CS 395 – Analysis of Algorithms

Chapter 2 – Fundamentals of the Analysis of Algorithm Efficiency

Read 2.3 & 2.4

- **Using limits for comparing orders of growth**
  - **L'Hôpital's rule**
- **Mathematical analysis of recursive algorithms**
  - **Example 3: Count Binary Digits**
  - **Example 4: Fibonacci Numbers**
  - **Example: factorial function (Stirling's formula)**

I've told a lot of terrible math jokes, but this **Fibonacci** joke is as bad as the last two I've told combined

Bet you did nacci that joke coming

Establishing order of growth using limits

$$\lim_{n\to\infty} t(n)/g(n) = \begin{cases} 0 & \text{order of growth of } t(n) < \text{order of growth of } g(n) \\ c > 0 & \text{order of growth of } t(n) = \text{order of growth of } g(n) \\ \infty & \text{order of growth of } t(n) > \text{order of growth of } g(n) \end{cases}$$

L'Hôpital's rule and Stirling's formula

L'Hôpital's rule:

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

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

Examples

Example 1: Compare the orders of growth of $10n$ and n^2.

$$\lim_{n \to \infty} \frac{10n}{n^2}$$

Since the limit is 0 the function has a smaller order of growth, i.e., $10n \in O(n^2)$.

Little-oh notation

Example 2: Compare the orders of growth of $\log_2 n$ and $\sqrt{n}$.

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}}$$

Since the limit is equal to *0*, $\log_2 n$ has a smaller order of growth than $\sqrt{n}$, or $\log_2 n \in o(\sqrt{n})$.

little-oh notation: $t(n) \in o(g(n))$ means that $g(n)$ grows much faster than $t(n)$.

5

$$\frac{d}{dx}\log_2(x) = \frac{1}{x \cdot \ln(2)}$$

Factorial: Stirling's formula

- **Definition:** $n! = 1 \cdot 2 \cdot \ldots \cdot (n\text{-}1) \cdot n$ for $n \geq 1$ and $0! = 1$
- $n! \approx (2\pi n)^{1/2} (n/e)^n$ for large values of n.

Stirling's formula

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \quad \text{for large values of } n.$$

Examples

Stirling's formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \quad \text{for large values of } n.$$

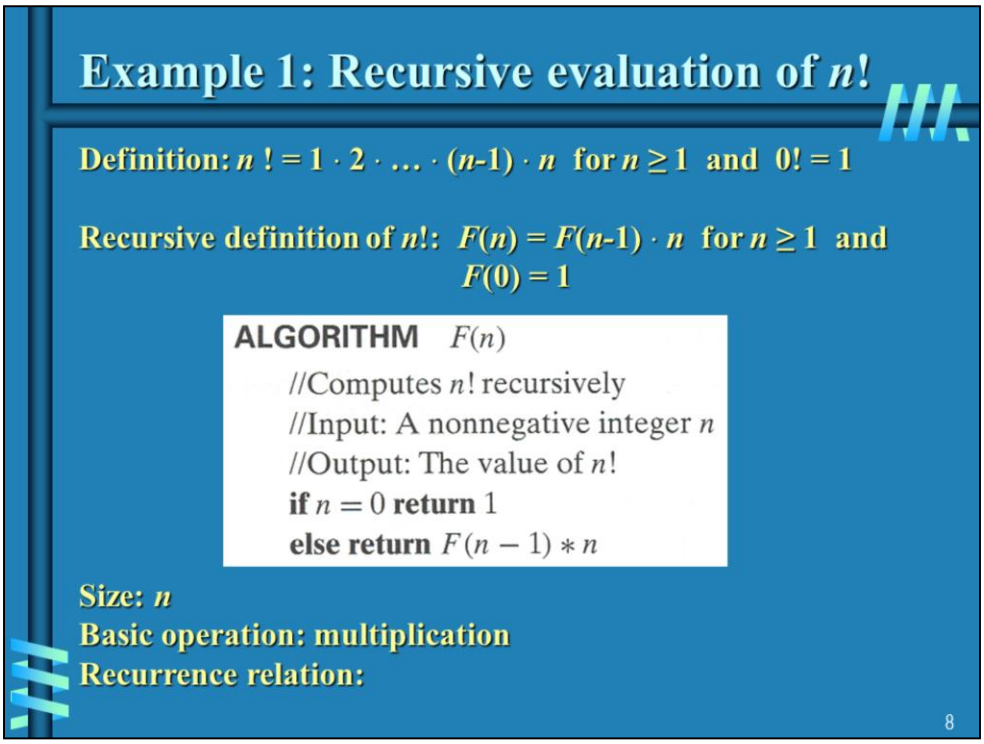
Example 3: Compare the orders of growth of $n!$ and 2^n.

$$\lim_{n\to\infty}\frac{n!}{2^n} = \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n}$$

$$= \lim_{n\to\infty}\sqrt{2\pi n}\,\frac{n^n}{2^n e^n}$$

$$= \lim_{n\to\infty}\sqrt{2\pi n}\,(\frac{n}{2e})^n = \infty$$

- 2^n grows fast, but $n!$ grows faster.
- $n! \in \Omega(2^n)$



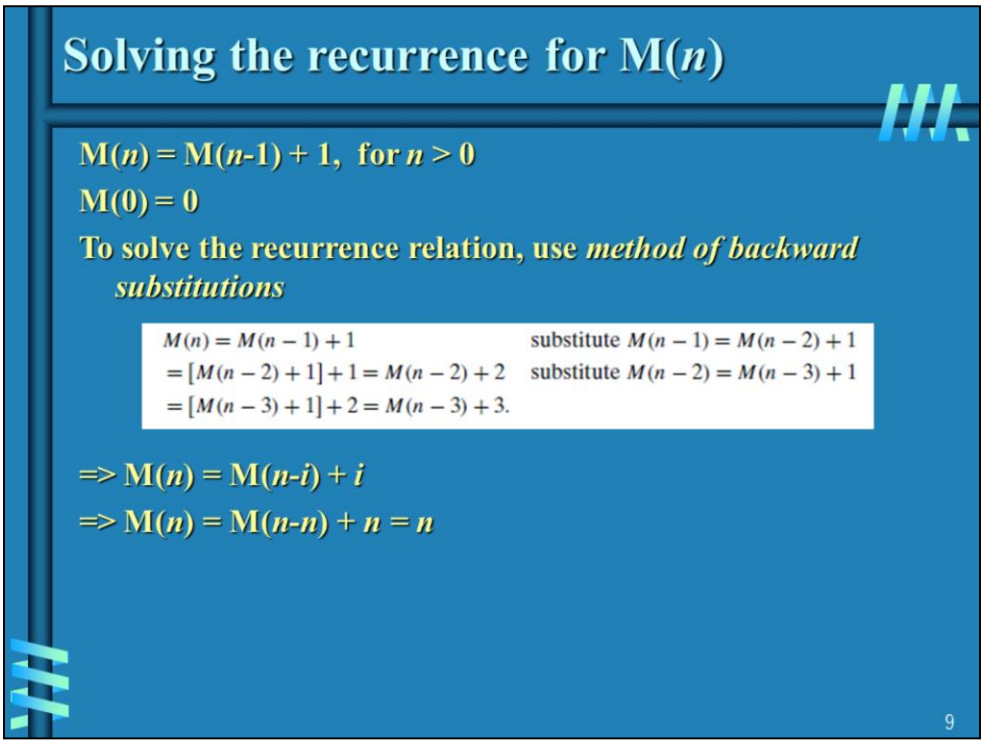
Note the difference between the two recurrences. Students often confuse these!

F(n) = F(n-1) n
F(0) = 1

for the values of n!

M(n) =M(n-1) + 1
M(0) = 0

for the number of multiplications made by this algorithm



Try this on the doc cam:

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

T(0) = 0

The method of forward substitution proceeds by generating the first half-dozen or so terms in the s

n	T(n)
0	0
1	1
2	4
3	9
4	16
5	25

This example was carefully rigged to give $T(n)=n^2$.

Unfortunately, there are infinitely many sequences that start with these six numbers, so we can't be

So let's do so using backwards substitution:

$T_n = T_{n-1} + 2n-1$

$=T_{n-2} + 2(n-1) – 1 + 2n – 1$

$=T_{n-2} + 2(2n) +2(-1) -2(1)$

$=T_{n-3} + 3(2n) +2(-2-1) -3(1)$

$=T_{n-4} + 4(2n) +2(-3-2-1) -4(1)$

$=T_{n-n} + n(2n) -2(n(n-1)/2) -n$

$= 0 +2n^2 –n^2 +n - n$

$= n^2$

Example 3: Counting #bits

ALGORITHM *BinRec*(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return *BinRec*$(\lfloor n/2 \rfloor) + 1$

Size: *n*

Basic operation: addition

Recurrence relation:

$A(n) = A(\text{floor}(n/2)) + 1$ **for** $n > 1$

$A(1) = 0$

Solving the recurrence

- The division and floor function in the argument of the recursive call makes the analysis difficult.
- Solve for $n = 2^k$ and apply *smoothness rule*.
  - Order of growth for $n = 2^k \approx$ Order of growth for all values on n
- Substitute $n = 2^k$

  $A(2^k) = A(2^k\text{-}1) + 1 \quad$ for $k > 0$

  $A(2^0) = 0$
- Use backward substitutions, we get

  $A(2^k) = A(1) + k = k$
- Substitute back, $n = 2^k$ and hence $k = log_2 n$

  $A(n) = \log_2 n \in \Theta(\log n)$

Example 4: Fibonacci numbers

The Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

The Fibonacci recurrence:

$F(n) = F(n-1) + F(n-2)$ for $n > 1$

$F(0) = 0$

$F(1) = 1$

- **Backward substitutions method does not work**
- **General 2nd order linear homogeneous recurrence with constant coefficients:**

$$aX(n) + bX(n-1) + cX(n-2) = 0$$

Linear homogeneous recurrence with constant coefficients

Linear refers to the fact that X(n−1), and(n−2) appear in separate terms and to the first power.

Homogeneous refers to the fact that the total degree of each term is the same (thus there is no constant term)

- **General 2nd order linear homogeneous recurrence with constant coefficients:**

$$aX(n) + bX(n\text{-}1) + cX(n\text{-}2) = 0$$

Constant Coefficients refers to the fact that a, b and c are fixed real numbers that do not depend on n.

13

Linear homogeneous recurrence with constant coefficients

- Are these linear homogeneous recurrence with constant coefficients?

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

  Yes (With degree 2)

  $A(n) = (1.04)A(n-1)$

  Yes (With degree 1. It can be any degree not just 2)

  $A(n) = A(n-1) * A(n-2)$

  No. It is non-linear.

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

  No. It is not homogeneous. … but we can deal with that.

  $C(n) = nC(n-1)$

  No. It does not have constant coefficients.

  $D^n = aD^{(n-1)} + cD^{(n-2)} + cD^{(n-3)}$

  Surprisingly yes.

14

Solving $aX(n+2) + bX(n+1) + cX(n) = 0$

- Set up the characteristic equation (quadratic)

$$ar^2 + br + c = 0$$

- Solve to obtain roots r_1 and r_2
- General solution to the recurrence

  if r_1 and r_2 are two distinct real roots: $X(n) = \alpha r_1^n + \beta r_2^n$

  if $r_1 = r_2 = r$ are two equal real roots: $X(n) = \alpha r^n + \beta n r^n$

- Particular solution of α and β can be found by using initial conditions

Application to the Fibonacci numbers

- $F(n) = F(n\text{-}1) + F(n\text{-}2)$ or $F(n) - F(n\text{-}1) - F(n\text{-}2) = 0$
- Characteristic equation: $r^2 - r - 1 = 0$
- Roots of the characteristic equation:

  $r_{1,2} = (1 \pm \sqrt{(1-4(-1))})/2 = (1 \pm \sqrt{5})/2$

  $\Rightarrow \varphi$ or φ' { $\varphi = (1+\sqrt{5})/2$ and $\varphi' = (1-\sqrt{5})/2$}
- General solution to the recurrence:

  $F(n) = \alpha\varphi^n + \beta\varphi'^n$ where α and β are unknowns

  Using the initial conditions:

  $\alpha + \beta = 0$

  $\varphi\alpha + \varphi'\beta = 1$
- Solve by substituting the first equation into the second,

  $\alpha = 1/\sqrt{5}$ and $\beta = -1/\sqrt{5}$
- So, $F(n) = 1/\sqrt{5}\,(\varphi^n - \varphi'^n) = \varphi^n/\sqrt{5}$ rounded to the nearest integer

16

Computing Fibonacci numbers

1. Definition-based recursive algorithm
2. Nonrecursive definition-based algorithm
3. Explicit formula algorithm
4. Logarithmic algorithm based on formula:

$$\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n$$

for $n \geq 1$, assuming an efficient way of computing matrix powers.

17

Option 4: Here is a good article: https://kukuruku.co/post/the-nth-fibonacci-number-in-olog-n/ (See next –hidden - slide)

Recursive Algorithm

ALGORITHM $F(n)$

//Computes the nth Fibonacci number recursively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
if $n \leq 1$ **return** n
else return $F(n-1) + F(n-2)$

Size: n
Basic operation: addition
No difference between worst and best case
Recurrence relation:

$A(n) = A(n-1) + A(n-2) + 1$ for $n > 1$
$A(0) = A(1) = 0$

Inhomogeneous recurrence

19

Solving the recurrence

- In general solution to the inhomogeneous problem is equal to the sum of solution to homogenous problem plus solution only to the inhomogeneous part. The undetermined coefficients of the solution for the homogenous problem are used to satisfy the initial conditions.

In this case, $A(n) = B(n) + I(n)$ where

$A(n)$ is solution to complete inhomogeneous problem

$B(n)$ is solution to homogeneous problem

$I(n)$ solution to only the inhomogeneous part of the problem

20

Solving the recurrence

- We guess at $I(n)$ and then determine the new initial conditions for the homogenous problem for $B(n)$
- For this problem the correct guess is $I(n) = 1$
- Substitute $A(n) = B(n) - 1$ into the recursion and get

  $B(n) - B(n-1) - B(n-2) = 0$ with $B(0) = B(1) = 1$
- The same as the relation for $F(n)$ with different initial conditions
- We do not really need the exact solution; We can conclude

  $A(n) = B(n) - 1 = F(n+1) - 1 \in \Theta(\varphi^n)$, *exponential*

Iterative Algorithm

ALGORITHM *Fib*(n)

```
//Computes the nth Fibonacci number iteratively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
F[0] ← 0; F[1] ← 1
for i ← 2 to n do
    F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Size: *n*

Basic operation: addition

No difference between worst and best case

A*(*n* - 1), *linear algorithm

Algorithm Visualization

1. http://www.algomation.com/
2. https://visualgo.net/

Just for fun

- You are a parking attendant at a garage with n stalls:

- A motor bike takes 1 slot. A car takes two slots.
- The lot fills up every day.
- What is the longest number of days you can go without having the same configuration of cars/motorcycles repeated? (I.e. how many unique configurations are there?) Give a formula in terms of n.