CS112: Data Structures

Lecture 05

Structures for search/add/delete

Review: Recursion

- **Recursion is a way of looking at a problem**
- **EG problem: print pattern like**

  *
  * *
  * * *
  * * * *

Recursion

- **Non-recursive view**

```
*
* *
* * *
* * * *
```

- **A size 4 triangle is four lines, lengths 1, 2, 3, 4**

Recursion

- **Recursive view**

```
*
* *
* * *
* * * *
```

- **A size 4 triangle is**
  - **A size 3 triangle, followed by**
  - **A line of length 4**

Recursive Definitions in Math

- **Factorial**

  $n! = n * (n-1)!$

  $1! = 1$

  e.g., $3! = 3*2! = 3 * (2 * 1!) = 3 * (2 * 1) = 6$
- **Definition looks circular, but is not**
- **Two parts:**
  - **recursive case**
  - **base case**

Recursive Methods

- **Example: palindrome**
  - **same letters backwards as forwards (assume no space or punctuation)**
  - **e.g, radar**
    madam im adam

    a man a plan a canal panama
- **How can we write a method to test if a string is a palindrome?**

Recursive Definition

- A string is a palindrome if
  - first and last characters are the same, and

    radar
  - rest of string without first and last is a palindrome

    ada
- A string of length 0 or 1 is a palindrome

  d
- See RecString.java

Integer Power

How many multiplies does it take to calculate 3^8 ?

$3 * 3 = 9$	3^2
$9 * 3 = 27$	3^3
$27 * 3 = 81$	3^4
$81 * 3 = 243$	3^5
$243 * 3 = 729$	3^6
$729 * 3 = 2187$	3^7
$2187 * 3 = 6561$	3^8

7 *'s

$3 * 3 = 9$	3^2
$9 * 9 = 81$	3^4
$81 * 81 = 6561$	3^8

3 *'s

$$3^{y/2} * 3^{y/2} = 3^{(y/2+y/2)}$$
$$= 3^y$$

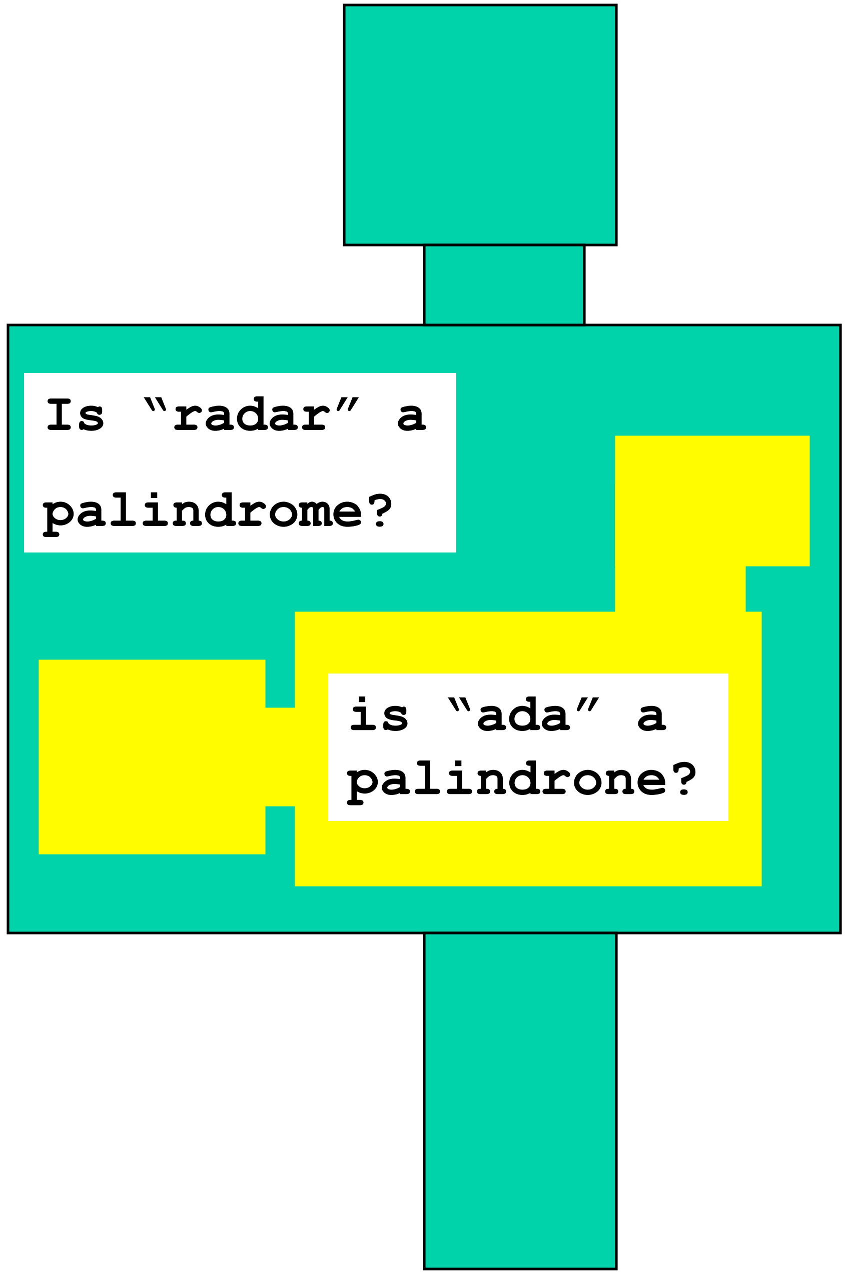
Recursive Definition

- **y even: $x^y = x^{y/2} * x^{y/2} = (x^{y/2})^2$**
- **y odd: $x^y = x * x^{\lfloor y/2 \rfloor} * x^{\lfloor y/2 \rfloor}$**

  $= x * (x^{\lfloor y/2 \rfloor})^2$
- **y = 1: $x^y = x$**
- **y = 0: $x^y = 1$**

- **See Power.java**

- **Seeing recursion**
  - **look at a problem as if it were "pregnant":**
    - **inside it is a small version of the same problem**



Designing Recursive Methods

- **Print triangle of *s**

```
*
**
***
****
```

- **Print a triangle size 4,**
  - **Can you see how solving a similar but smaller problem would help solve this one?**

Designing Recursive Methods

- **Print triangle of *s**

```
*
**
***
****
```

- **To print triangle size 4,**
  - **print a triangle of size 3**
  - **print 4 stars**

Ruler Pattern

```
*
**
*
***
*
**
*
****
*
**
*
***
*
**
*
```

Ruler Pattern

- Suppose you have a method that prints

```
*
**
*
```

How could you use it to print?

```
*
**
*
***
*
**
*
```

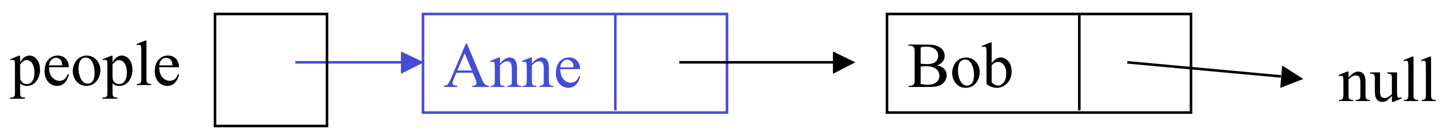
Ruler Pattern

```
*
**
*
***
*
**
*
```

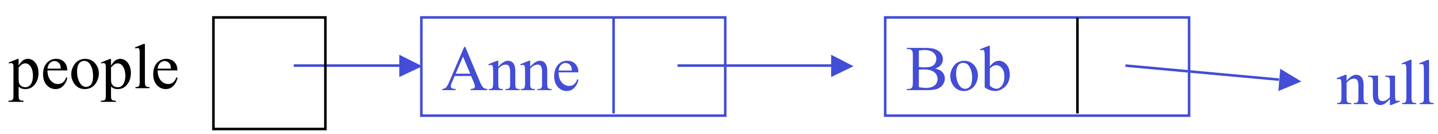
- Smaller problem appears twice!
- To do ruler 3:
  - do ruler 2
  - print 3 *s
  - do ruler 2
- See RecString.java

“Recursive” Data Types

- **We can look at a reference to a node in two ways**
  - **It refers to a specific node**

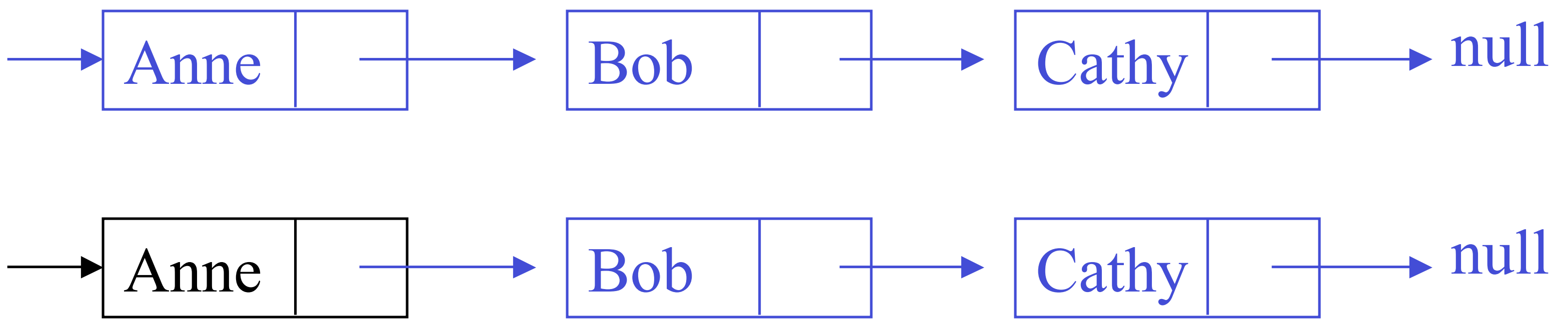


  - **It refers to the entire list that the node starts**



"Recursive" Data Types

- **If a reference to a node means the whole list, then the next field of that node is "the rest" of the list.**

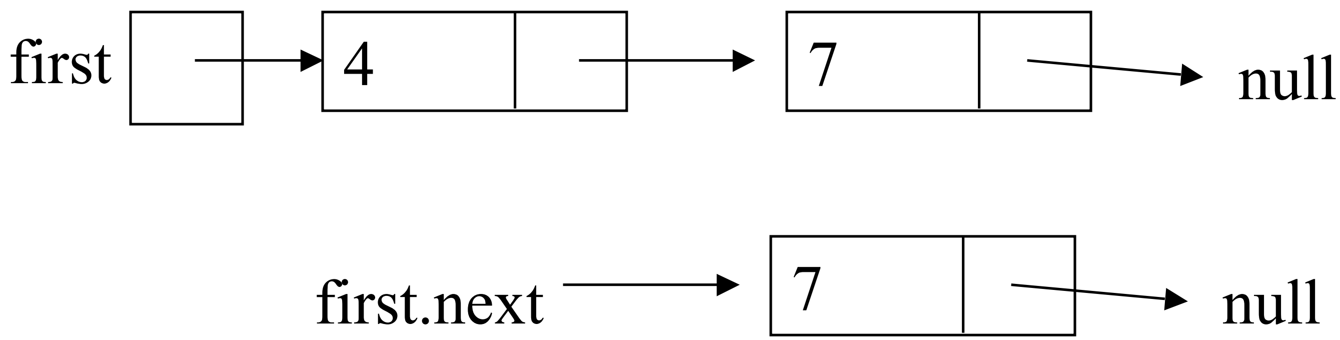


- **"Next is the first node of the rest of your list."**

“Recursive” Data Types

- **See RecNode2.java**

NodeToString

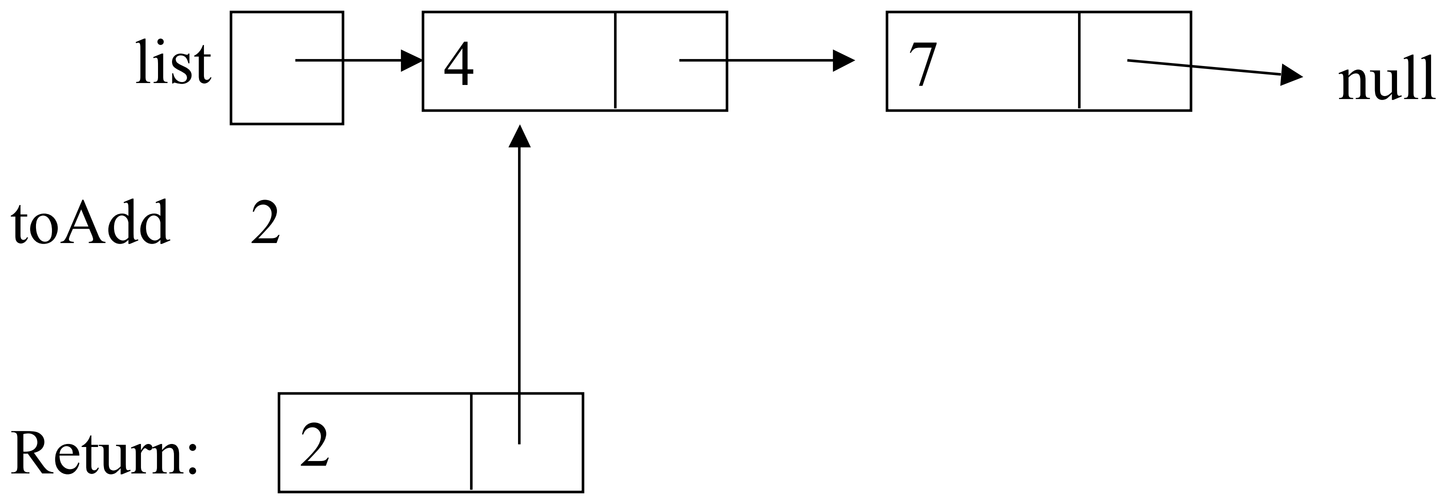


nodeToString(first.next) is “7 -> [end]”

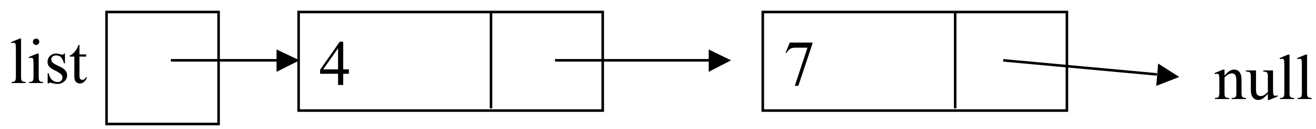
first.data is 4

nodeToString(first) returns “4 -> 7 ->[end]”

InsertInOrder

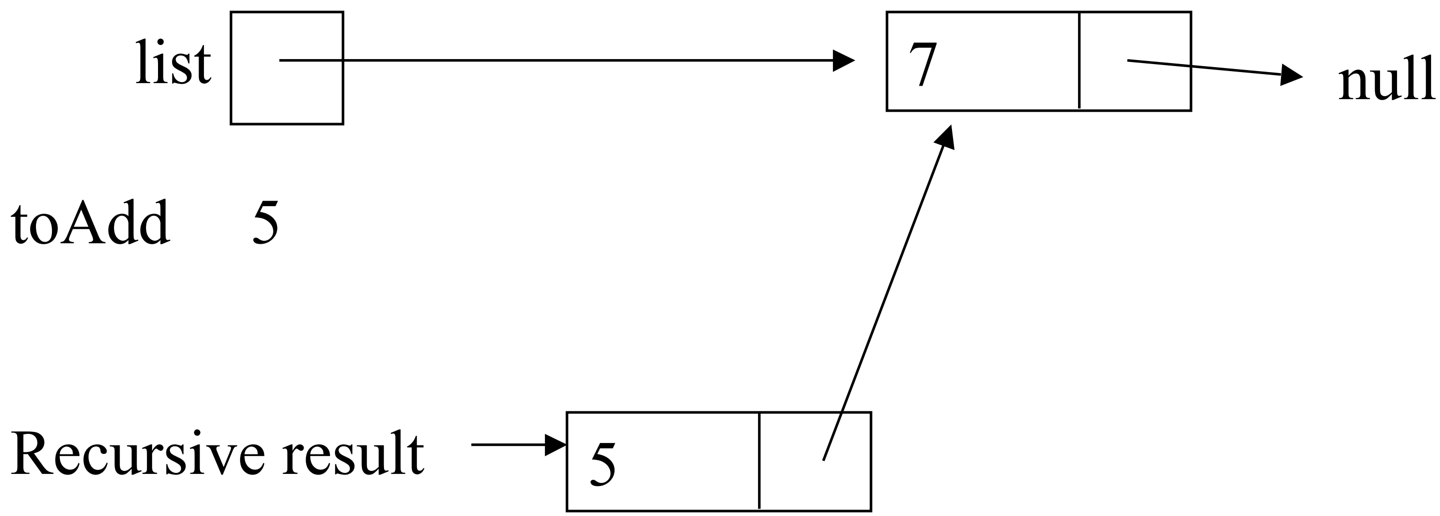


InsertInOrder

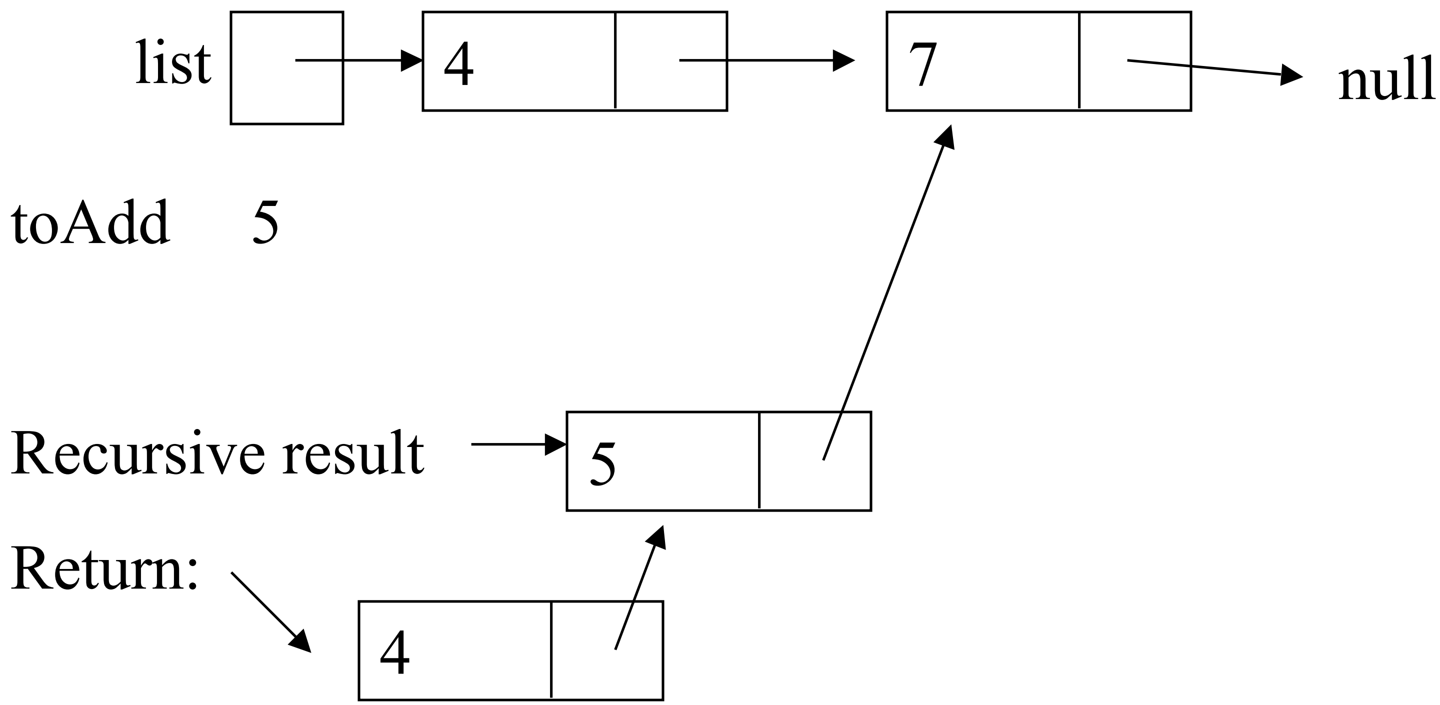


toAdd 5

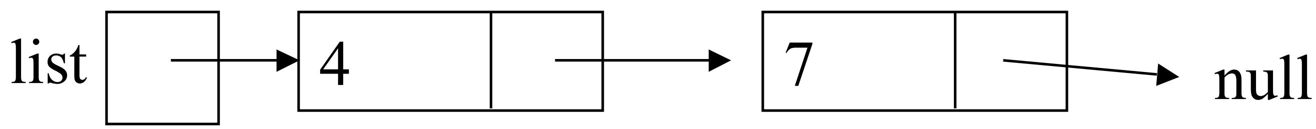
Recursive call



InsertInOrder

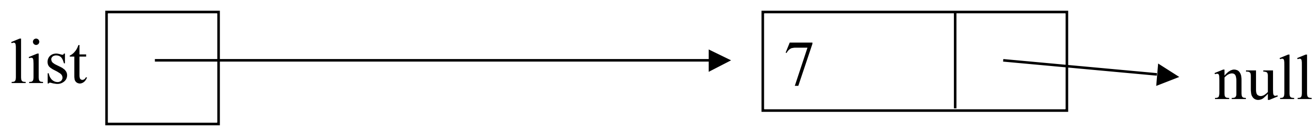


InsertInOrder



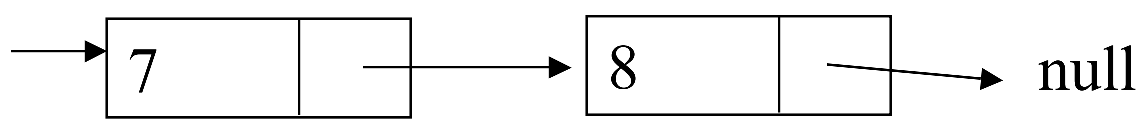
toAdd 8

Recursive call

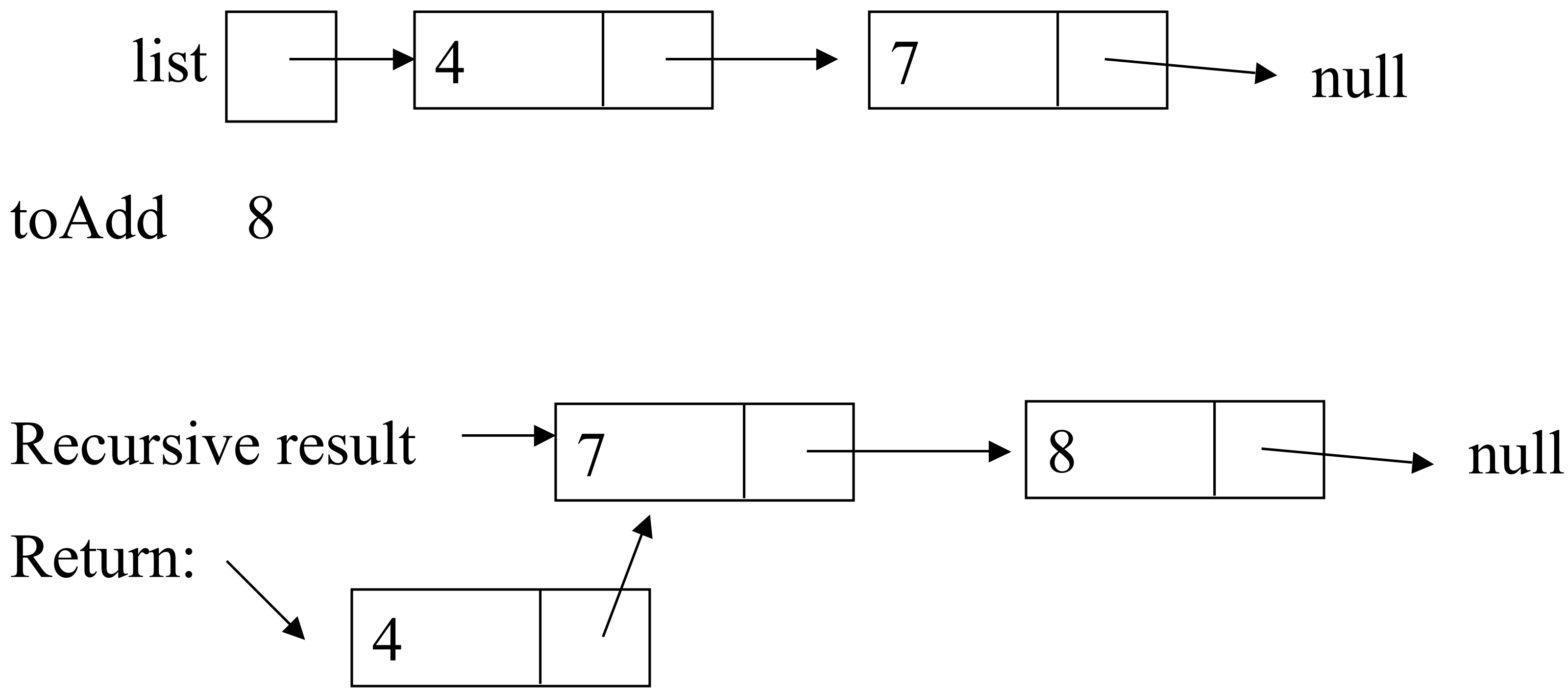


toAdd 8

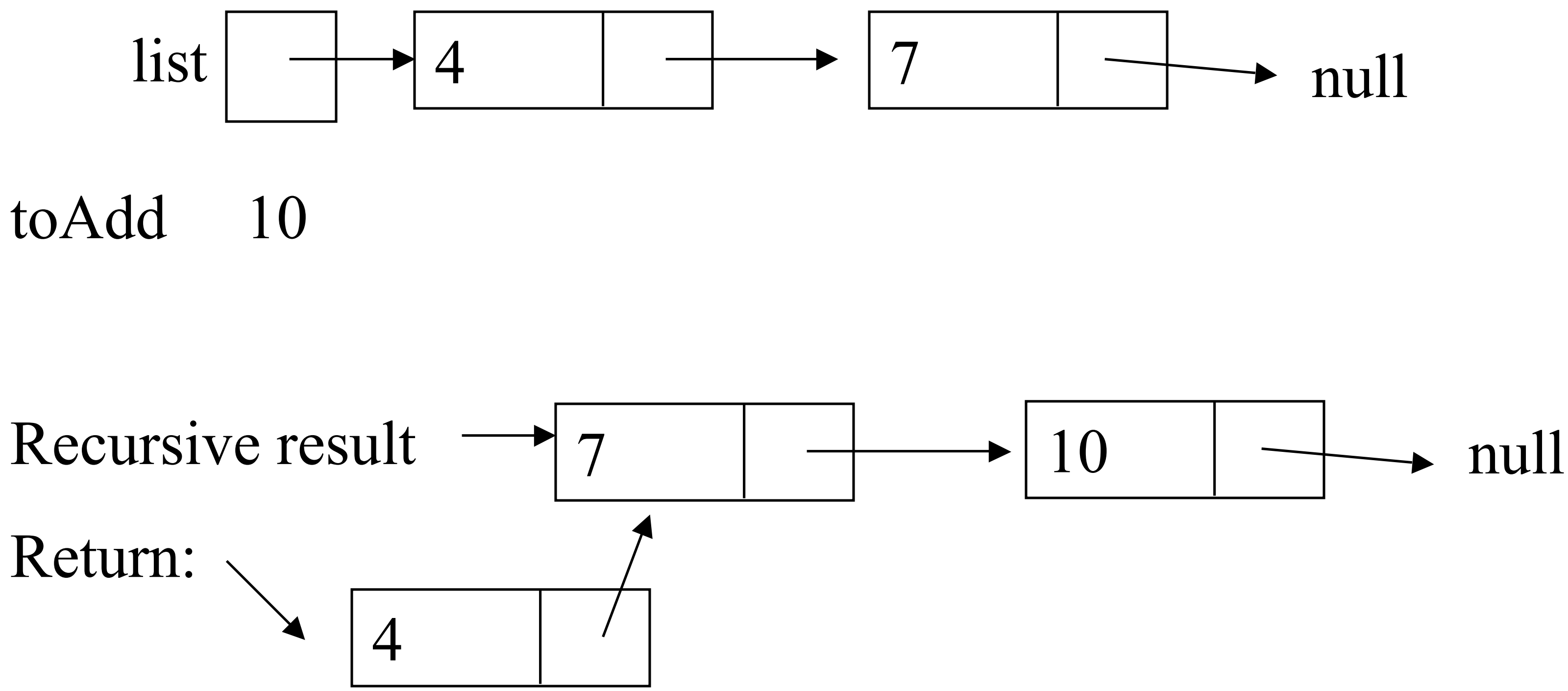
Recursive result



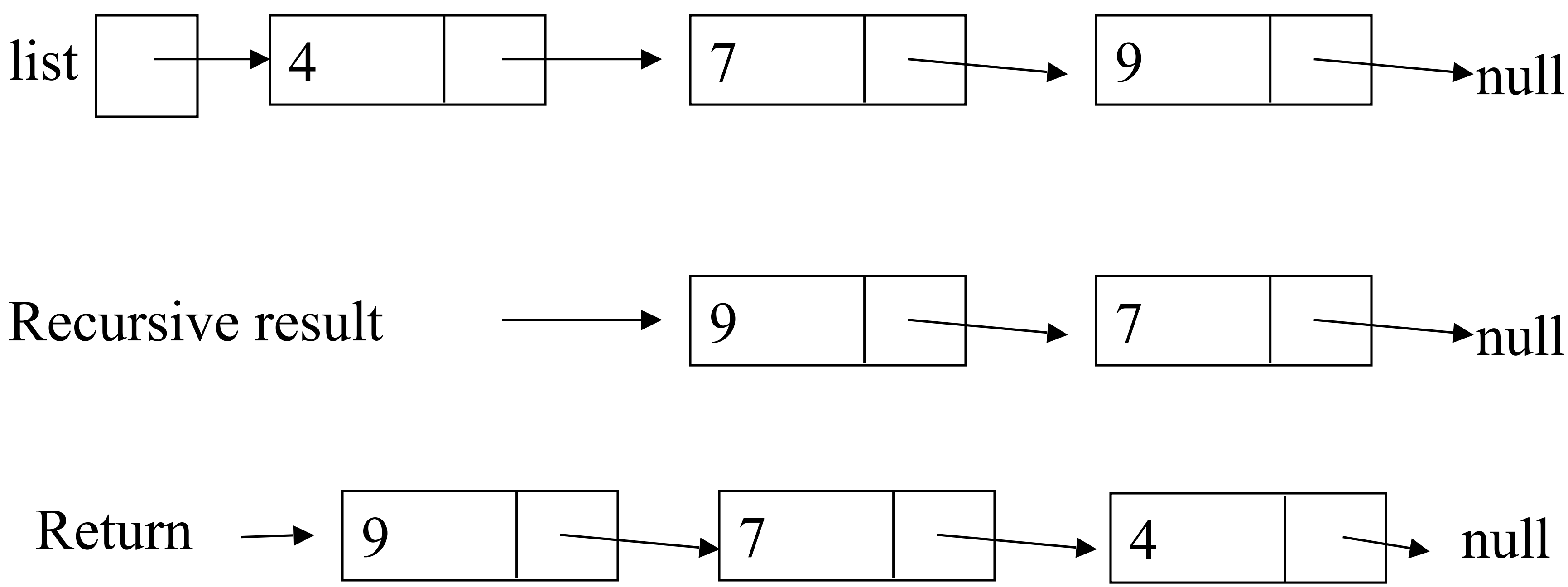
InsertInOrder



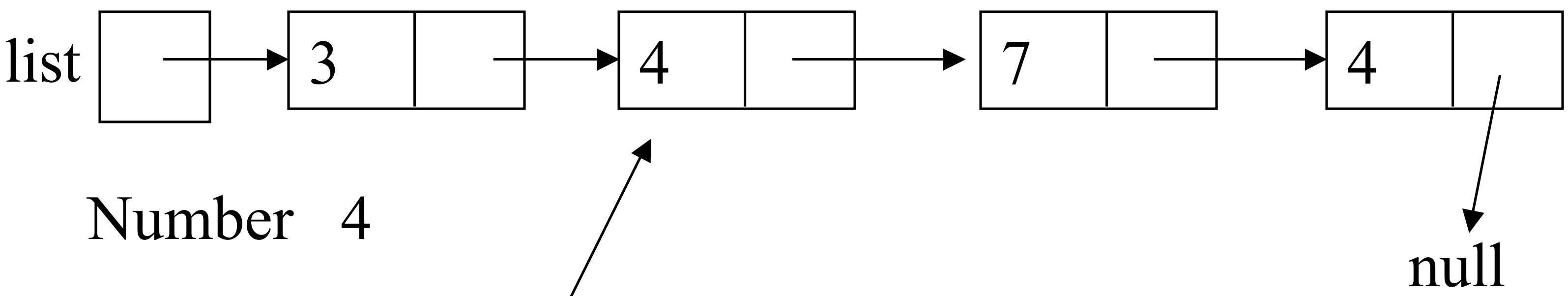
AddAtTail



Reverse



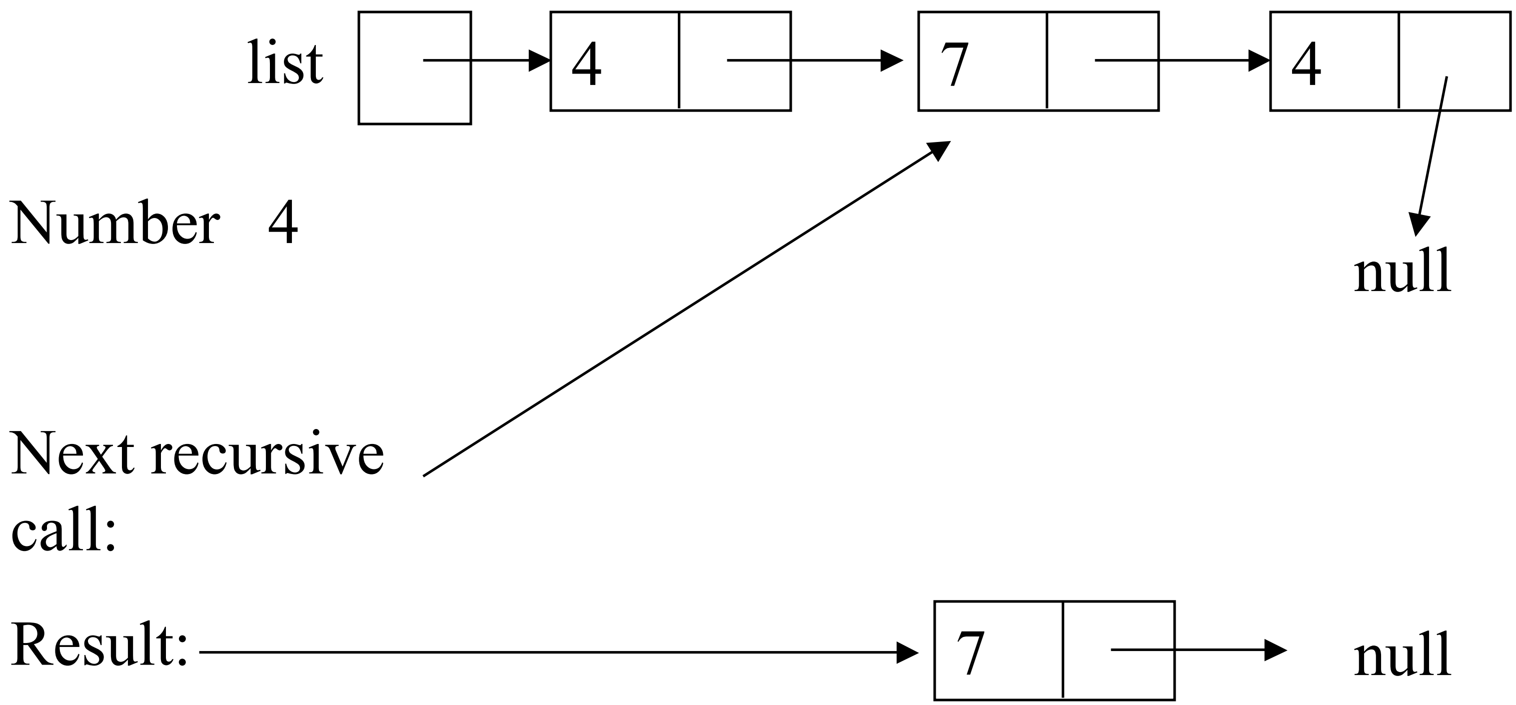
WithoutAll



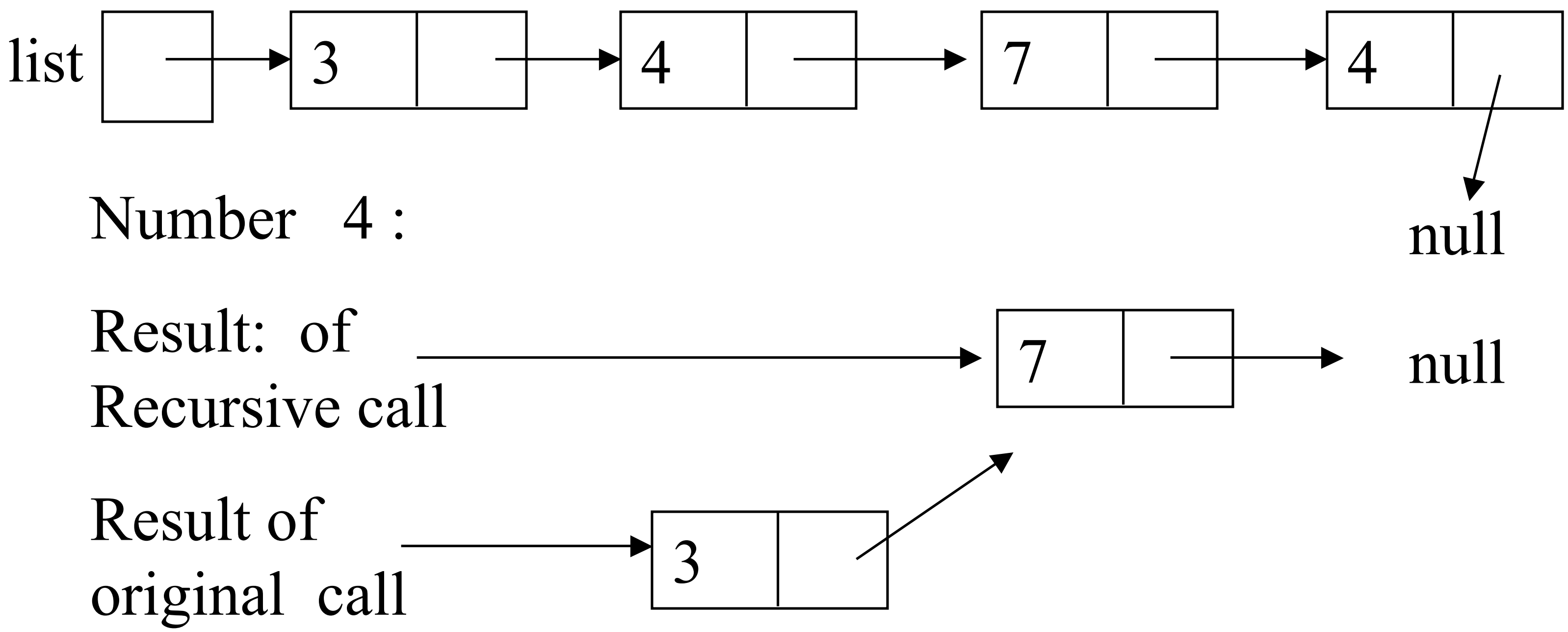
Number 4

Recursive call:

Recursive call



Original call



Debugging List Code

- **See BadList.java**

New: Add / Delete / Search

- **Basic task:**
  - **Set of data items**
    - **E.g. “Al”, “Bob”, “Cindy”**
  - **Operations:**
    - **Add an item**
    - **Delete an item**
    - **Search for an item**
- **Goal: minimize**

  worst case O(add + delete + search)

Unordered Array

- Insert O(1) if there's space
- Delete O(1) (move last element)
- Search O(n) where n is size of set
- Overall O(n)

Ordered Array

- **Insert O(n)**
- **Delete O(n)**
- **Search ??**
- **Overall ??**

Searching an ordered array
Binary search

- requires sorted values
- each comparison rules out half of the remaining elements
- O(log(n)) - we will prove this later
- Find A, find R

A	D	E	F	G	H	J	M	P	T
0	1	2	3	4	5	6	7	8	9

Searching an array Performance

- Search among 1 billion entries
- Check 1 million entries per second
  - Sequential search
    - 1 billion operations needed
    - Requires 1000 seconds - about 20 minutes
  - Binary search
    - 30 operations needed
    - Requires 30 microseconds
    - 30 million times faster

Ordered Array

- Insert O(n)
- Delete O(n)
- Search O(log n)
- Overall O(n)

Unordered Linked List

- Insert O(1)
- Delete O(1)
- Search O(n)
- Overall O(n)

Ordered Linked List

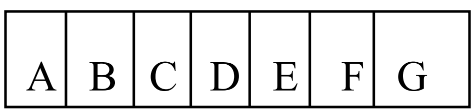
- **Insert O(n)**
- **Delete O(1)**
- **Search O(n)**
- **Overall O(n)**

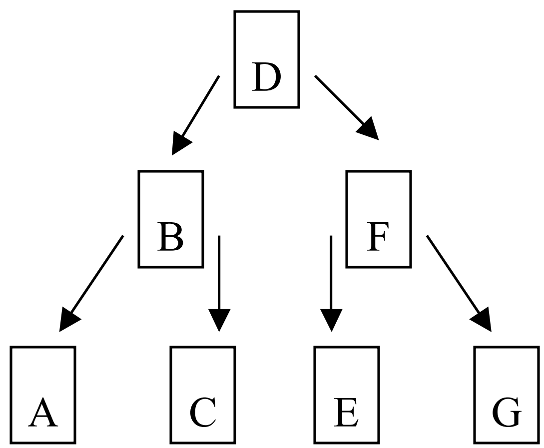
Links Speed Up Add/Delete

- **Idea: Use linked list to make add / delete faster**
- **Problem: Search of linked list is O(n)**
  - **Why not binary search on linked list?**

Links Speed Up Add/Delete

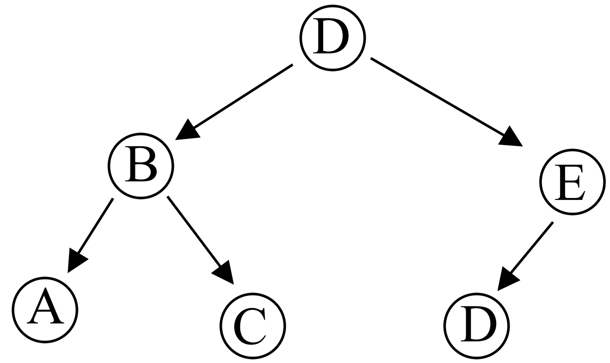
- **Idea: Use linked list to make add / delete faster**
- **Problem: Search of linked list is O(n)**
  - **Why not binary search on linked list?**
- **Idea: links to *two* places**

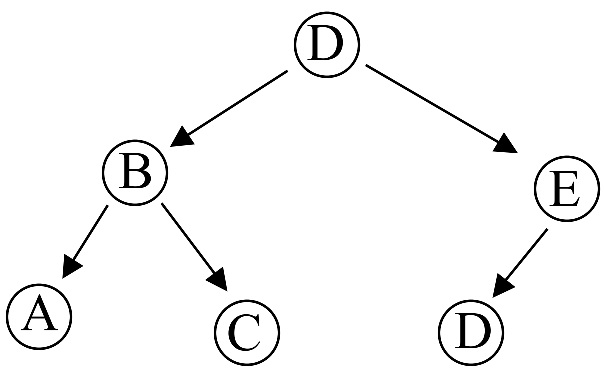


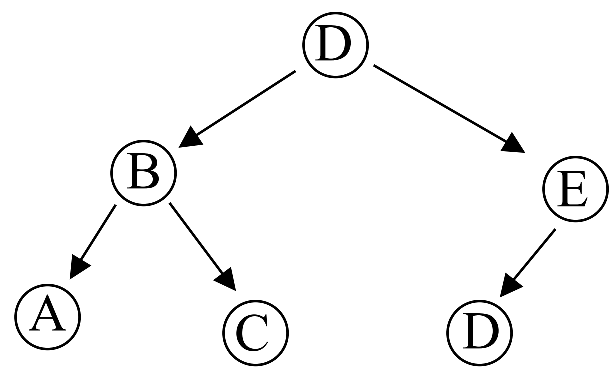


Trees

- Nodes and arcs (edges)
- Relationships:
  - Parent and Child
  - Root and Subtree

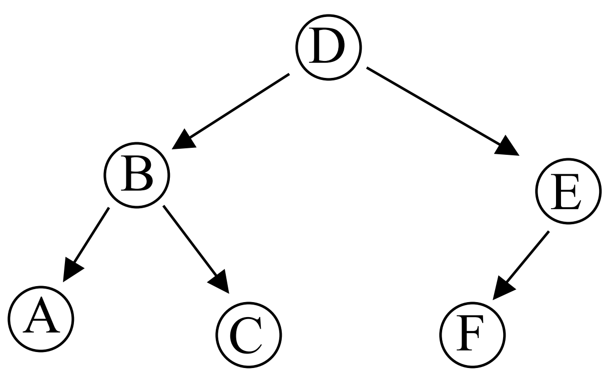






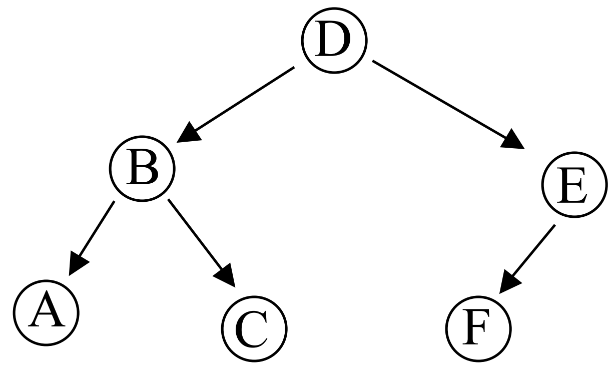
Trees

- Root has no parents
- Leaf node has no children
- All nodes except the root have a single parent
- There is exactly one path from root to any node

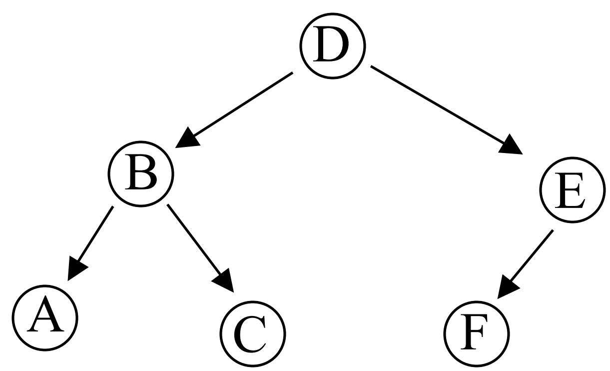


Trees

- **Height of tree**

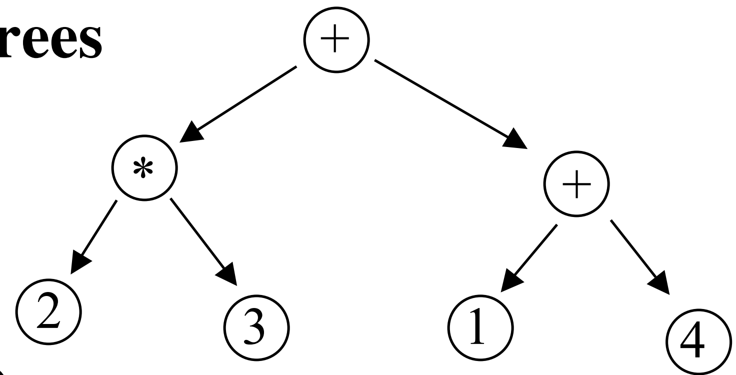


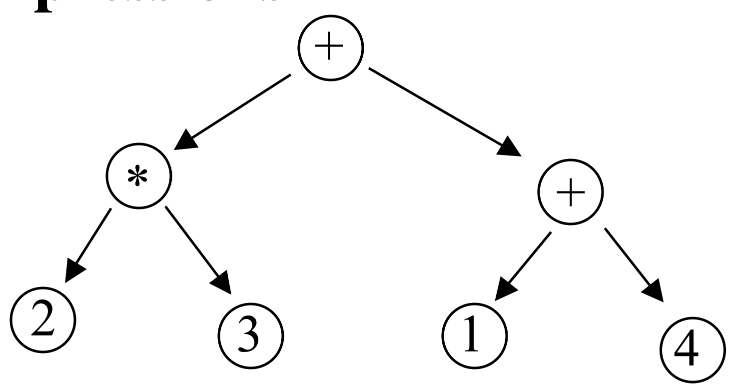
- **Depth of a node**



Binary tree

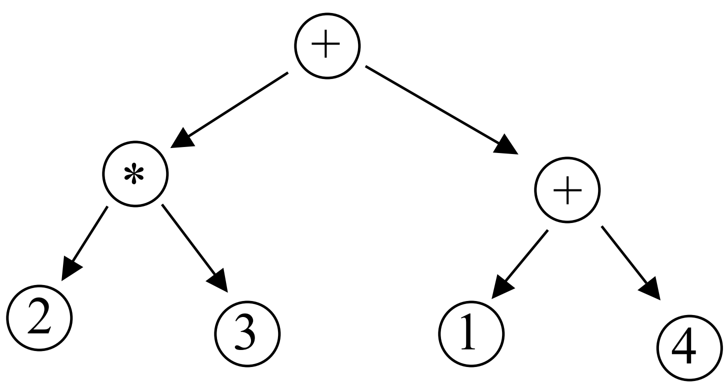
- each node has at most 2 subtrees
  – left and right subtree
- Examples of binary trees
  – 20 questions game (after animal/vegetable/mineral)
  – Arithmetic expressions

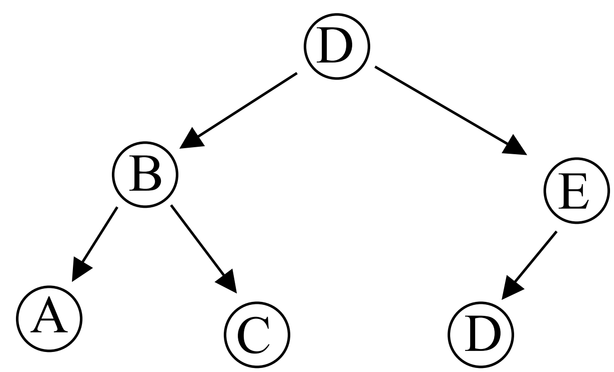




Binary tree

- Strict binary tree
  - only 0 or 2 subtrees
  - why not “only 2 subtrees”?
- Complete binary tree
  - every level but last is full,
  - last filled left-to-right





Recursive Data Structures

- **Recursive definition of a binary tree**
  - **empty (i.e. null)**
  - **not empty**
    - **the root**
    - **a left subtree, which is a binary tree**
    - **a right subtree, which is a binary tree**

Recursive functions

- **Common form of function on a tree is recursive**

```
f(tree):
   if (tree == null) return “terminal value”
     else  return g(data, f(tree.lst),  f(tree.rst))
```

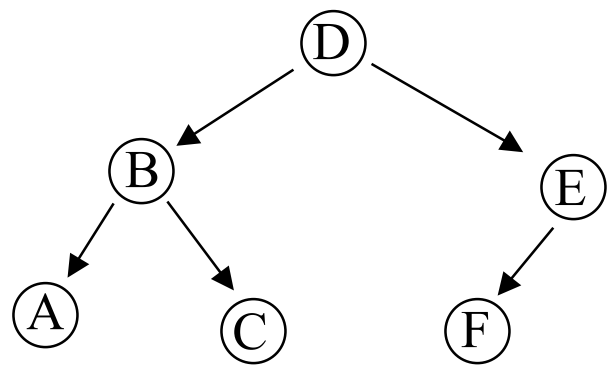
Recursive functions
height

```
f(tree):
   if (tree == null) return "terminal value"
     else  return g(data, f(tree.lst),  f(tree.rst))


height(tree):
   if (tree  == null) return -1
     else  return 1 + max(height(tree.lst,
                                  height(tree.rst))


g (d, lv, rv) =  1 + max(lv, rv)
```

Recursive functions
height



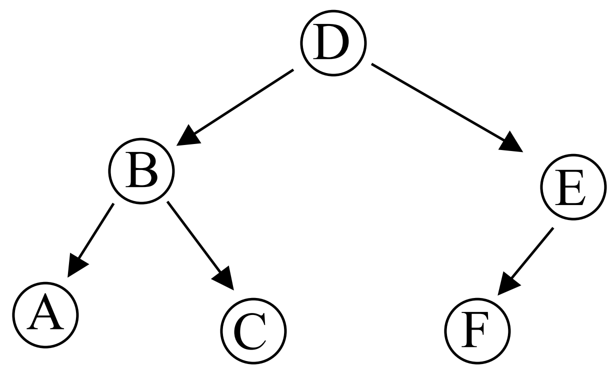
Recursive functions nodeCount

```
f(tree):
   if (tree == null) return "terminal value"
     else  return g(data, f(tree.lst),  f(tree.rst))

nodeCount(tree):
   if (tree == null) return 0
     else  return 1 + nodeCount(tree.lst) +
                      nodeCount(tree.rst))
```

g (d, lv, rv) = 1 + lv + rv

Recursive functions nodeCount



Recursive functions
Sum

```
f(tree):
   if (tree == null) return "terminal value"
     else  return g(data, f(tree.lst),  f(tree.rst))

sum(tree):
   You write this!

g (d, lv, rv) = ??
```

Recursive functions
Has 0?

```
f(tree):
   if (tree == null) return "terminal value"
     else  return g(data, f(tree.lst),  f(tree.rst))

has0(tree):
   You write this!

g (d, lv, rv) = ??
```