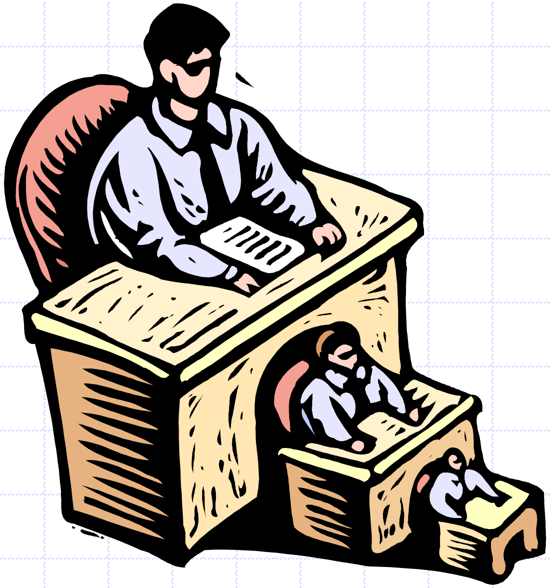
Recursion



The Recursion Pattern

- Classic example – the factorial function:

  $$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n$$

- Recursive definition:

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$$

```
int factorial(int n)
{
    if (n == 0)          // base case
        return 1;
    else if (n == 0)     // recursive case
        return n * factorial(n-1);
}
```

Content of a Recursive Method

- Base case(s)
  - Values of the input variables for which we perform no recursive calls are called base cases (there should be at least one base case).
  - Every possible chain of recursive calls must eventually reach a base case.
- Recursive calls
  - Calls to the current method.
  - Each recursive call should be defined so that it makes progress towards a base case.

A Perspective on Recursion

1. Decomposition
   - Decompose the problem into smaller identical problems
2. Base case
   - Smallest problem with known solution
3. Composition
   - Compose the solutions for smaller problems

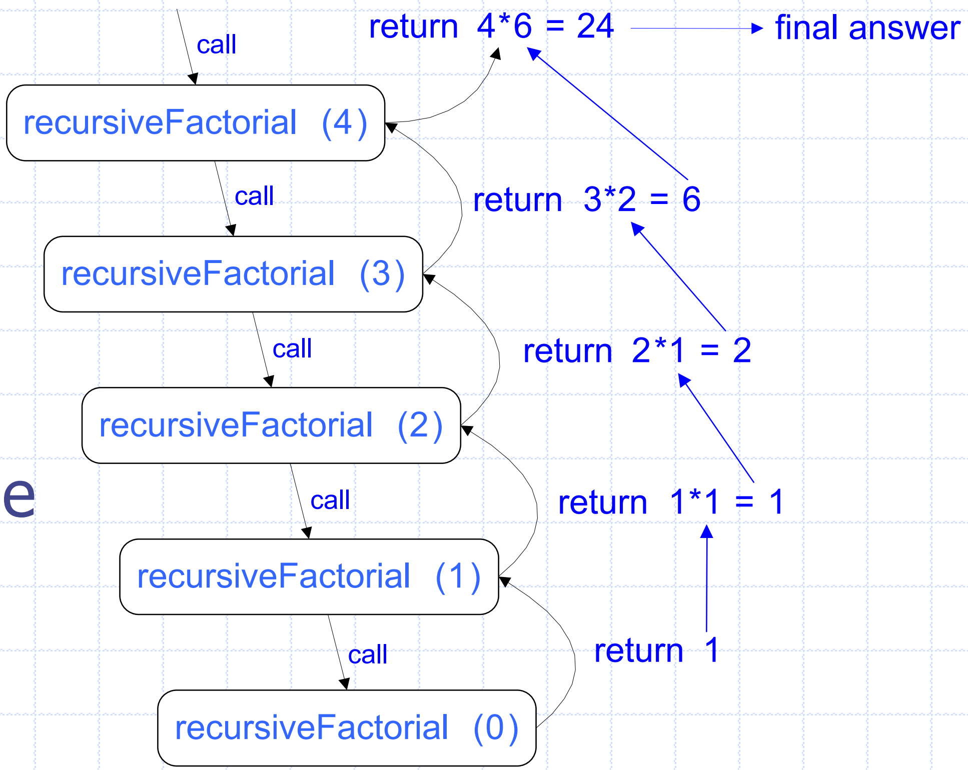
The Recursion Pattern

- Decomposition into smaller problems
- Base case: smallest problem
- Composition of solutions

```
int factorial(int n) // n >= 0
{
    if (n == 0)
        return 1;
    else
        {
            int smaller = factorial(n-1);
            return n * smaller;  // or just return n * factorial(n-1)
        }
}
```

Visualizing Recursion

- Recursion trace
  - A box for each recursive call
  - An arrow from each caller to callee
  - An arrow from each callee to caller showing return value
- Example



Linear Recursion

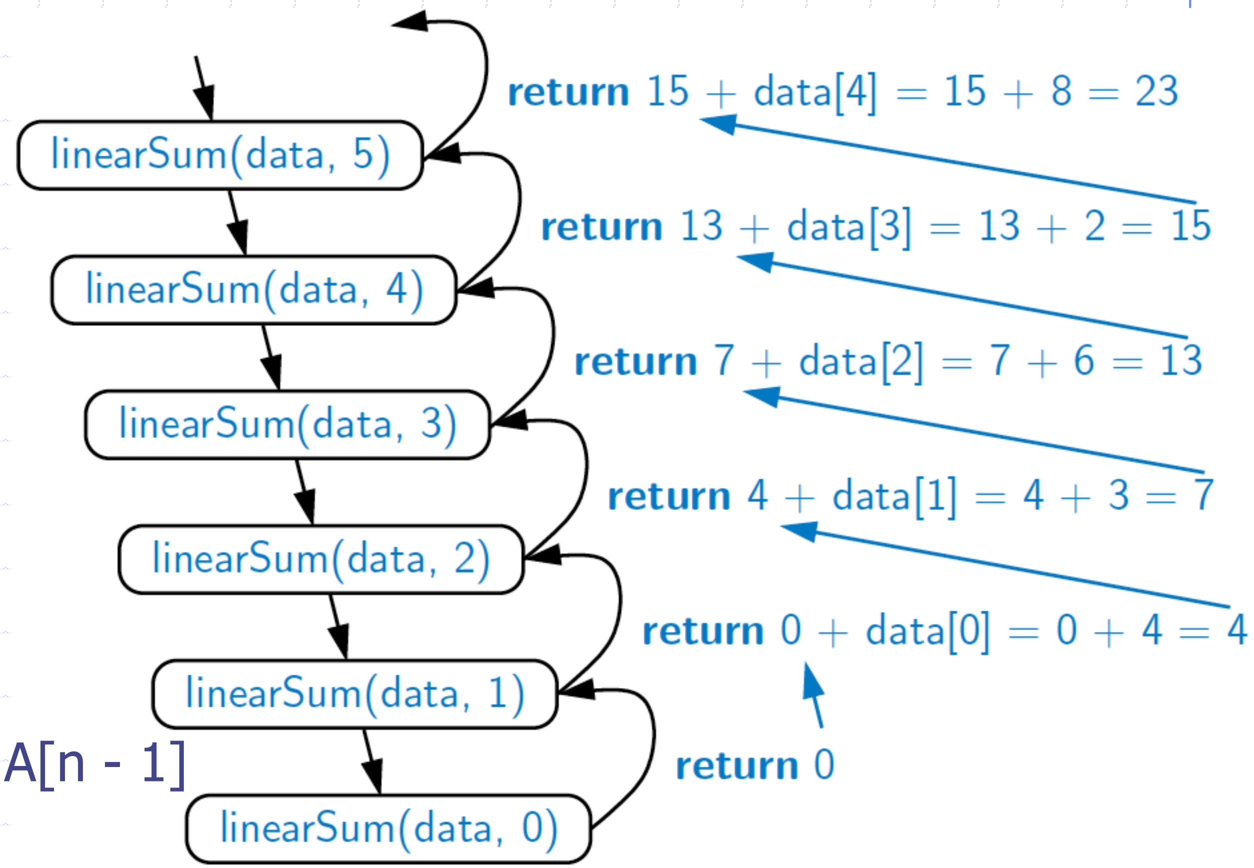
- Test for base cases
  - Every possible chain of recursive calls must eventually reach a base case.
- Recur once
  - Perform a single recursive call
  - Might branch to one of several possible recursive calls
  - makes progress towards a base case.

Example of Linear Recursion

Algorithm linearSum(A, n):
Input:
  Array, A, of integers
  Integer n such that
    $0 \leq n \leq |A|$
Output:
    Sum of the first n integers in A

if n = 0 then
  return 0
else
  return
linearSum(A, n - 1) + A[n - 1]

Recursion trace of linearSum(data, 5) called on array data = [4, 3, 6, 2, 8]



Example of Linear Recursion

Algorithm linearSum(A, n):

Input:

Array, A, of integers

Integer n such that $0 \le n \le |A|$

Output:

Sum of the first n integers in A

if $n = 0$ then

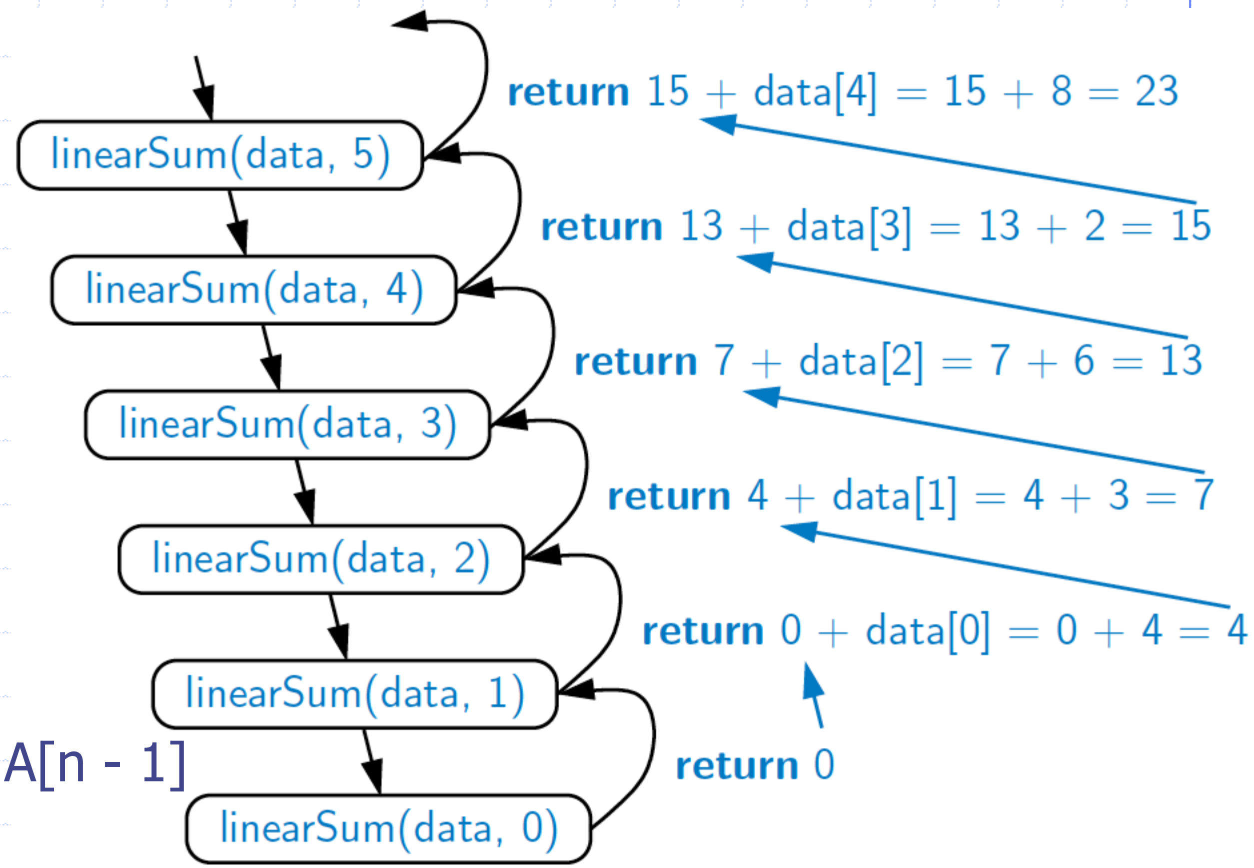
return 0

else

return

linearSum(A, n - 1) + A[n - 1]

Recursion trace of linearSum(data, 5) called on array data = [4, 3, 6, 2, 8]



Insertion Sort

algorithm insertionSort(A[0..n-1])

{

A[0]   if n=1

insert(insertionSort(A[0..n-2]), A[n-1])   o.w.

}

algorithm insert(A[0..n-1], key)

{

append(A[0..n-1], key)   if key>=A[n-1]

append(newarray(key), A[0])   if n=1&key<A[0]

append(insert(A[0..n-2],key), A[n-1])   o.w.

}

Reversing an Array

Algorithm reverseArray(A, low, high):

Input: An array A and nonnegative integer indices low and high

Output: The reversal of the elements in A starting at index low and ending at high

Reversing an Array

Algorithm reverseArray(A, low, high):

Input: An array A and nonnegative integer indices low and high

Output: The reversal of the elements in A starting at index low and ending at high

```
if low >=  high then return

Swap A[low] and A[high]
reverseArray(A, low + 1,  high – 1)
```

Tail Recursion

- linearly recursive method makes its recursive call as its last step.
- The array reversal method is an example.
- Such methods can be easily converted to non-recursive methods (which saves on some resources).
- Example:

Algorithm IterativeReverseArray(A, low, high):

Input: An array A and indices low and high

Output: The reversal of the elements in A starting at index low and ending at high

```
while low <  high do
    Swap A[low] and A[high]
    low  = low + 1
    high  = high - 1
return
```

Binary Recursion

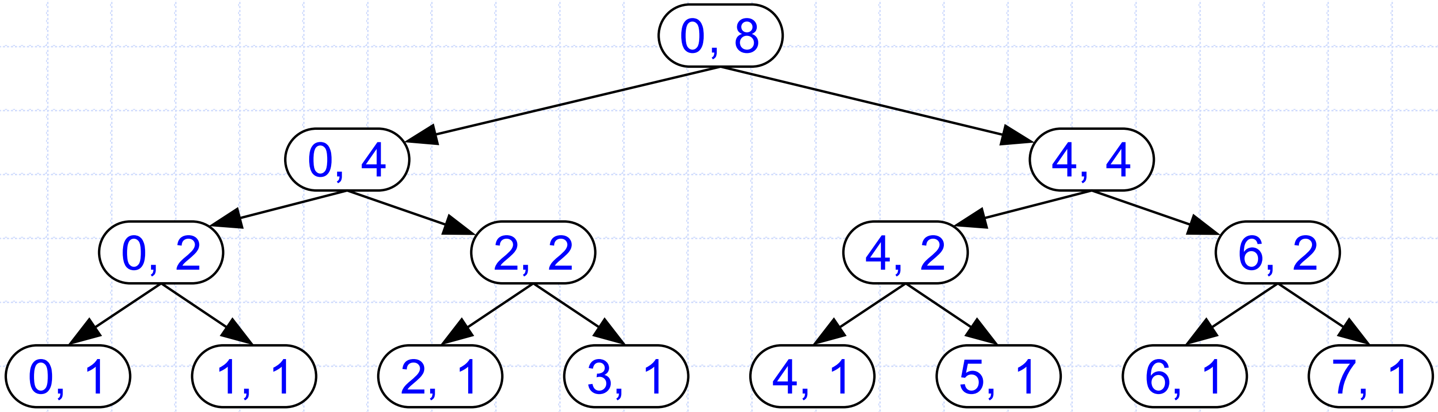
- **two** recursive calls for each non-base case.

Binary Recursion

- Problem: add all the numbers in an integer array A:

  Algorithm BinarySum(A, i, n):
  Input: An array A and integers i and n
  Output: The sum of the n integers in A starting at index i
  if n = 1 **then**
  return A[i]
  return BinarySum(A, i, n/ 2) + BinarySum(A, i + n/ 2, n/ 2)

- **Example trace:**



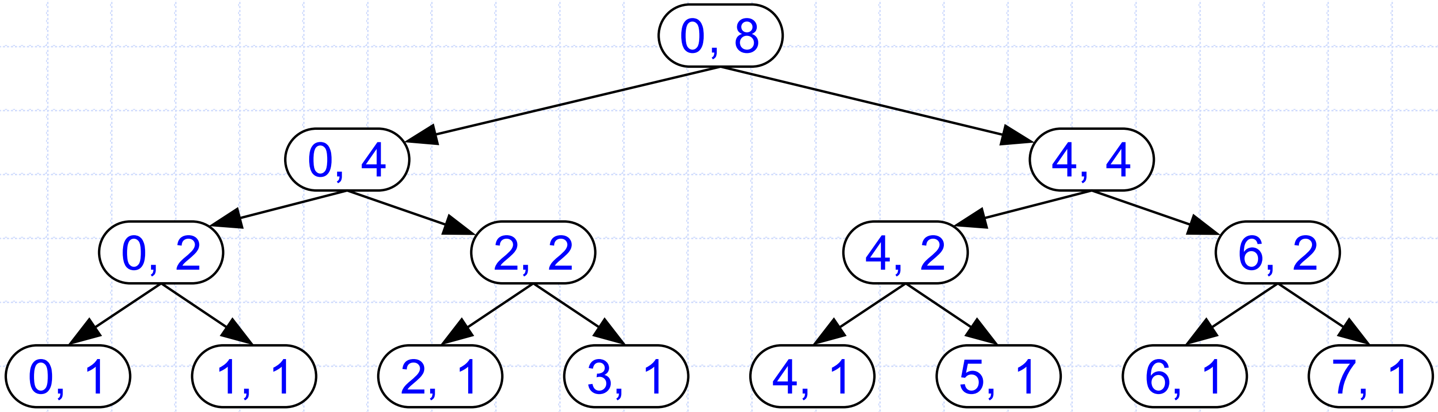
Binary Recursion

- Problem: add all the numbers in an integer array A:

  Algorithm BinarySum(A, i, n):
      Input: An array A and integers i and n
      Output: The sum of the n integers in A starting at inde
    if n = 1 **then**
    return A[i]
    return BinarySum(A, i, n/ 2) + BinarySum(A, i + n/ 2, n/ 2)

Decomposition?
Base case?
Composition?

- **Example trace:**



Summary

- 3 components of recursion
  - Decomposition (smaller problems)
  - Base case (smallest problem with known solution)
  - Composition (solution from smaller solutions)

Examples	Smaller	# of smaller problems
Factorial	-1	1
ArraySum	-1	1
InsertionSort	-1	1
Reverse array	-2	1
BinarySum	1/2	2