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: $f(0) = 1 \quad a_0 = 1$
Recursion: $f(n) = 2f(n - 1) \quad a_n = 2a_{n-1}$

Recursive definition of $f(n) = a_n = n!$
Base Case: $f(0) = 1 \quad a_0 = 1$
Recursion: $f(n) = n f(n - 1) \quad a_n = n a_{n-1}$

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:

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

```plaintext
procedure powers_of_2(n: N)
    if \( n = 0 \)
        then answer := 1
    else   answer := 2 * powers_of_2(n - 1)
    return answer
end procedure
```
procedure \textit{power}(a: \mathbb{R}, n: \mathbb{N})
    \begin{align*}
    &\text{if } n = 0 \\
    &\quad \text{then } \text{answer} := 1 \\
    &\quad \text{else } \text{answer} := a \ast \text{power}(a, n - 1) \\
    &\text{return } \text{answer}
    \end{align*}
end procedure

procedure \textit{fast\_power}(a: \mathbb{R}, n: \mathbb{Z}^{+})
    \begin{align*}
    &\text{if } n = 1 \\
    &\quad \text{then } \text{answer} := a \\
    &\quad \text{else if } n \text{ is even} \\
    &\quad \quad \text{then } x := \text{fast\_power}(a, n/2) \\
    &\quad \quad \quad \text{answer} := x \ast x \\
    &\quad \quad \text{else } x := \text{fast\_power}(a, n - 1) \\
    &\quad \quad \text{answer} := a \ast x \\
    &\text{return } \text{answer}
    \end{align*}
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.

\begin{verbatim}
procedure binary_search(k, l, r: \mathbb{Z}^+, a_1, \ldots, a_n: \mathbb{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, \ldots, a_n)
                else answer :=
                    binary_search(k, m + 1, r, a_1, \ldots, a_n)
    return answer
\end{verbatim}

\begin{verbatim}
procedure stupid_fibonacci(n: \mathbb{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
\end{verbatim}
procedure smart_fibonacci(n: N)
if n = 0
then answer := (0, 0)
belse if n = 1
    then answer := (1, 0)
else (a, b) := smart_fibonacci(n - 1)
    answer := (a + b, a)
return answer
end procedure