

Recursive Definitions

Recursive defs. specify the base value(s), and other results by induction from the base case(s).

Recursive definition of $f(n) = a_n = 2^n$

Base Case: $f(0) = 1$ $a_0 = 1$

Recursion: $f(n) = 2f(n - 1)$ $a_n = 2a_{n-1}$

Recursive definition of $f(n) = a_n = n!$

Base Case: $f(0) = 1$ $a_0 = 1$

Recursion: $f(n) = n f(n - 1)$ $a_n = n a_{n-1}$

Recursive definition of $f(m, n) = m + n$

Base Case: $f(0, n) = n$

Recursion: $f(m, n) = 1 + f(m - 1, n)$

Recursive definition of $f(m, n) = mn$

Base Case: $f(0, n) = 0$

Recursion: $f(m, n) = n + f(m - 1, n)$

Recursive Definitions to Algorithms

Basic recursive definition pattern:

Base Case: $f(a) = b$

Recursion: $f(n) = g(n, f(n - 1))$

Basic recursive algorithm pattern:

```
procedure  $f(n$ : an integer  $\geq a$ )  
  if  $n = a$   
    then  $answer := b$   
    else  $answer := g(n, f(n - 1))$   
  return  $answer$ 
```

```
procedure  $powers\_of\_2(n$ :  $\mathbf{N}$ )  
  if  $n = 0$   
    then  $answer := 1$   
    else  $answer := 2 * powers\_of\_2(n - 1)$   
  return  $answer$   
end procedure
```

```
procedure power(a:  $\mathbf{R}$ , n:  $\mathbf{N}$ )  
  if n = 0  
    then answer := 1  
    else answer := a * power(a, n - 1)  
  return answer  
end procedure
```

```
procedure fast_power(a:  $\mathbf{R}$ , n:  $\mathbf{Z}^+$ )  
  if n = 1  
    then answer := a  
    else if n is even  
      then x := fast_power(a, n/2)  
          answer := x * x  
      else x := fast_power(a, n - 1)  
          answer := a * x  
    return answer  
end procedure
```

```
procedure factorial(n: N)  
  if n = 0  
    then answer := 1  
    else answer := n * factorial(n - 1)  
  return answer  
end procedure
```

```
procedure gcd(a, b: N)  
  if a = 0  
    then answer := b  
    else answer := gcd(b mod a, a)  
  return answer  
end procedure
```

k is the key, l is the left end, r is the right end.

```

procedure binary_search( $k, l, r: \mathbf{Z}^+, a_1, \dots, a_n: \mathbf{Z}$ )
  if  $l = r$ 
  then if  $k = a_l$ 
    then  $answer := l$ 
    else  $answer := 0$ 
  else  $m := \lfloor (l + r)/2 \rfloor$ 
    if  $k \leq a_m$ 
    then  $answer :=$ 
       $binary\_search(k, l, m, a_1, \dots, a_n)$ 
    else  $answer :=$ 
       $binary\_search(k, m + 1, r, a_1, \dots, a_n)$ 
  return  $answer$ 

```

```

procedure stupid_fibonacci( $n: \mathbf{N}$ )
  if  $n = 0$ 
  then  $answer := 0$ 
  else if  $n = 1$ 
    then  $answer := 1$ 
    else  $answer := stupid\_fibonacci(n - 1)$ 
       $+ stupid\_fibonacci(n - 2)$ 
  return  $answer$ 
end procedure

```

```
procedure smart_fibonacci( $n$ :  $\mathbf{N}$ )  
  if  $n = 0$   
  then  $answer := (0, 0)$   
  else if  $n = 1$   
    then  $answer := (1, 0)$   
    else  $(a, b) := smart\_fibonacci(n - 1)$   
       $answer := (a + b, a)$   
  return  $answer$   
end procedure
```