A Mathematical Example: Factorial

- Non-recursive definition:
  \[ n! = n \times (n-1) \times \ldots \times 2 \times 1 \]
  \[ = n \times (n-1) \times \ldots \times 2 \times 1 \]

- Recursive definition:
  \[ n! = n \times (n-1)! \quad \text{for } n \geq 0 \]
  \[ 0! = 1 \]

What happens if there is no base case?

Factorial as a Recursive Function

```python
def factorial(n):
    # Returns: factorial of n.
    # Pre: n \geq 0 and n is an int
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Recursive case

Base case

What happens if there is no base case?

Example: Fibonacci Sequence

- Sequence of numbers: 1, 1, 2, 3, 5, 8, 13, ...
- Get the next number by adding previous two
- What is \( a_9 \)?

- Recursive definition:
  \( a_n = a_{n-1} + a_{n-2} \)
  Base Case
  \( a_0 = 1 \)
  Base Case
  \( a_1 = 1 \)
  (another) Base Case

Why did we need two base cases this time?

Fibonacci as a Recursive Function

```python
def fibonacci(n):
    # Returns: Fibonacci no.
    # Precondition: n \geq 0 and n is an int
    if n <= 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

- Function that calls itself
  * Each call is new frame
  * Frames require memory
  * \( \infty \) calls = \( \infty \) memory

Fibonacci: # of Frames vs. # of Calls

Path to end = the call stack

Recursion is best for Divide and Conquer

- Fibonacci is very inefficient.
  * \( \text{fib}(n) \) has a stack that is always \( \leq n \)
  * But \( \text{fib}(n) \) makes a lot of redundant calls

Recurrence is best for Divide and Conquer

Goal: Solve problem P on a piece of data

<table>
<thead>
<tr>
<th>data 1</th>
<th>data 2</th>
</tr>
</thead>
</table>

Idea: Split data into two parts and solve problem

Combine Answer!
Divide and Conquer Example

Count the number of 'e's in a string:

p e n n e

Two 'e's

p e

One 'e'

n n e

One 'e'

Three Steps for Divide and Conquer

1. Decide what to do on “small” data
   - Some data cannot be broken up
   - Have to compute this answer directly
2. Decide how to break up your data
   - Both “halves” should be smaller than whole
   - Often no wrong way to do this (next lecture)
3. Decide how to combine your answers
   - Assume the smaller answers are correct
   - Combining them should give bigger answer

Exercise: Remove Blanks from a String

def deblank(s):
    """Returns: s w/o blanks"""
    if s == '':
        return s
    left = s[0]
    if s[0] == ' ':
        left = ''
    right = deblank(s[1:])
    return left+right

Minor Optimization

def deblank(s):
    """Returns: s w/o blanks"""
    if s == '':
        return s
    left = s[0]
    if s[0] == ' ':
        left = ''
    right = deblank(s[1:])
    return left+right

 Eliminate the second base by combining

Following the Recursion

debank a b c a b c
   a b c
   a b c
   a
   a b c
   a
   a
   a b c