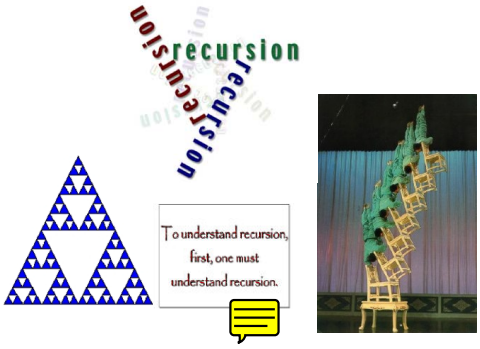

Iterative algorithms

- Definition: Algorithm where a problem is solved by iterating (going step-by-step) through a set of commands, often using loops

Algorithm power(a,n)
Input: non-negative integers a, n
Output: a^n
product ← 1
for i = 1 **to** n **do**
product ← product * a
return product

Challenge question: Can you compute a^n without using loops?

Recursion - definition

- An algorithm is recursive if in the process of solving the problem, it calls *itself* one or more times
- Example:

Algorithm power(a,n)
Input: non-negative integers a, n
Output: a^n
?
?

Simulating power(a,n)

```
Algorithm power(a,n)
 if (n=0) then return 1
 else
    previous ← power(a,n-1)
    return previous * a
```

When you call power(7,4), what happens?
power(7,4) calls
power(7,3), which calls
power(7,2), which calls
power(7,1), which calls
power(7,0), which returns 1
returns 7*1 = 7
returns 7*7 = 49
returns 7*49 = 343
returns 7*343 = 2401

Recursive binary search

Search for 7? 0 0 2 3 3 3 4 5 5 6 6 6 6 7 9 9 9

Algorithm binarySearch(array, start, stop, key)
Input: - A *sorted* array
- the region start...stop (inclusively) of indices to be searched
- the key to be found
Output: returns the index at which the key k has been found, or -1 if it is not in array[start...stop].

Fibonacci sequence

- The Fibonacci sequence is a sequence of numbers where each number is the sum of the two that precede it:

  $F(0) = 0$

  $F(1) = 1$

  $F(n) = F(n-1) + F(n-2)$ if $n \geq 2$

n	0	1	2	3	4	5	6	7	8	9	10	11
F(n)	0	1	1	2	3	5	8	13	21	34	55	89 ...

Iterative Fibonacci

```
Algorithm IterFib(n)
  Input: an non-negative integer n
  Output: the n-th Fibonacci number
  if (n=0) then return 0;
  if (n=1) then return 1;
  previous ← 0   // used to store previous Fib term
  current ← 1    // used to store current Fib term
  for i = 2 to n do
      tmpCurrent ← current
      current ← current + previous
      previous ← tmpCurrent
  return current
```

Recursive Fibonacci

```
Algorithm Fib(n)
  Input: an non-negative integer n
  Output: the n-th Fibonacci number
```

Recursion is not always efficient

Fib(5)

Question: When computing Fib(n), how many times are F(0) or F(1) called?