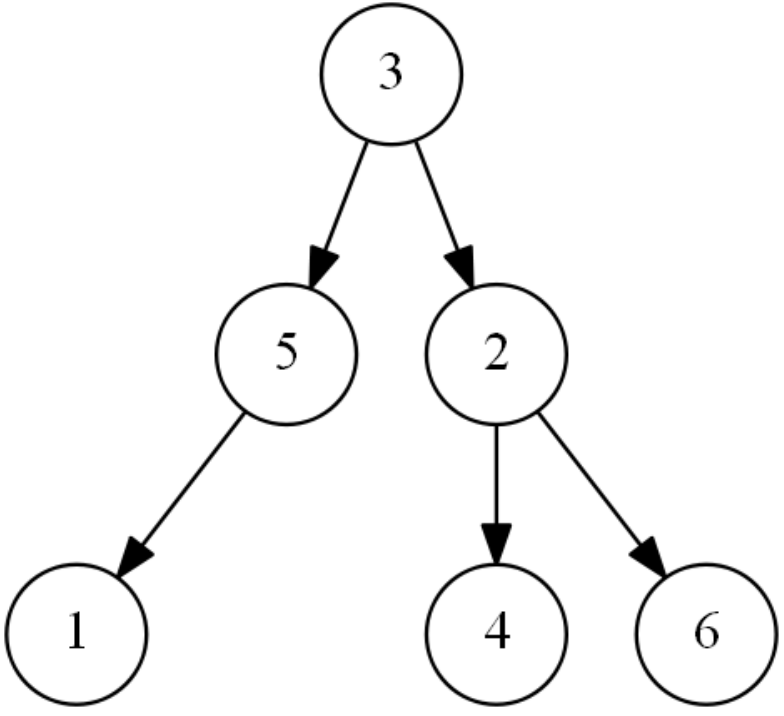
A. 3,5,2,1,4,6

B. 3,5,1,2,4,6

C. 1,5,3,4,2,6

D. 1,5,4,6,2,3

E. 1,2,3,4,5,6



the elements of the given tree would be in the following order using a in-order traversal:

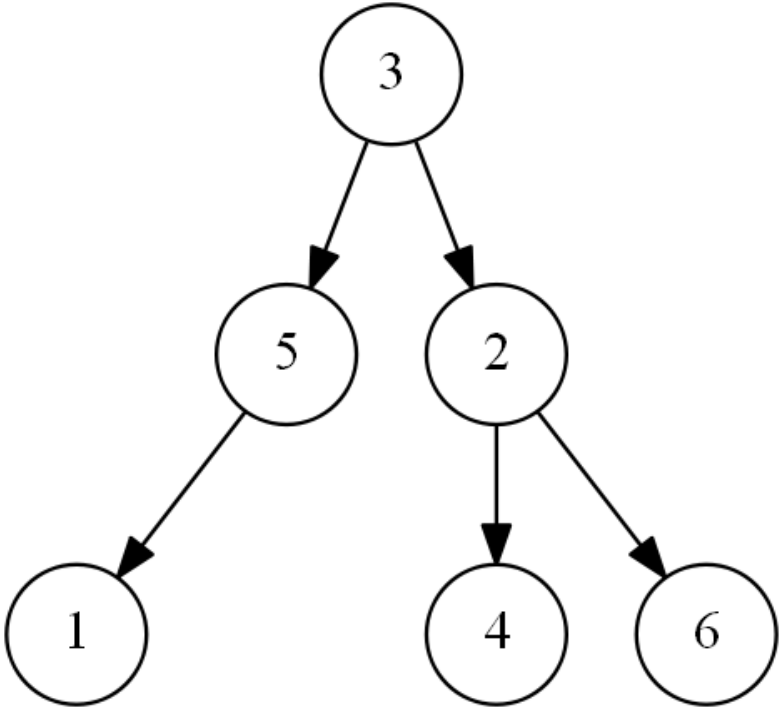
A. 3,5,2,1,4,6

B. 3,5,1,2,4,6

C. 1,5,3,4,2,6

D. 1,5,4,6,2,3

E. 1,2,3,4,5,6



the elements of the given tree would be in the following order using a post-order traversal:

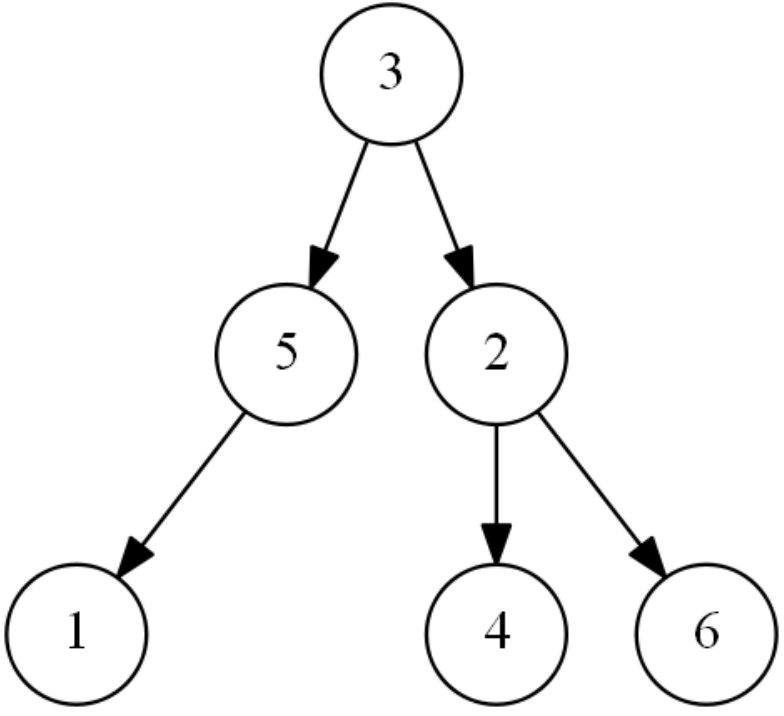
A. 3,5,2,1,4,6

B. 3,5,1,2,4,6

C. 1,5,3,4,2,6

D. 1,5,4,6,2,3

E. 1,2,3,4,5,6



recursion on trees

a binary tree is either:

an empty tree

a root node that refers to two other trees,
known as the left subtree and the right subtree

template for tree methods

```
// an IntTree object represents an entire binary tree of ints
public class IntTree {
    private IntTreeNode root; // null for an empty tree

    // ...
    public type name(parameters) {
        name(root, parameters);
    }

    private type name(IntTreeNode node, parameters) {
        //...
    }
}
```

tree methods are often implemented recursively

- with a **public**/**private** pair
- the **private** version accepts the root node to process

typical approach to traversal

```
public void doSomething() {
    doSomething(root);
}

private void doSomething(IntTreeNode node) {
    // base case(s)?
    // recursive case(s)?









}
```

typical approach to traversal

```
public void doSomething() {
    doSomething(root);
}

private void doSomething(IntTreeNode node) {
    // base case: do nothing (node is null)
    // recursive case
    if (node != null) {
        // do something (pre-order)
        doSomething(node.left);
        // do something (in-order)
        doSomething(node.right);
        // do something (post-order)
    }
}
```

exercise

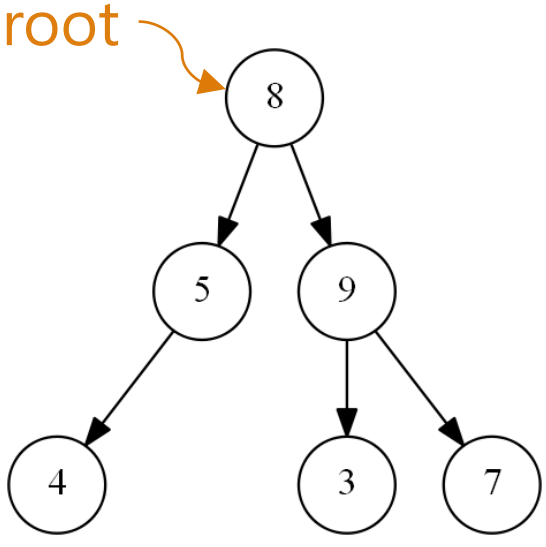
add a method `print` to the `IntTree` class that prints the elements of the tree, separated by spaces using a pre-order traversal

example:

```
tree.print();
```

output:

```
8  5  4  9  3  7
```



```
// an IntTree object represents an entire binary tree of ints
public class IntTree {
    private IntTreeNode root; // null for an empty tree

    // ...
    public void print() {
        print(root);
        System.out.println(); // end the line of output
    }

    private void print(IntTreeNode node) {
        // (base case is implicitly to do nothing for empty trees)
        if (node != null) {
            // recursive case: print this node, left tree, right tree
            System.out.print(node.data + " ");
            print(node.left);
            print(node.right);
        }
    }
}
```

```
// an IntTree object represents an entire binary tree of ints
public class IntTree {
    private IntTreeNode root; // null for an empty tree

    // ...
    public void print() {
        print(root);
        System.out.println(); // end the line of output
    }

    private void print(IntTreeNode node) {
        // (base case is implicitly to do nothing for empty trees)
        if (node != null) {
            // recursive case: print this node, left tree, right tree
            System.out.print(node.data + " "); // pre-order
            print(node.left);
            print(node.right);
        }
    }
}
```

```
// an IntTree object represents an entire binary tree of ints
public class IntTree {
    private IntTreeNode root; // null for an empty tree

    // ...
    public void print() {
        print(root);
        System.out.println(); // end the line of output
    }

    private void print(IntTreeNode node) {
        // (base case is implicitly to do nothing for empty trees)
        if (node != null) {
            // recursive case: print left tree, this node, right tree
            print(node.left);
            System.out.print(node.data + " "); // in-order
            print(node.right);
        }
    }
}
```

```
// an IntTree object represents an entire binary tree of ints
public class IntTree {
    private IntTreeNode root; // null for an empty tree

    // ...
    public void print() {
        print(root);
        System.out.println(); // end the line of output
    }

    private void print(IntTreeNode node) {
        // (base case is implicitly to do nothing for empty trees)
        if (node != null) {
            // recursive case: print left tree, right tree, this node
            print(node.left);
            print(node.right);
            System.out.print(node.data + " "); // post-order
        }
    }
}
```

is there any way these traversals could help us with searching/sorting?

binary tree summary

binary tree terminology

- node, root, leaf, parent, child, sibling, subtree, etc.

`IntTree` encapsulates creation/modification of `IntTreeNode`, their links, etc.

traversal

- traversal is typically done recursively
- pre-order, in-order, post-order traversal

next:

binary search trees