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 an int
    if n == 0:
        return 1  # Base case(s)
    return n * factorial(n-1)  # Recursive case
```

What happens if there is no base case?

Example: Fibonacci Sequence

• Sequence of numbers: 1, 1, 2, 3, 5, 8, 13, ...
  \[ a_0, a_1, a_2, a_3, a_4, a_5 \]
  * Get the next number by adding previous two
  * What is \( a_5 \)?

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

Why did we need two base cases this time?

Fibonacci as a Recursive Function

```python
def fibonacci(n):
    #***Returns: Fibonacci no. a_n
    Precondition: n \geq 0 an int
    if n <= 1:
        return 1  # Base Case
    return (fibonacci(n-1) + fibonacci(n-2))  # Recursive Case
```

Recursion is best for Divide and Conquer

Goal: Solve problem P on a piece of data

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
+  n  n  e
```

**One ‘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
    elif len(s) == 1:
        return '' if s[0] == ' ' else s
    else:
        return deblank(s[0]) + deblank(s[1:])

# Handle small data
# Break up the data
# Combine answers
```

---

**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
```

---

**Following the Recursion**

```
def deblank(s):
    """Returns: s w/o blanks"""
    if s == ' ':
        return s
    elif len(s) == 1:
        return '' if s[0] == ' ' else s
    else:
        deblank(s[0])
        deblank(s[1:])
```

Eliminate the second base by combining

Less recursive calls