Session 50: Recursive Algorithms

- Definition of recursive algorithms
- Examples of recursive algorithms

Recursive Algorithms

Definition: An algorithm is called **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input.

For the algorithm to terminate, the instance of the problem must eventually be reduced to some initial case for which the solution is known.

Recursive Factorial Algorithm

A recursive algorithm for computing $n!$, where n is a nonnegative integer.

factorial(n) :=

if $n = 0$

then return 1

else return $n \cdot factorial(n - 1)$

Recursive Computation of Factorial

factorial(4) =

*4*factorial(3) =*

4(3*factorial(2)) =*

4(3*(2*factorial(1))) =*

4(3*(2*(1*factorial(0)))) =*

4(3*(2*(1*1)))) =*

4(3*(2*1))) =*

4(3*2)) =*

*4*6 =*

24

Recursive Exponentiation Algorithm

A recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

power(a, n) :=

if n <= 0
then return 1
else return $a \cdot power(a, n-1)$

This is a bad algorithm!

Recursive Computation of Exponentiation

power(a, 6) =
*a * power(a, 5) =*
*a * (a * power(a, 4)) =*
*a * (a * (a * power(a, 3))) =*
*a * (a * (a * (a * power(a, 2)))) =*
*a * (a * (a * (a * (a * power(a, 1))))) =*
*a * (a * (a * (a * (a * (a* power(a, 0)))))) =*
*a * (a * (a * (a * (a * (a* 1))))) =*
*a * (a * (a * (a * (a * a)))) =*
*a * (a * (a * (a * a^2))) =*
*a * (a * (a * a^3)) =*
*a * (a * a^4) =*
*a * a^5 =*
a^6

Better Recursive Exponentation

fast_power(a, n) :=

if $n \leq 0$

then 1

else $a^{n\&1} \cdot (\mathit{fast\_powerl}(a, \lfloor n/2 \rfloor))^2$

$\lfloor n/2 \rfloor$ is the Integer part of n/2

n&1 is 0 if n is even and 1 if n is odd

Recursive Computation of Power

fast_power(a, 6) =

$a^0 * fast\_power(a, 3)^2 =$

$a^0 * (a^1 * fast\_power(a, 1)^2)^2 =$

$a^0 * (a^1 * (a^1 * fast\_power(a, 0)^2)^2)^2 =$

$a^0 * (a^1 * (a^1 * 1^2)^2)^2 =$

$a^0 * (a^1 * (a)^2)^2 =$

$a^0 * (a^3)^2 =$

a^6

Recursive Linear Search

procedure *linear search*(x: integer, a_1, a_2, …,a_n: distinct integers)

i := 1

while ($i \leq n$ and $x \neq a_i$)

  i := i + 1

  if $i \leq n$ **then** *location* := i **else** *location* := 0

return *location*

procedure *recursive_linear_search*(i, j, x: integers, $1 \leq i \leq j \leq n$)

  if $a_i = x$ **then return** i

  else **if** $i = j$ **then return** 0

    else return *recursive_linear_search*(i + 1, j, x)

Recursive Binary Search Algorithm

Assume we have $a_1, a_2, \ldots, a_n$, an increasing sequence of integers.

Initially i is 1 and j is n. We are searching for x.

recursive_binary_search(i, j, x) :=

$m := \lfloor (i + j)/2 \rfloor$

if $x = a_m$ **then return** m

else if ($x < a_m$ and $i < m$) **then return** *recursive_binary_search*($i, m-1, x$)

else if ($x > a_m$ and $j > m$) **then return** *recursive_binary_search*($m+1$, j, x)

else return 0

Example :

1 2 3 4 5 6 7 8 9

search (1, 9, 7)

$m = \left\lfloor \frac{(1+9)}{2} \right\rfloor = 5$

$7 > a_5$ and $9 > 5$

search (6, 9, 7)

search (6, 9, 7)

$m = \left\lfloor \frac{(6+9)}{2} \right\rfloor = 7$

$7 = a_7$

return 7

Summary

- Recursive algorithms for
  - Factorial function
  - Exponentation
  - Search