CS 281
Prof. Alex Pothen

Webbook:

www.cs.odu.edu/~webbook
www.cs.odu.edu/~pothen/Courses/CS281

CLASS NOW IN CONSTANT 120
Mon Oct 26, 1998
Lecture 16
Sec 3.5 Program Correctness
Sec 3.4 Recursive Algorithms
Next Lecture: Sec 3.3 Recursive Definitions
Sec 3.4 Recursive Algorithms
Selection Sort

(1) \textbf{for} \quad \textit{step} := 1 \ \textbf{to} \ (n - 1) \ \textbf{do}

(2) \quad \textit{small} \ \text{is index of a smallest element}
    \text{in} \ A[\textit{step} : n];

(3) \quad \text{swap} \ A[\textit{small}] \ \text{with} \ A[\textit{step}];

\textbf{end}

Loop Invariant:
When \textit{step} is set to \textit{k} (in statement (1)),
(a). \(A[1 : k - 1]\) is in nondecreasing order;
(b). every element in \(A[k : n]\) is at least as big
as \(A[k - 1]\).

Basis: \quad \textit{step} = 1: \text{true since} \ A[1 : 0] \text{is empty.}
Selection Sort II

Selection Sort III

Ind. step: At the beginning of the loop, $step = k$. As a result of statement (2), $small$ contains the index of a smallest element in $A[k : n]$. Statement (3) then stores the smallest element in $A[k]$. Since this element was at least as big as $A[k - 1]$, we have that $A[1 : k]$ is non-decreasing. Further, since $A[k]$ now contains a smallest element in $A[k : n]$, every element in $A[k + 1 : n]$ is at least as big as $A[k]$. When $step$ is incremented to $n$, then the algorithm terminates with $A[1 : n - 1]$ sorted. But this implies that $A[1 : n]$ is sorted as well.
Factorial function

\[ n! = 1 \times 2 \times \cdots \times (n - 1) \times n. \]

Another (recursive) definition:

\[ n! \begin{cases} 
\quad = 1, & \text{for } n = 0 \text{ or } 1 \\
\quad = (n - 1)! \times n, & \text{for } n \geq 2. 
\end{cases} \]
Iterative Algorithm

Factorial function

\textbf{procedure} iterative_factorial(n)
\begin{align*}
\quad \text{fact} & := 1; \\
\quad \text{for } & \ \text{number} := 1 \ \text{to } n \ \text{do} \\
\quad & \quad \text{fact} := \text{fact} \ * \ \text{number}; \\
\quad \text{end for}
\end{align*}
Recursive Algorithm
Factorial function

procedure \( \text{fact}(n) \)

\[
\begin{align*}
\text{if} & \quad n \leq 1 \quad \text{then} \\
& \quad \text{fact} := 1; \quad \text{return}; \\
\text{else} & \\
& \quad \text{fact} := n \times \text{fact}(n - 1); \quad \text{return};
\end{align*}
\]

end if
Recursive definitions

- A recursive definition involves
  (1) one or more basis rules, which define some simple objects, and
  (2) one or more inductive rules, whereby larger objects are defined in terms of smaller objects in the collection.

- Fibonacci numbers

  Basis: \( f_0 = 0, f_1 = 1, \)
  Induction: \( f_i = f_{i-1} + f_{i-2}, \text{ for } i \geq 2. \)

Hence \( f_2 = 1, f_3 = 2, f_4 = 3, f_5 = 5, \)
\( f_6 = 8, f_7 = 13, f_8 = 21, f_9 = 34, \)
\( f_{10} = 55, \ldots. \)

\[ f_n > \left( \frac{1 + \sqrt{5}}{2} \right)^n \approx 1.6^n. \]
Fibonacci numbers
Iterative Algorithm

procedure fibonacci(n)
    if n == 0 then
        f := 0; return;
    else
        fprev := 0; \{f_{i-1}\}
f := 1; \{f_i\}
for i := 1 to n - 1 do
    fnew := f + fprev; \{f_{i+1}\}
    \{ Update f’s for next iteration\}
fprev := f; \{f_i\}
f := fnew; \{f_{i+1}\}
end for
end if
return;
Fibonacci numbers
Recursive Algorithm

procedure fibonacci(n)
    if $n = 0$ then
        fibonacci := 0; return;
    else if $n = 1$ then
        fibonacci := 1; return;
    else
        fibonacci :=
        fibonacci($n - 1$) + fibonacci($n - 2$);
    return;
end if
Fibonacci numbers
Truth in Advertising

• The recursive algorithm takes $f_{n+1} - 1$ additions to compute $f_n$. Recall that $f_{n+1} > (1.6)^{n+1}$.

• The inductive algorithm takes only $n-1$ additions to compute $f_n$. Hence in this case, the inductive algorithm is more efficient.

• On many computers, function calls are expensive, and this makes recursive algorithms slower than algorithms that use induction. However, in some cases, the inductive algorithms are not as convenient to write as recursive algorithms, since we must manage stacks for the former. Hence recursive algorithms are preferred when they do not affect efficiency.