Algorithm Analysis
part 2

cse2011

Good Algorithms! Good Data structures!

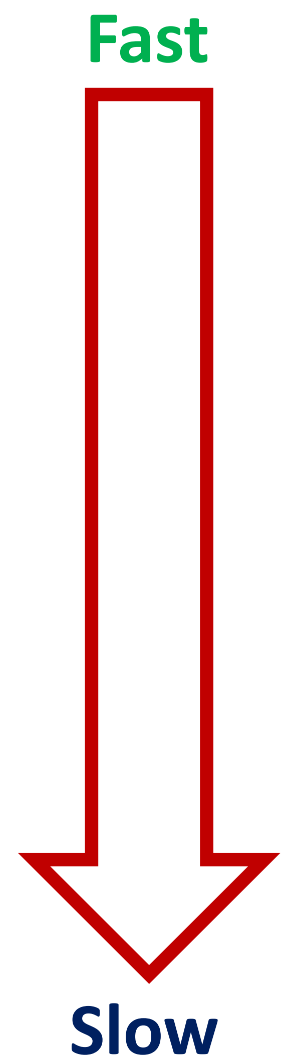
- **Data structure**: a systematic way of organizing and accessing data
- **Algorithm**: step by step procedure for performing a task in a finite amount of time.
- **Program**: **Data structure + Algorithm**
- Which algorithm/ds is better?
- We need precise ways to analyze them.

Methodology

- Assume that ***n*** is the input size (e.g., the size of the input array to be sorted) of the algorithm
- For each algorithm:
  - A function ***f(n)*** that characterize its running time is associated to it.
- We are generally interested in the **big-picture** approach.
  - **Constant factors** are generally disregarded.

Typical Functions ***f(n)***

1. Constant: 1
2. Logarithm: **log** *n* ($x = \log_b n \Leftrightarrow b^x = n$, b>1)
3. Linear: *n* (one ***for*** loop)
4. n-log-n: $n.\mathbf{log}\, n$
5. Quadratic: $n^{\mathbf{2}}$ (two ***for*** loops)
6. Cubic: $n^{\mathbf{3}}$ (three ***for*** loops)
7. Exponential: $\mathbf{a}^n$ (a is a constant)



Example

```
int findMax ( int [] A) {
      int max = A[0];
      for ( int i = 1; i < A.length; i++ )
            if(max < A[i])
                  max = A[i];
      return max;
}
```

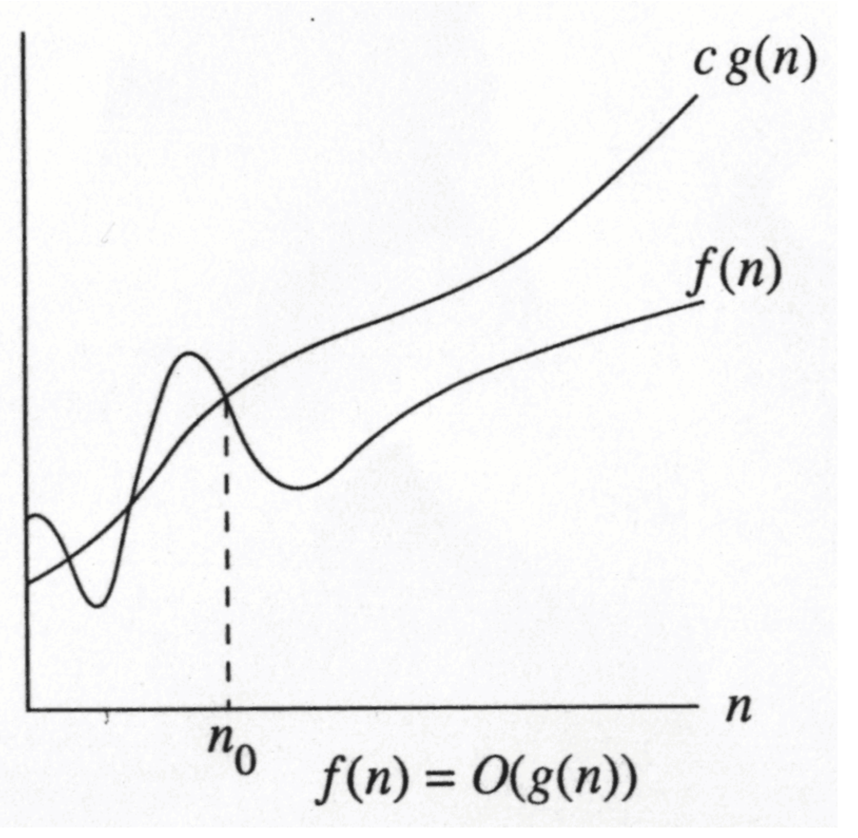
n : the **size** of array A (A.length)

We are just interested to see that findMax **grows proportionally** to ***n***

Constants are not important …

f(n) ≈ ???

Growth Rate



- The idea is to establish a relative order among functions for large n.
- $\exists$ c , $n_0 > 0$ such that $f(n) \leq c\, g(n)$ when $n \geq n_0$
- **f(N) grows no faster than g(N) for "large" N**

Asymptotic Notation: Big-Oh

- f(N) is O(g(N)) if
- There are positive constants c and n_0 such that

$$f(n) \leq c.g(n) \text{ when } n \geq n_0$$

  where c is a real number.

- The growth rate of f(N) is *less than or equal to* the growth rate of g(N).

- **g(N) is an upper bound for f(N).**

Big-Oh: Examples

- Let $f(N) = 2N^2$. Then
  - $f(N)$ is $O(N^4)$
  - $f(N)$ is $O(N^3)$
  - $f(N)$ is $O(N^2)$ (best answer, asymptotically **tight**)
  - The tighter the Big-Oh, the more accurate the estimate

- $O(N^2)$: reads "order N-squared" or "Big-Oh N-squared"

Example

- Show that $7N^2 + 10N + 5N\log N + 3$ is $O(N^2)$.

- Find c and n_0 such that when $N \geq n_0$
  $7N^2 + 10N + 5N\log N + 3 \leq cN^2$

- $7N^2 + 10N + 5N\log N + 3 \leq 7N^2 + 10N^2 + 5N^2 + 3N^2$
  $\leq 25N^2$ when $N \geq 1$
  So **$c = 25$** and **$n_0 = 1$**.

- Use the same "technique" for the following problems.

Big Oh: More Examples

- $(N^2 / 2) - 3N$ is $O(N^2)$
- $1 + 4N$ is $O(N)$
- $7N^2 + 10N + 3$ is $O(N^2)$, is also $O(N^3)$
- $\log_{10} N = \log_2 N / \log_2 10$ is $O(\log_2 N)$ or $O(\log N)$
- sin N is O(1); 10 is O(1), 10^{10} is O(1) [all of them are constants regardless of the size of the input]

- $\sum_{i=1}^{N} i \le N \cdot N = O(N^2)$

$$\sum_{i=1}^{N} i^2 \le N \cdot N^2 = O(N^3)$$

- $\log N + N$ is $O(N)$
- $\log^k N$ is $O(N)$ for any constant k
- N is $O(2^N)$, but 2^N is not $O(N)$
- 2^{4N} is not $O(2^N)$

Math Review: Logarithmic Functions

$$x^a = b \quad iff \quad \log_x b = a$$

$$\log ab = \log a + \log b$$

$$\log_a b = \frac{\log_m b}{\log_m a}$$

$$\log a^b = b \log a$$

$$a^{\log n} = n^{\log a}$$

$$\log^b a = (\log a)^b \neq \log a^b$$

$$\frac{d \log_e x}{dx} = \frac{1}{x}$$

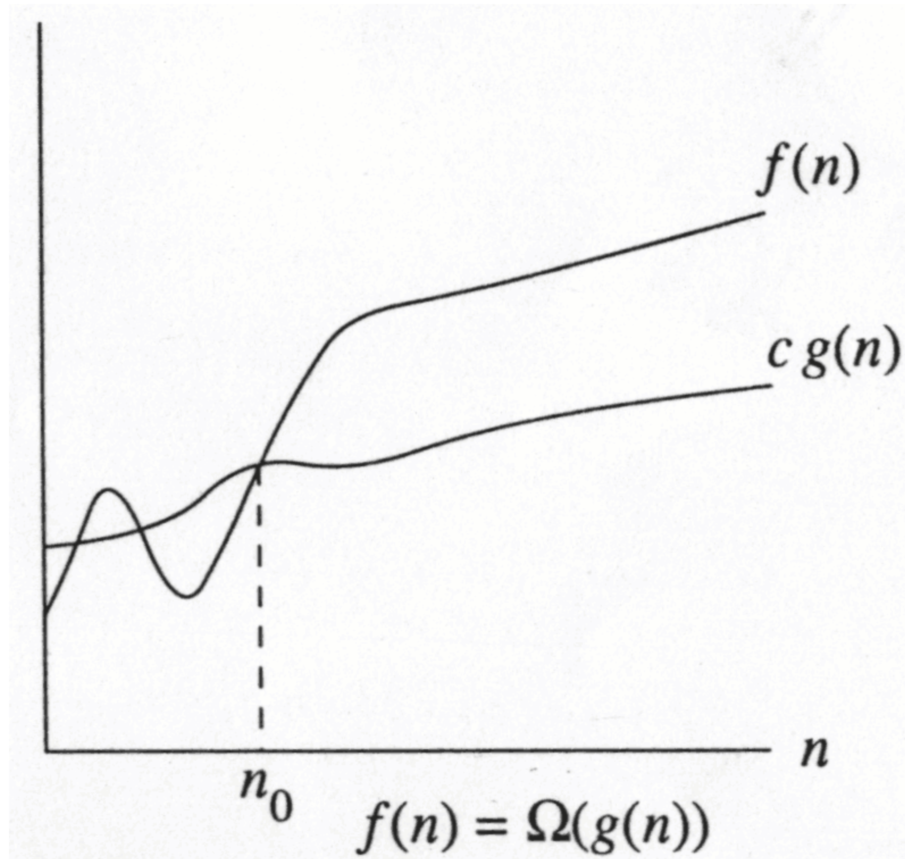
Some Rules

When considering the growth rate of a function using O()

- **Ignore** the lower order terms and the coefficients of the highest-order term
- **No need** to specify the base of logarithm
  - Changing the base from one constant to another changes the value of the logarithm by only a constant factor

- If $T_1(N)$ is O(f(N)) and $T_2(N)$ is O(g(N)), then
  - $T_1(N) + T_2(N)$ is O(f(N) + g(N)) *[if/else statements]*
    (or less formally it is max (O(f(N)), O(g(N)))),
  - $T_1(N) * T_2(N)$ is O(f(N) * g(N)) *[nested for loops]*

Big-Omega



- $\exists$ c , $n_0 > 0$ such that $f(N) \geq c\, g(N)$ when $N \geq n_0$
- f(N) grows no slower than g(N) for “large” N

Big-Omega

- f(N) is $\Omega(g(N))$ if
- There are positive constants c and n_0 such that

  $$f(N) \geq c\, g(N) \text{ when } N \geq n_0$$

  where c is a real number.

- The growth rate of f(N) is *greater than or equal to* the growth rate of g(N).

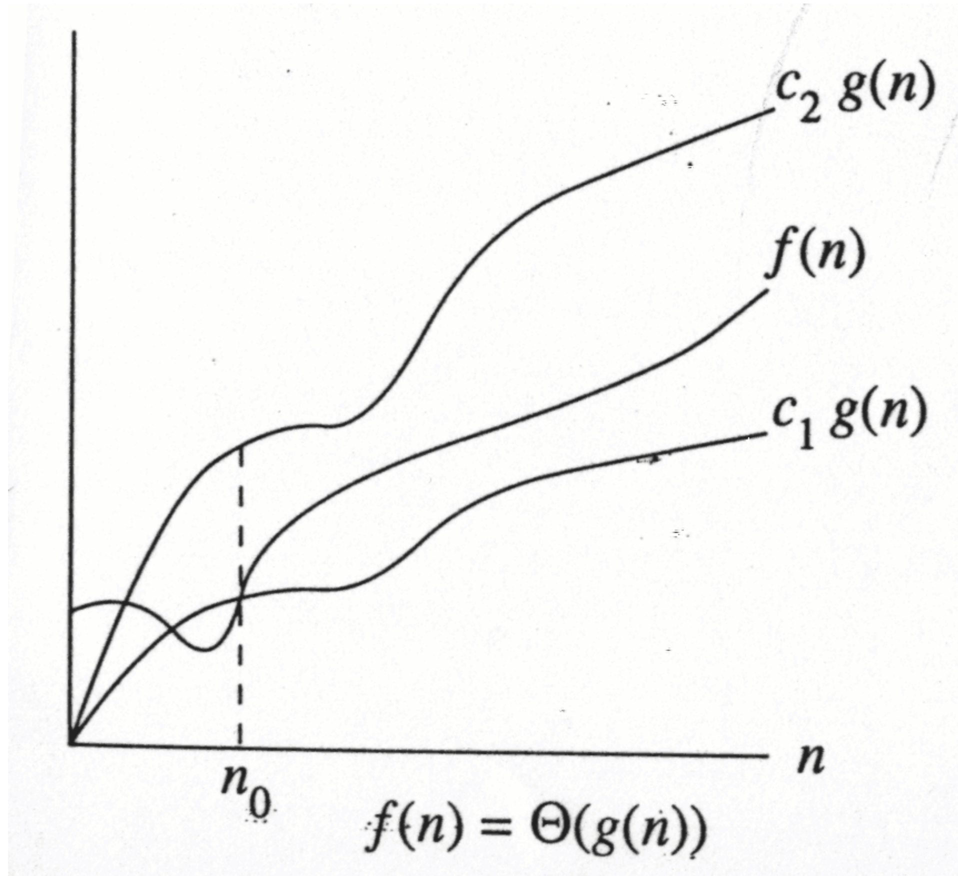
- g(N) is a **lower bound** on f(N).

Big-Omega: Examples

- Let $f(N) = 2N^2$. Then
  - $f(N)$ is $\Omega(N)$ (not tight)
  - $f(N)$ is $\Omega(N^2)$ (best answer)

  - We are still interested in the **tightest** function!

Big-Theta



- The growth rate of f(N) *is the same as* the growth rate of g(N)
- f(N) is Θ(g(N)) iff f(N) is O(g(N)) and f(N) is Ω(g(N))

Big-Theta: Example

- Let $f(N) = N^2$, $g(N) = 2N^2$
  - Since $f(N)$ is $O(g(N))$ and $f(N)$ is $\Omega(g(N))$,
    $f(N) = \Theta(g(N))$.

- $c_1 = 1$, $n_1 = 0$
- $c_2 = ½$, $n_2 = 0$

An Example of O, Ω and Θ

- Given an unsorted array of integers, return true if a number k is in the array and false otherwise.

```
for( i = 0; i < N; i++ )
   if ( k == A[i] )
      return ( true );
return ( false );
```

- **Worst-case running time is O(N).** **upper bound**
  ⇒The alg. has O(N) running time.
- **Best-case running time is O(1).** **lower bound**
  ⇒ The alg. has Ω(1) running time.

⇒We do not have Θ running time for this *algorithm*.

An Example of O, Ω and Θ

- Given an unsorted array of integers, find and return the maximum value stored in the array.

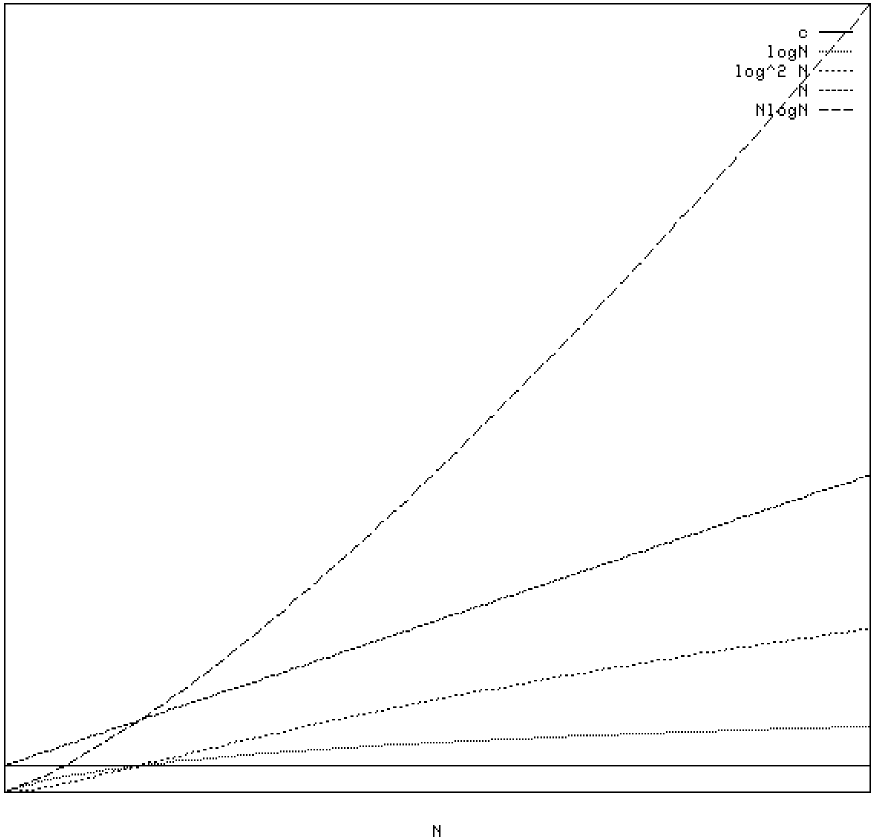
```
max = A[0];
for( i = 1; i < N; i++ )
   if ( max < A[i] )
      max = A[i];
return( max );
```

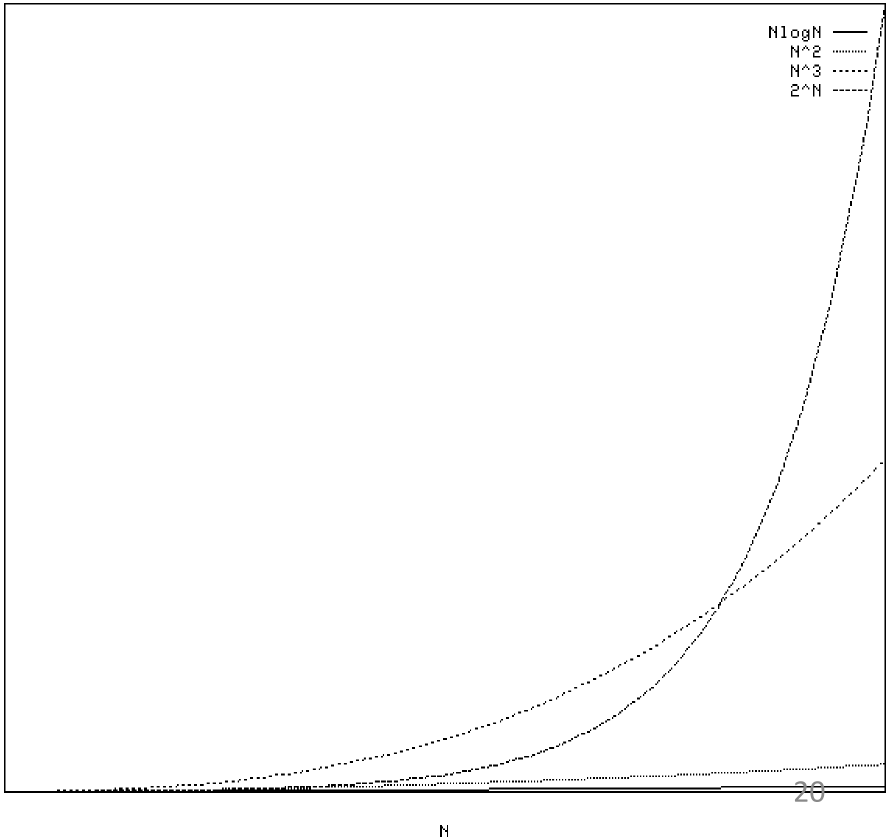
- **Worst-case running time is O(N).** **upper bound**
  ⇒The alg. has O(N) running time.
- **Best-case running time is O(N).** **lower bound**
  ⇒The alg. has Ω(N) running time.

- **⇒The algorithm has Θ(N) running time.**

Typical Growth Rates

Function	Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential





Some More Rules

- If T(N) is a polynomial of degree *k*, then
  T(N) is $\Theta(N^{k})$.

- For logarithmic functions,
  $T(\log_{m} N)$ is $\Theta(\log N)$.

- $\log^{k} N$ is O(N) for any constant *k*
  (logarithms grow very slowly)

Small-oh

- f(N) is ***o***(g(N)) if

- f(N) is ***o***(g(N))
  if f(N) is ***O***(g(N)) and f(N) is not **Θ**(g(N))

- g(N) grows faster than f(N) for “large” N.

Small-oh: Example

- Let $f(N) = \frac{3}{4} N^2$ and f(N) be ***o***(g(N)).
  - $g(N) = N^2$?
  - $g(N) = N^2 \log N$?
  - $g(N) = N^3$?

Determining Relative Growth Rates of Two Functions

1. Using simple algebra (discussed previously)

   Example: which function grows faster?

   – $f(N) = N \log N$

   – $g(N) = N^{1.5}$

2. Using L' Hôpital' s rule

Using L' Hôpital's Rule

- L' Hôpital's rule

  If $\lim_{n\to\infty} f(N) = \infty$ and $\lim_{n\to\infty} g(N) = \infty$

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

- Determine the relative growth rates: compute $\lim_{n\to\infty} \frac{f(N)}{g(N)}$

  – if 0: f(N) is ***o***(g(N))
  – if constant $\neq$ 0: f(N) is Θ(g(N))
  – if ∞: g(N) is ***o***(f(N))
  – limit oscillates: no relation

Summary of Chapter 4

- Given an algorithm, compute its running time in terms of O, Ω, and Θ (if any).
  - Usually the big-Oh running time is enough.
- Given f(n) = 5n + 10, show that f(n) is O(n).
  - Find c and n_0
- Compare the grow rates of 2 functions.
- Order the grow rates of several functions.
  - Use simple algebra.
  - Use L'Hôpital's rule.

Next time …

- Recursion (Chapter 3)
- Reading for this lecture: Chapter 4