

## Efficiency of Algorithms

- Time
  - usually most important
  - space sometimes
- How measured?
  - seconds?
  - running two algorithms and comparing?
- What case?
  - average?
  - best case?
  - worst case?
- Mathematical approach
  - idealized computer
  - no overflow
  - no differences between programming languages, hardware, operating system

© 2005-06. 324-1-25

## Order

- Measure efficiency:
  - in “orders” of functions
  - example: in the order of  $n^2$
- Definition:
  - an algorithm is in the order of  $f(n)$ 
    - there exists a positive number  $c$  so that:
      - for every instance of size  $n$  that's large enough:
      - $n$  can be solved in time  $cf(n)$
- Questions:
  - size of the instance?
  - why a constant?
  - what is  $c$ ?

© 2005-06. 324-1-26

## Order

- Size of instance
  - measured in bits
  - but usually: does not matter how measured
  - as long as measured the same in all algorithms we are comparing
- $c$ 
  - depends on hardware, operating system, programming language
  - small  $c$ : fast computer
  - big  $c$ : slow computer
- Examples of orders
  - linear
  - quadratic
  - exponential
  - logarithmic
- Asymptotical efficiency
  - for arbitrarily large inputs
  - “for all except finitely many”

© 2005-06. 324-1-27

## Fibonacci Sequence

$f(0) = 0$   
 $f(1) = 1$   
 $f(n) = f(n-1) + f(n-2)$  for  $n \geq 2$

- Obvious algorithm:

```
int fib( int n ) {
    if( n < 1 )
        return 0;
    if( n == 1 )
        return 1;
    else
        return fib( n - 1 ) + fib( n - 2 );
}
```
- Correct?
- Does it terminate?
- Efficient?

© 2005-06. 324-1-28

## Fibonacci

- Efficiency of recursive algorithm
  - in the order of the fib() itself
  - de Moivre's formula says: it's exponential!
- order of  $c^n$

$$fibonacci(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}$$

$$\phi = \frac{1 + \sqrt{5}}{2}$$

```
int fibiter( int n ) {
    int previous = 1;
    int current = 0;
    for( int k = 1; k<=n; k++ ) {
        int temp = current;
        current = previous + current;
        previous = temp;
    }
    return current;
}
```

© 2005-06. 324-1-29

## Efficiency of Fibonacci

- Recursive:
  - $O(\phi^n)$  – exponential
- Iterative:
  - $O(n)$  – linear
- Big difference!

	n=10	20	50	100
rec.	8ms	1s	2min	$10^9$ years
iterative	.2ms	.3ms	.5ms	1.5ms

- Elementary operations??

© 2005-06. 324-1-30

## Logarithm Rules (Review)

- $a^x = y \Leftrightarrow x = \log_a y$  (definition of logarithm)
- $\log_a(xy) = \log_a x + \log_a y$
- $\log_a 1 = 0$
- $\log_a(1/x) = -\log_a x$
- $\log_a x = \log_b x * (1/\log_b a)$
- derivative:  $(\log_a x)' = 1/(x \ln a)$
- $\lg x := \log_2 x$

© 2005-06. 324-1-31

## GCD Algorithm

- Greatest common divisor

```
int gcd( int m, int n ) {  
    int i = min( m, n ) + 1;  
    while( ! divides( i, m ) && divides( i, n ) ) {  
        i--;  
    }  
    return i;  
}
```

- Does it terminate? Efficiency?

© 2005-06. 324-1-32

## GCD Algorithm

- Euclid

```
int gcd( int m, int n ) {  
    if( n==0 ) {  
        return m;  
    }  
    return gcd( n, m % n );  
}
```

- Does it terminate?
- Efficiency:  $O(\log n)$
- 3500 years old!

© 2005-06. 324-1-33

## Sorting, 1<sup>st</sup> Attempt

- Insertion sort
  - sort like a hand of cards
  - go through set, inserting each into the already sorted part
  - number of steps needed?
  - faster way to do it?
- Selection sort
  - pick smallest, move to front
  - repeat
- Pigeonhole sort
  - limited applicability
- Quicksort
  - divide-and-conquer
  - $O(n \log n)$

© 2005-06. 324-1-34

## Asymptotic Notations

- $O( )$ 
  - Def.:  $O( f(n) )$  is the set of all functions  $t(n) \leq c f(n)$ 
    - $c$ : some constant
    - as long as  $n$  is large enough
      - (there is a threshold  $n_0$ )
- Examples:
  - $n^2$  is in the order of  $O(n^3)$
  - frequently spelled:  $n^2 = O(n^3)$ 
    - incorrect!
  - $n$  is in the order of  $O(n^3)$
  - ...

© 2005-06. 324-1-35

## Maximum Rule

- Theorem:  $O( f(n) + g(n) ) = O( \max( f(n), g(n) ) )$
- Example:
  - $f(n) = n^3 + 3n^2 + 15n$
  - $O( f(n) ) = O( \max( n^3, 3n^2, 15n ) ) = O( n^3 )$
  - why??

© 2005-06. 324-1-36

## More about $O()$

- No need to specify base of logarithm
  - $O(\log_a n) = O(\log_b n)$
  - reason:  $\log_a n = \log_a b * \log_b n$ 
    - $\log_a b$  is a positive constant
    - so the order is the same
- “is in  $O()$ ” is reflexive
- “is in  $O()$ ” is transitive

© 2005-06. 324-1-37

## Limit rule

- Theorem: if  $\lim f(n)/g(n)$  is positive real,  
then  $f$  in  $O(g)$  and  $g$  in  $O(f)$
- Theorem: if  $\lim f(n)/g(n) = 0$ , then  $f$  in  $O(g)$ 
  - (but not the other way)
- Theorem: if  $\lim f(n)/g(n) = \infty$ , then  $g$  in  $O(f)$ 
  - (but not the other way)

© 2005-06. 324-1-38

## Using the Limit Rule

- Which grows faster:  $\log n$  or  $\sqrt{n}$ ?
- Use limit rule and de l'Hôpital's Theorem
  - de l'Hôpital:
    - if:  $\lim f(n) = \lim g(n) = 0$   
or  $\lim f(n) = \lim g(n) = \infty$   
then:  $\lim f(n) / \lim g(n) = \lim f'(n) / \lim g'(n)$
- Combine with limit rule:
  - $\lim \log n / \sqrt{n} = \lim(1/n) / (1/(2\sqrt{n})) = \lim 2 / \sqrt{n} = 0$
  - hence:  $\log n$  is in  $O(\sqrt{n})$
  - and not the other way round.

© 2005-06. 324-1-39

## $\Omega$ and $\Theta$

- $O$  is upper bound
- $\Omega$  (omega) is lower bound
  - exactly the opposite!
- Example:
  - $n^3$  is in  $\Omega(n^2)$
  - $n^2$  is in  $\Omega(n)$
- Upper and lower bound:  $\Theta$  (theta)
  - $\Theta(f(n)) = \text{intersection of } O(f(n)) \text{ and } \Omega(f(n))$

© 2005-06. 324-1-40