Recursion

The Big Recursive Idea

When to Use Recursion

Recursively Defined Data Structures

Who's Our Best Customer?

The manager of DelegateCorp needs to determine which of six customers produces the most revenue for his company. Two factors complicate this otherwise simple task:

1. Determining the total revenue for a customer requires going through that customer's whole file and tallying numbers on dozens of orders and receipts.

2. The employees of DelegateCorp love to delegate, and each employee passes work along to someone at a lower level whenever possible. To keep the situation from getting out of hand, the manager enforces a rule: When you delegate, you must do some portion of the work yourself, and you have to give the delegated employee less work than you were given.

Assume the following employees can help, from highest rank to lowest: Manager, Vice Manager, Associate Manager, Junior Manager, Intern.

The Big Recursive Idea

The Big Recursive Idea

When you pass along a smaller version of a problem for someone else to solve, you don't care *how* it's done, just that it *is*.

The Big Recursive Idea

When you pass along a smaller version of a problem for someone else to solve, you don't care *how* it's done, just that it *is*.

…so pretend there is no recursion at all!

How can we make this recursive?

```
int iterativeArraySum(int[] integers, int size)
{
    int sum = 0;
    for (int i = 0; i < size; i++)
    {
        sum += integers[i];
    }
    return sum;
}
```

The trick: we hand off work to the next procedure. We care *what* answer we get back, but now *how* it's computed.

First: Code that is halfway between iterative and recursive…

Add a dispatcher that hands off most work to an already written iterative function.

The dispatcher must:

1. completely handle the trivial case
2. pass a smaller version of the problem to the iterative function

First: Code that is halfway between iterative and recursive…

```
int arraySumDelegate(int[] integers, int size)
{
   if (size == 0) return 0;
   int lastNumber = integers[size - 1];
   int allButLastSum =
      iterativeArraySum(integers, size - 1);
   return lastNumber + allButLastSum;
}
```

Second: Change the dispatcher to call itself, remove the iterative function

```
int arraySumRecursive (int[] integers, int size)
{
   if (size == 0) return 0;
   int lastNumber = integers[size - 1];
   int allButLastSum =
      arraySumRecursive(integers, size - 1);
   return lastNumber + allButLastSum;
}
```

Don’t literally do this – just think about what the dispatcher would do if there was an iterative solution already written.

If you correctly created a dispatcher, you already have a recursive solution.

That’s the Big Recursive Idea!

When to Use Recursion

Arguments Against Recursion

Conceptual Complexity

It's often easier to write code with loops.

Performance

Lots of function calls incur overhead.

Space Requirements

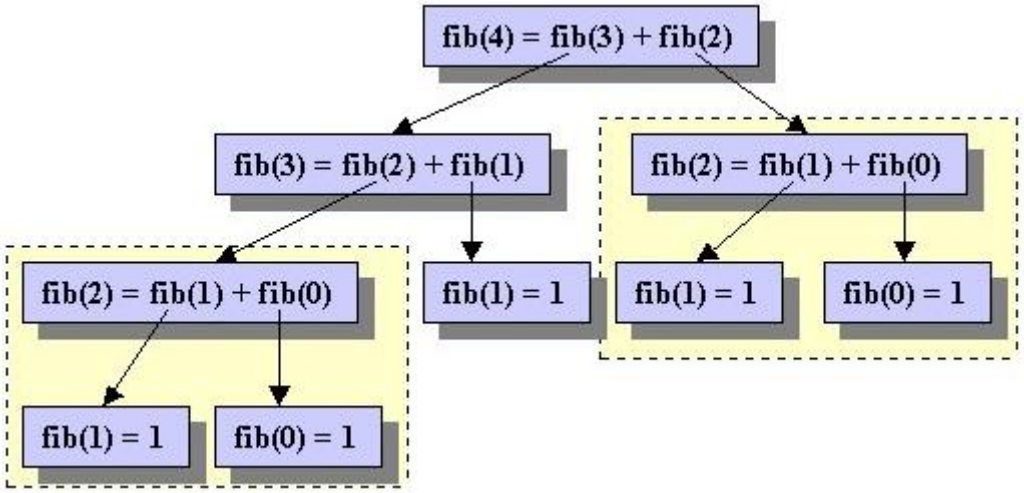
Function calls are nested.

```
public static int fibonacci(int n)
{
        if (n <= 1)
                return 1;

        return fibonacci(n-1) + fibonacci(n-2);
}
```

```
public static int fibonacci(int n)
{
        if (n <= 1)
                return 1;

        return fibonacci(n-1) + fibonacci(n-2);
}
```



```
public static int fibonacci2(int  n)
{
	int first = 1;
	int second = 1;
	int third = 1;

	for (int i=2; i<=n; i++)
	{
		third = first + second;
		first = second;
		second = third;
	}

	return third;
}
```

```
public static int fibonacci2(int  n)
{
        int first = 1;
        int second = 1;
        int third = 1;

        for (int i=2; i<=n; i++)
        {
                third = first + second;
                first = second;

        }

        return third;
}
```

Compute new value

```
public static int fibonacci2(int  n)
{
        int first = 1;
        int second = 1;
        int third = 1;

        for (int i=2; i<=n; i++)
        {
                third = first + second;
                first = second;
                second = third;
        }

        return third;
}
```

Shift others to the right

When to Use Recursion

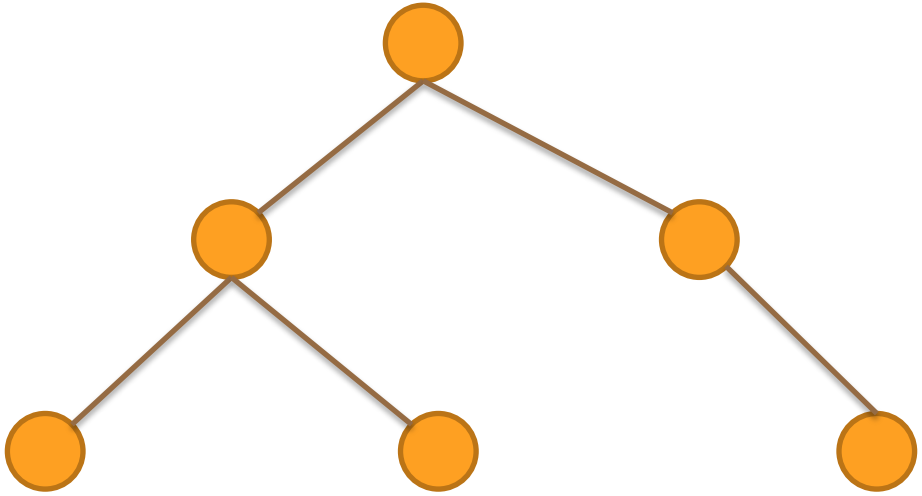
Use recursion when the arguments against it don’t apply.

An **indirectly recursive method** is one that does not call itself, but it does call a recursive method.

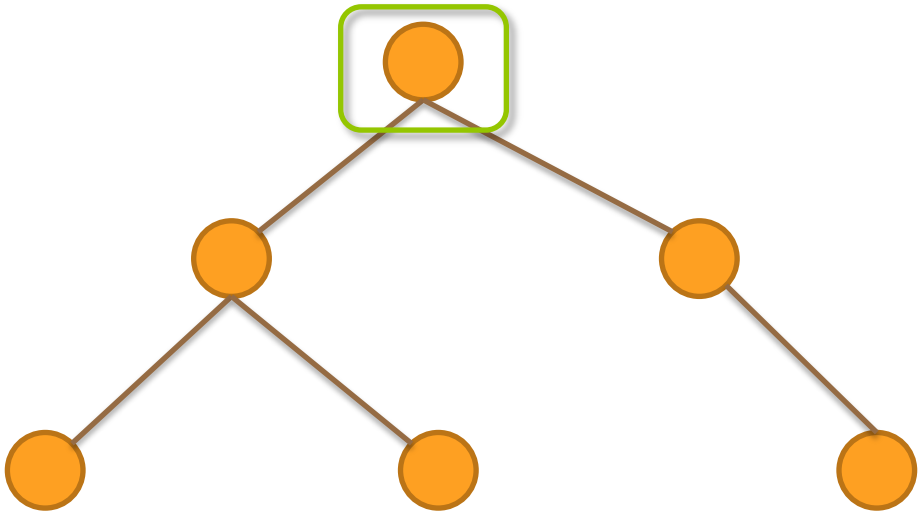
Recursively Defined Data Structures

Linked lists are recursively defined.
Why?

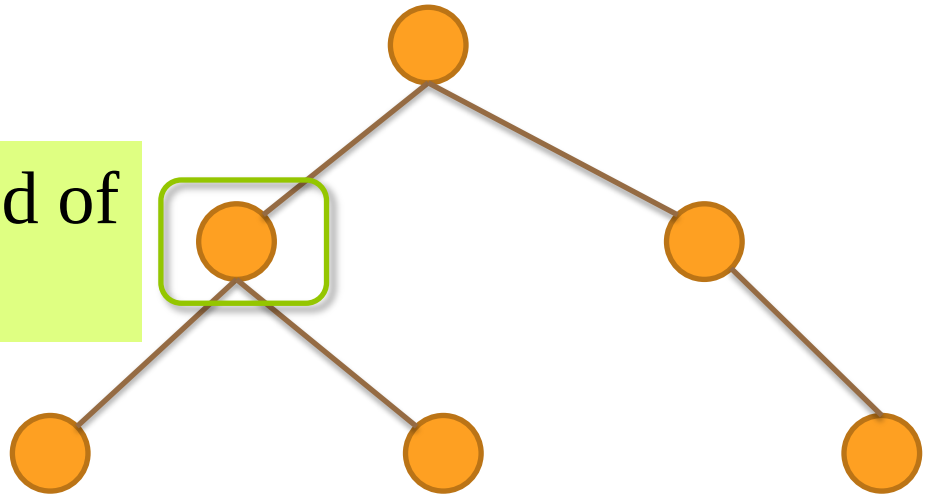
Are binary trees recursively defined?



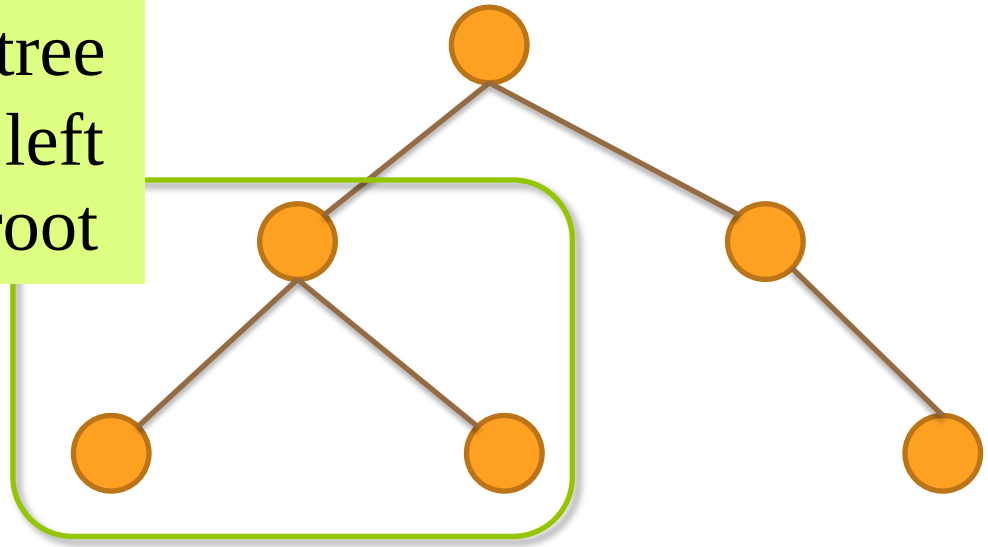
Root of the tree

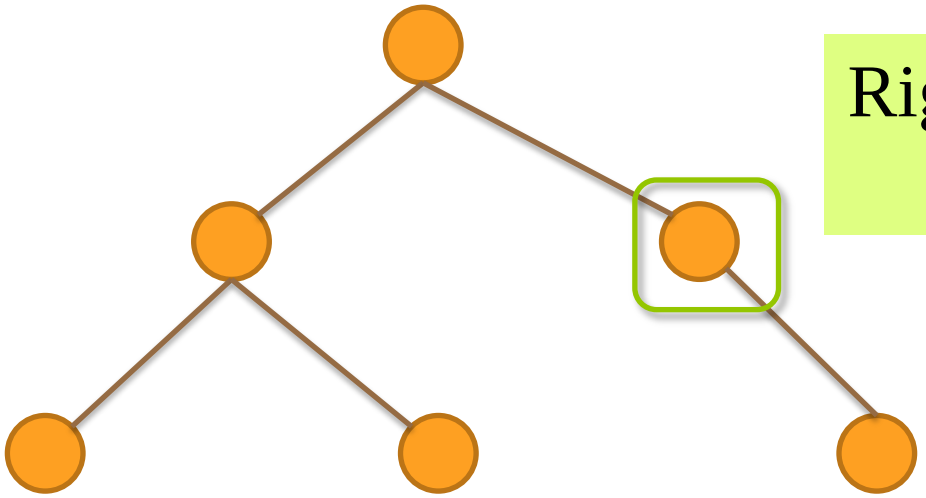


Left child of root

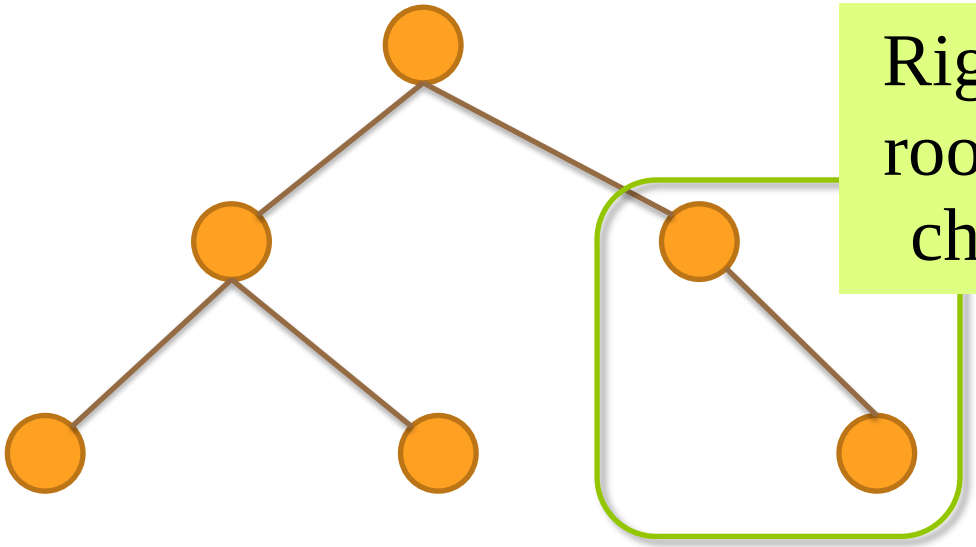


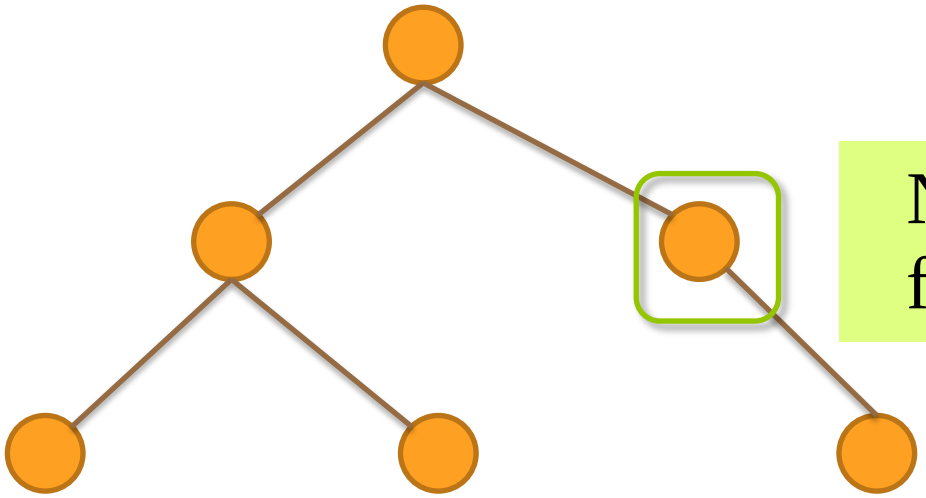
Left sub-tree rooted at left child of root



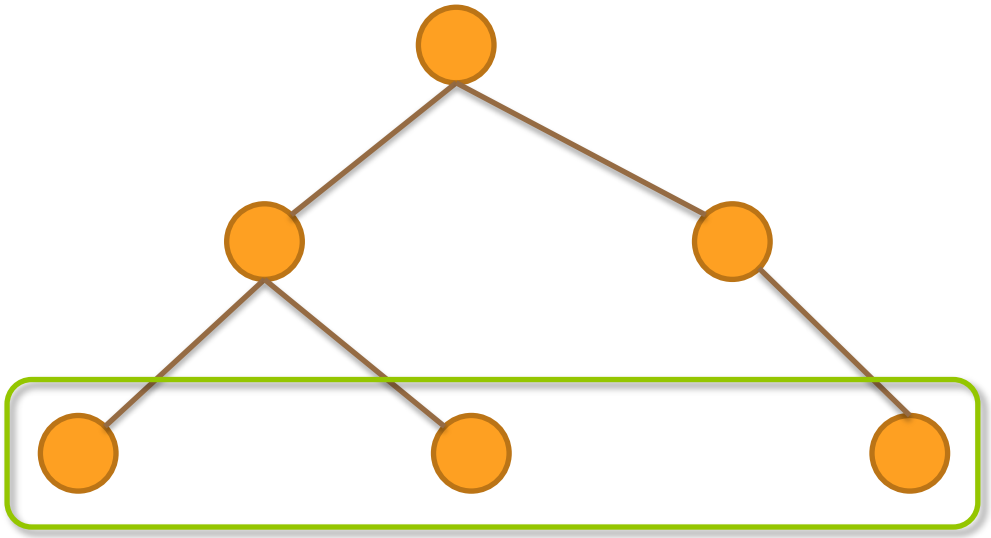


Right child of root

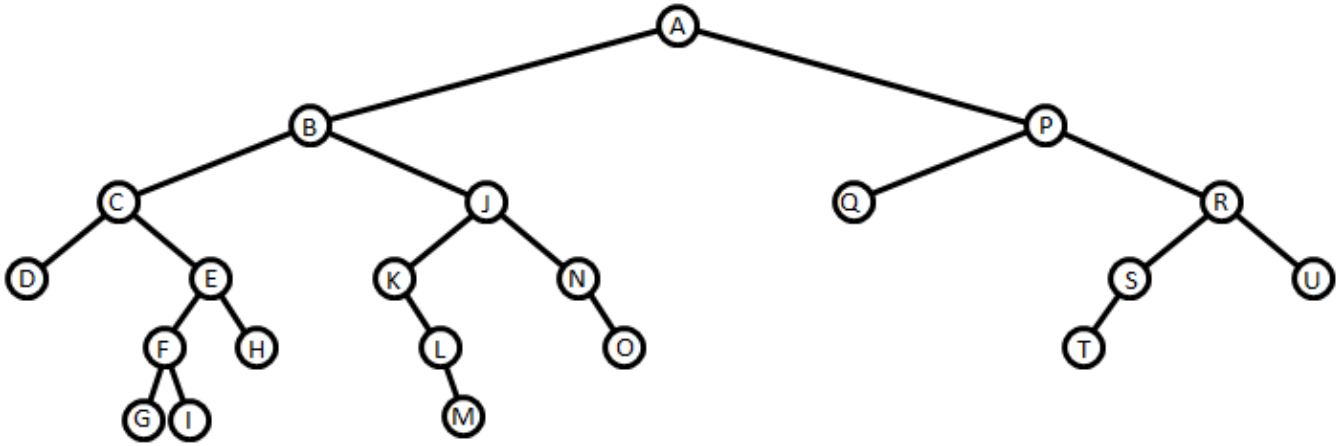

Right sub-tree
rooted at right
child of root



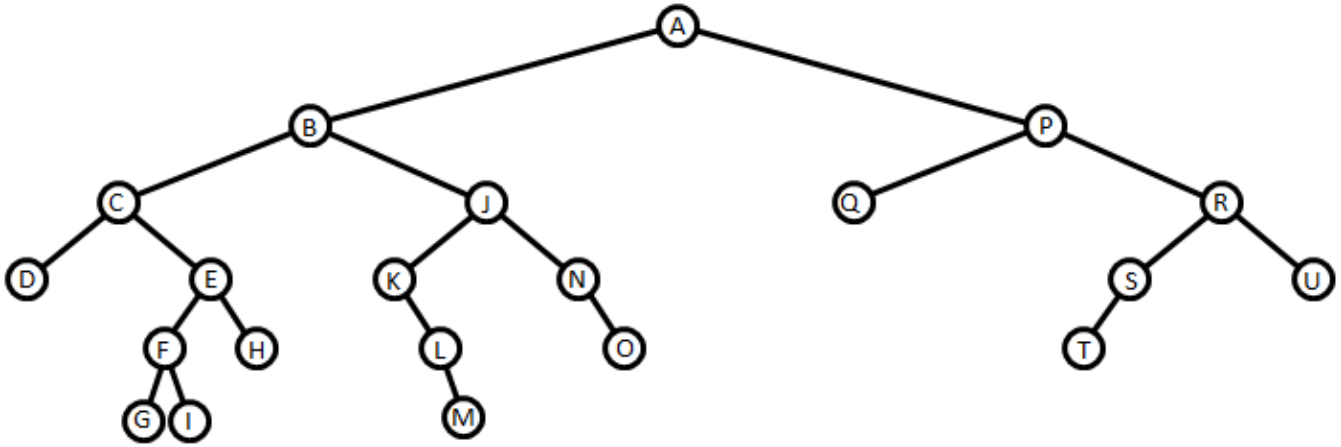
No left child
for this node



Leaves: no children at all

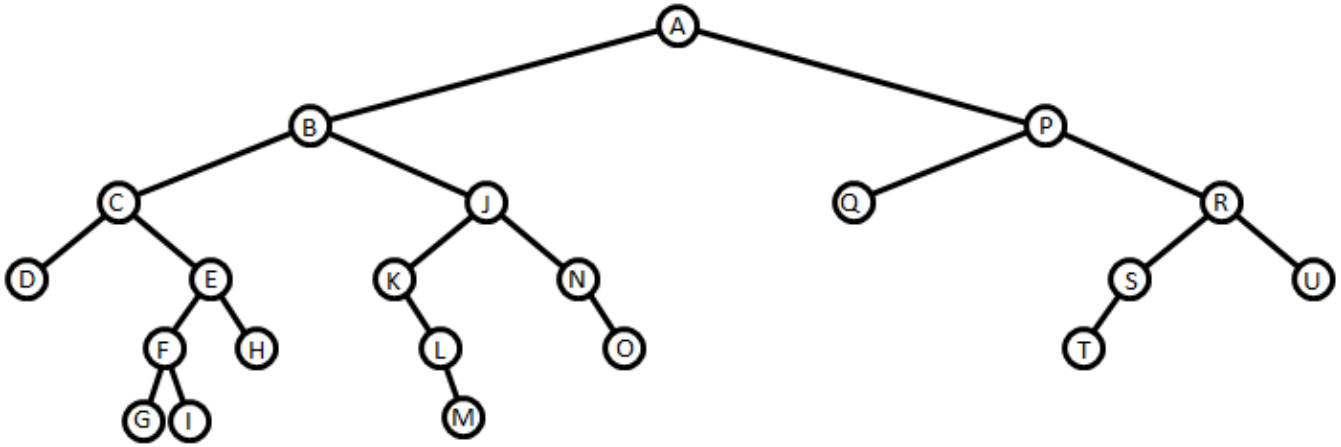

A
B
P
C
J
Q
R
D
E
K
N
S
U
F
H
L
O
T
G
I
M

How can we find the height of a tree?



height = 6

How can we collect all the leaves to return?



leaves: D, G, I, H, M, O, Q, T, U