CS483-04 Non-recursive and Recursive Algorithm Analysis

Instructor: Fei Li

Room 443 ST II

Office hours: Tue. & Thur. 4:30pm - 5:30pm or by appointments

`lifei@cs.gmu.edu` with subject: CS483

`http://www.cs.gmu.edu/~ lifei/teaching/cs483_fall07/`

Outline

- Review and More
- Analysis of Non-recursive Algorithms
- Analysis of Recursive Algorithms
- Examples

Review

- $O(g(n)) := \{f(n) \mid$ there exist positive constants c and n_0 such that $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0$ $\}$.

$$f(n) \in O(g(n))$$

$f(n)$ grow *no faster* than $g(n)$.

- $\Omega(g(n)) := \{f(n) \mid$ there exist positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n)$ for all $n \ge n_0$ $\}$.

$$f(n) \in \Omega(g(n))$$

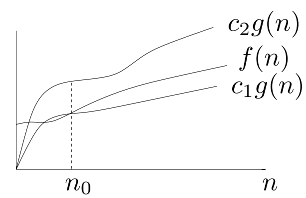
$f(n)$ grows *at least as fast* as $g(n)$.

- $\Theta(g(n)) := \{f(n) \mid$ there exist positive constants c_1, c_2, and n_0 such that $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ for all $n \ge n_0$ $\}$.

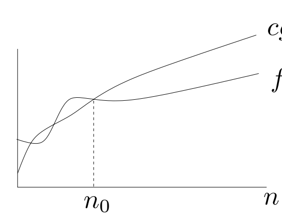
$$f(n) \in \Theta(g(n))$$

$f(n)$ grows *at the same rate* as $g(n)$.

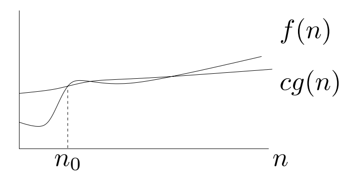
Asymptotic Notations



$f(n) \in \Theta(g(n))$



$f(n) \in O(g(n))$



$f(n) \in \Omega(g(n))$

Review

- Tools and techniques to get asymptotic notation
  - L'Hopital's rule

    If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ and the derivatives f' and g' exist, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

  - Stirling's formula

$$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$$

    where e is the base of natural logarithm, $e \approx 2.718$. $\pi \approx 3.1415$.

Exercises

- All logarithmic functions $\log_a n$ belong to the same class $\Theta(\log n)$ no matter what the logarithmic base $a > 1$ is.
- All polynomials of the same degree k belong to the same class $a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 \in \Theta(n^k)$.
- Exponential functions a^n have different orders of growth for different a's, i.e., $2^n \notin \Theta(3^n)$.
- $$\ln n < (\ln n)^2 < \sqrt{n} < n < n \cdot \ln n < n^2 < n^3 < 2^n < n! < n^n$$

Some Properties of Asymptotic Order of Growth

- Transitivity
  - $f(n) \in O(g(n))$ and $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$
  - $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$
  - $f(n) \in \Omega(g(n))$ and $g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$
- Reflexivity
  - $f(n) \in O(f(n))$
  - $f(n) \in \Theta(f(n))$
  - $f(n) \in \Omega(f(n))$
- Symmetry and Transpose Symmetry
  - $f(n) \in \Theta(g(n))$ if and only if $g(n) \in \Theta(f(n))$
  - $f(n) \in O(g(n))$ if and only if $g(n) \in \Omega(f(n))$

Outline

- Review
- Analysis of Non-recursive Algorithms
- Analysis of Recursive Algorithms
- Examples

Time Efficiency of Non-recursive Algorithms

- Decide on parameter n indicating input size.
- Identify algorithm's basic operation.
- Determine worst, average, and best cases for input of size n.
- Sum the number of basic operations executed.
- Simplify the sum using standard formula and rules (see Appendix A).

Time Efficiency of Non-recursive Algorithms

- $\sum_{i=l}^{u} 1 = 1 + 1 + \cdots + 1 = (u - l) + 1.$
- $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{n^2}{2} \in \Omega(n^2).$
- $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3} \in \Omega(n^3).$
- $\sum_{i=1}^{n} a^i = 1 + a + a^2 + \cdots + a^n = \frac{a^{n+1}-1}{a-1}, \forall a \neq 1.$

Example 1: Maximum Element

- Determine the value of the largest element in a given array.
- Input: An array $A[0, \cdots, n-1]$ of real numbers.
- Output: The value of the largest element in A.

Algorithm 0.1: MAXELEMENT($A[0, \cdots n-1]$)

$max = A[0]$
for $i = 1$ **to** $n - 1$
  do { **if** $A[i] > max$
    then $max = A[i]$
return (max)

Example 2: Element Uniqueness Problem

- Determine whether all the elements in a given array are distinct.
- Input: An array $A[0, \ldots, n-1]$.
- Output: Returns "true" if all the elements in A are distinct and "false" otherwise.

Algorithm 0.2: UNIQUEELEMENTS($A[0, \cdots n-1]$)

for $i = 0$ **to** $n - 2$
  do { **for** $j = i + 1$ **to** $n - 1$
    do { **if** $A[i] = A[j]$
      then return $(false)$
return $(true)$

Example 3: Matrix Multiplication

- Multiply 2 n-by-n matrices by the definition-based algorithm.
- Input: 2 n-by-n matrices A and B.
- Output: Matrix $C = A \cdot B$.

Algorithm 0.3: MatrixMulti(A, B)

for $i = 0$ **to** $n - 1$

do $\begin{cases} \textbf{for } j = 0 \textbf{ to } n-1 \\ \textbf{do} \begin{cases} C[i,j] = 0 \\ \textbf{for } k = 0 \textbf{ to } n-1 \\ \quad \textbf{do } C[i,j] = C[i,j] + A[i,k] \cdot B[k,j] \end{cases} \end{cases}$

return (C)

Example 4: Counting Binary Bits

- Input: A positive decimal integer n.
- Output: The number of binary digits in n's binary representation.

Algorithm 0.4: CountBinaryBits(n)

$count = 1$

while $n > 1$

do $\begin{cases} count = count + 1 \\ n = \lfloor n/2 \rfloor \end{cases}$

return $(count)$

Outline

- Review
- Analysis of Non-recursive Algorithms
- Analysis of Recursive Algorithms
- Examples

Recurrences

- A recurrence is an equation or inequality that describes a function in terms of its value over a smaller value.
- **Example**: Find $n!$

Recurrences

- A recurrence is an equation or inequality that describes a function in terms of its value over a smaller value.
- **Example**: Find $n!$
  - Non-recursive: $1 \cdot 2 \cdot 3 \cdots n$

Algorithm 0.5: FINDFACTORIAL-$\alpha(n)$

$factorial = 1$
for $i = 1$ **to** n
  do $factorial = factorial \cdot i$
return $(factorial)$

  - Recurrence: $f(n) = n \cdot f(n-1)$

Algorithm 0.6: FINDFACTORIAL-$\beta(n)$

if $n = 0$
  then return (1)
  else return $(n \cdot \text{FindFactorial} - \beta(n-1))$

Example: Counting Number of Bits

- Input: A positive decimal integer n.
- Output: The number of binary digits in n's binary representation.

Algorithm 0.7: NON-RECURSIVECOUNT(n)

$count = 1$
while $n > 1$
  do $\begin{cases} count = count + 1 \\ n = \lfloor n/2 \rfloor \end{cases}$
return $(count)$

Algorithm 0.8: RECURSIVECOUNT(n)

if $i = 1$
  do return (1)
  else return $(RecursiveCount(\lfloor n/2 \rfloor) + 1)$

Example: Fibonacci Numbers

- Output: A sequence of numbers $F_0, F_1, F_2, \ldots, F_n, \ldots$ such that

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \\ 0, & \text{if } n = 0. \end{cases}$$

Example: Fibonacci Numbers

- Output: A sequence of numbers $F_0, F_1, F_2, \ldots, F_n, \ldots$ such that

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \\ 0, & \text{if } n = 0. \end{cases}$$

Algorithm 0.9: FIBNUMBER(n)

if $n = 0$
  return (0)
if $n = 1$
  return (1)
else return $(FibNumber(n-1) + FibNumber(n-2))$

Example: Hanoi Tower Problem

- Move all the disks from peg a to peg c. Large disk cannot be on top of a smaller one.
- Input: n disks in order of sizes on peg a. 3 pegs a, b, and c

http://en.wikipedia.org/wiki/Tower_of_Hanoi

Algorithm 0.10: HANOITOWER(n, a, c, b)

if $n = 1$
  Move the disk from a to c
else {
  HanoiTower$(n-1, a, b, c)$
  Move the largest disk from a to c
  HanoiTower$(n-1, b, c, a)$
}

Analysis of Recursive Algorithms

- The iteration method
  - Expand (iterate) the recurrence and express it as a summation of terms depending only on n and the initial conditions.
- The substitution method
- Master Theorem

  (To be introduced in Chapter 4.)

Iteration Method: Examples

- $n!$

$$T(n) = T(n-1) + 1$$

- Tower of Hanoi

$$T(n) = 2T(n-1) + 1$$

Iteration: Example

- $n!$ $(T(n) = T(n-1) + 1)$

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= (T(n-2) + 1) + 1 \\ &= T(n-2) + 2 \\ \cdots & \quad \cdots \\ &= T(n-i) + i \\ \cdots & \quad \cdots \\ &= T(0) + n = n \end{aligned}$$

- Tower of Hanoi $(T(n) = 2T(n-1) + 1)$???

Iteration: Example

- $n!$ $(T(n) = T(n-1) + 1)$
- Tower of Hanoi $(T(n) = 2T(n-1) + 1)$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2T(n-2) + 2 + 1 \\ \cdots & \quad \cdots \\ &= 2^iT(n-i) + 2^{i-1} + \cdots + 1 \\ \cdots & \quad \cdots \\ &= 2^{n-1}T(1) + 2^{n-1} + 2^{n-1} + \cdots + 1 \\ &= 2^{n-1}T(1) + \sum_{i=0}^{n-2} 2^i \\ &= 2^{n-1} + 2^{n-1} - 1 \\ &= 2^n - 1 \end{aligned}$$

Assignment 1

- Problems
  1. Prove or find a counter-example:

$$(\frac{n}{3})^n < n! < (\frac{n}{2})^n, \quad \text{if } n \geq 6.$$

  2. p. 8, Exercises (1.1) 5, 6.
  3. p. 60, Exercises (2.2) 5, 6
  4. p. 67, Exercises (2.3) 2, 4
  5. p. 76, Exercises (2.4) 1, 3, 5
- Due date: September 20, 2007. In class