Syrian Private University

Algorithms & Data Structure I

Instructor: Dr. Mouhib Alnoukari

Asymptotic Analysis

التحليل التقاربي

Asymptotic Analysis

Outline:

we will look at:

- Justification for analysis
- Quadratic and polynomial growth
- Landau symbols
- Big-Θ as an equivalence relation
- Little-o as a weak ordering

Background

Suppose we have two algorithms, how can we tell which is better?

We could implement both algorithms, run them both

- Expensive and error prone

Preferably, we should analyze them mathematically

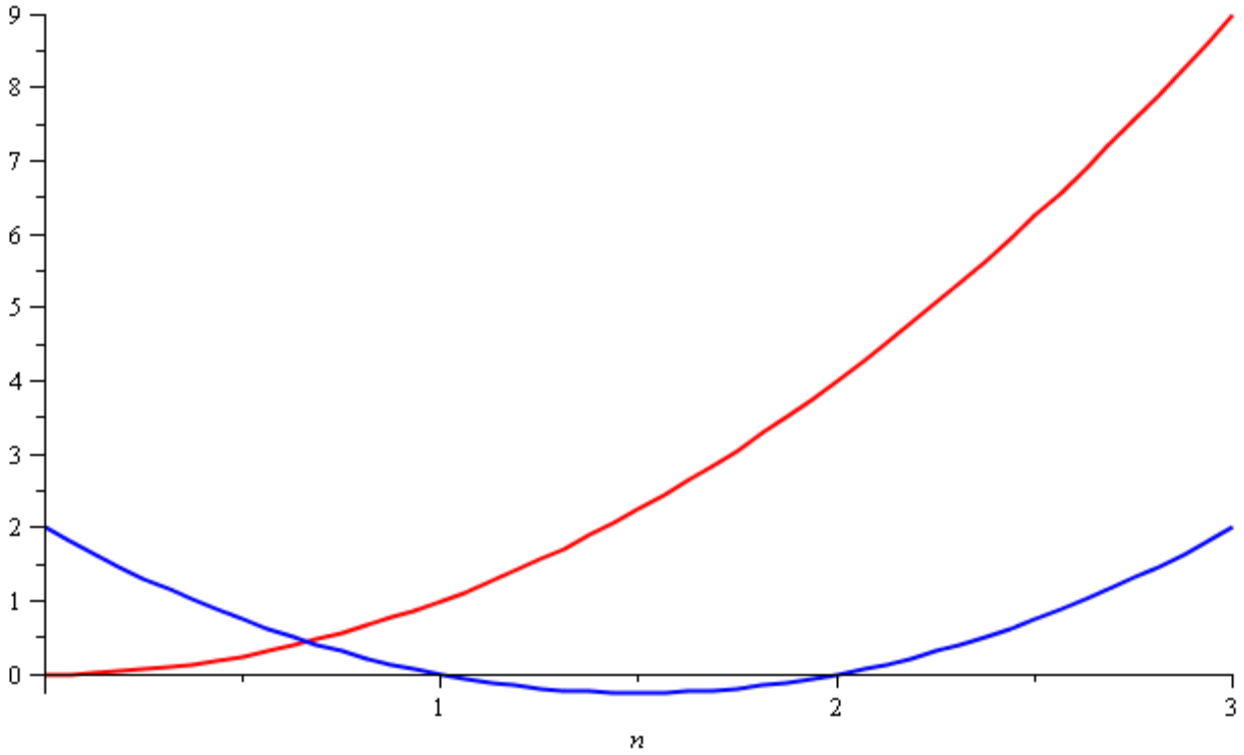
- *Algorithm analysis*

Quadratic Growth

Consider the two functions

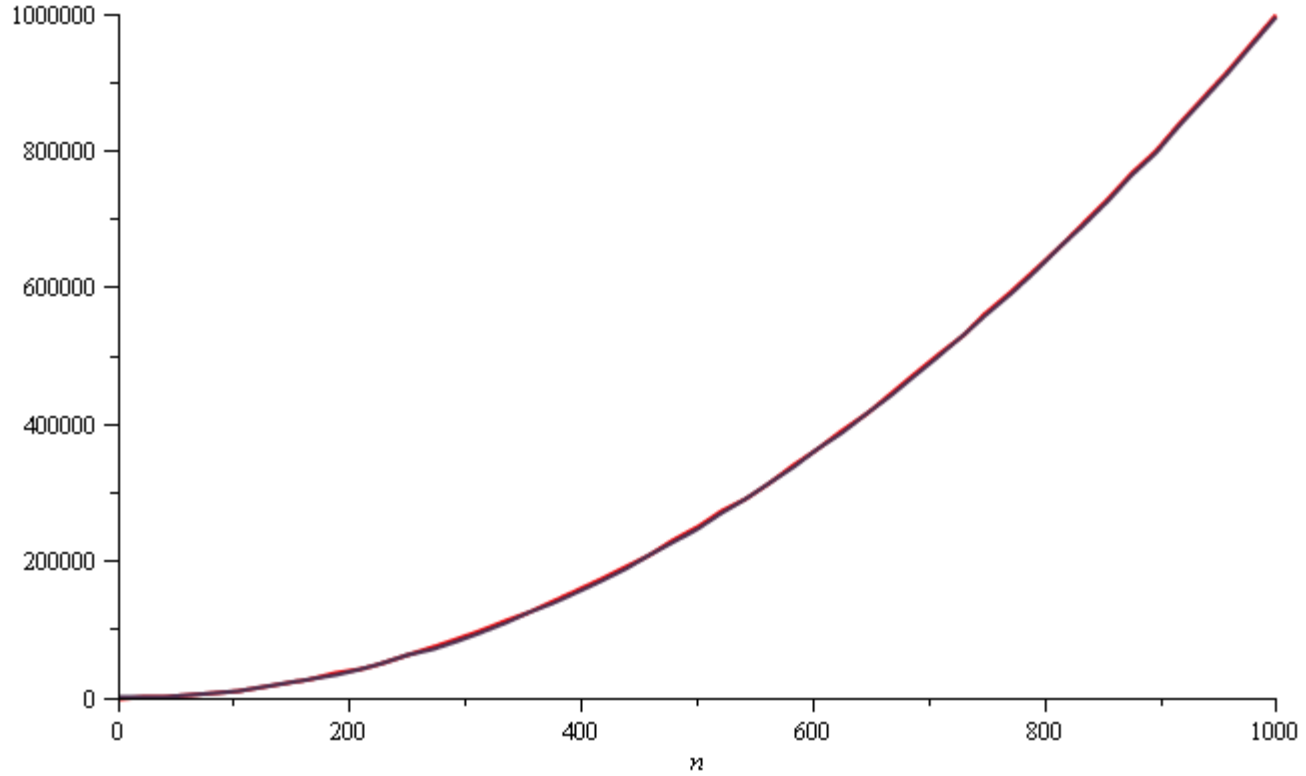
$f(n) = n^2$ and $g(n) = n^2 - 3n + 2$

Around $n = 0$, they look very different



Quadratic Growth

Yet on the range $n = [0, 1000]$, they are (relatively) indistinguishable:



Quadratic Growth

The absolute difference is large, for example,

$$f(1000) = 1\,000\,000$$

$$g(1000) = \quad 997\,002$$

but the relative difference is very small

$$\left|\frac{f(1000) - g(1000)}{f(1000)}\right| = 0.002998 < 0.3\%$$

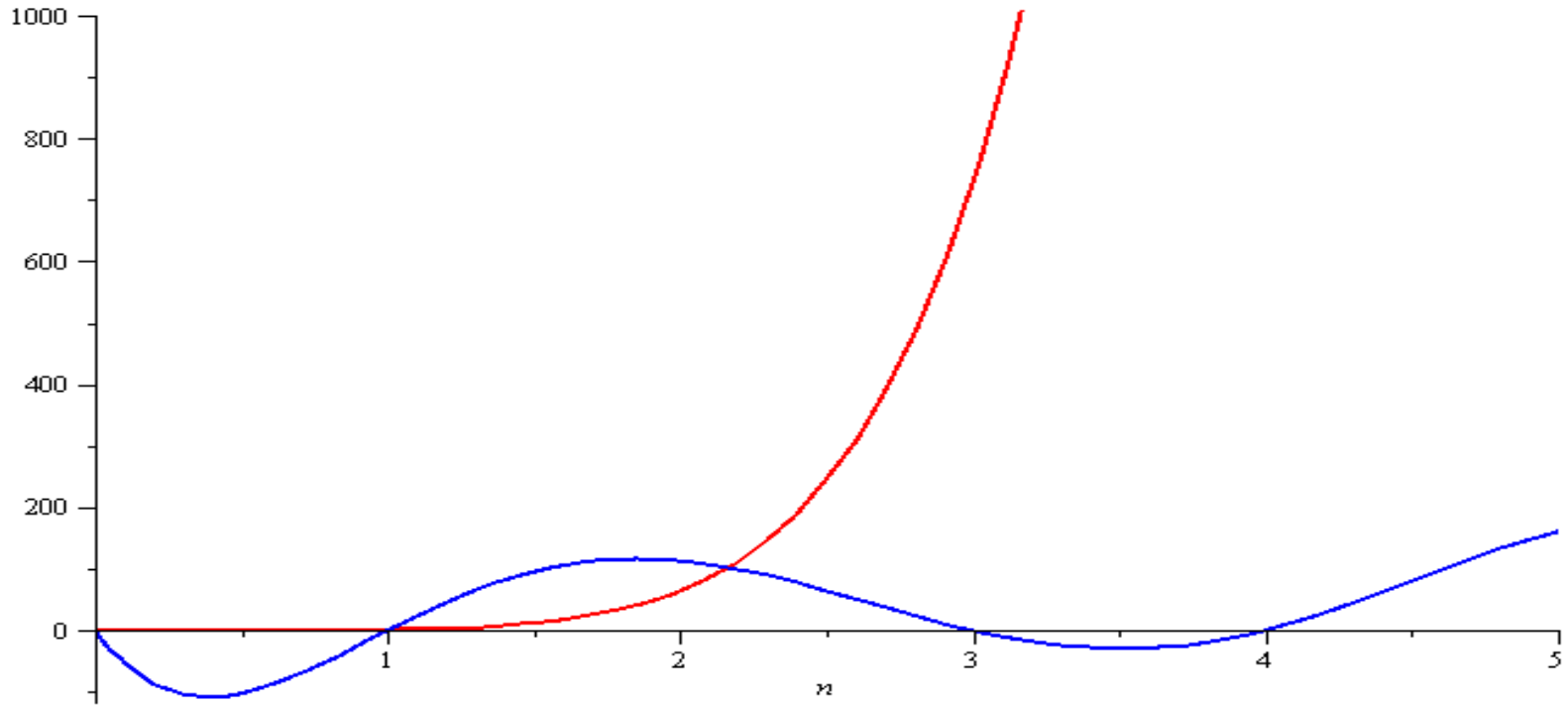
and this difference goes to zero as $n \to \infty$

Polynomial Growth

To demonstrate with another example,

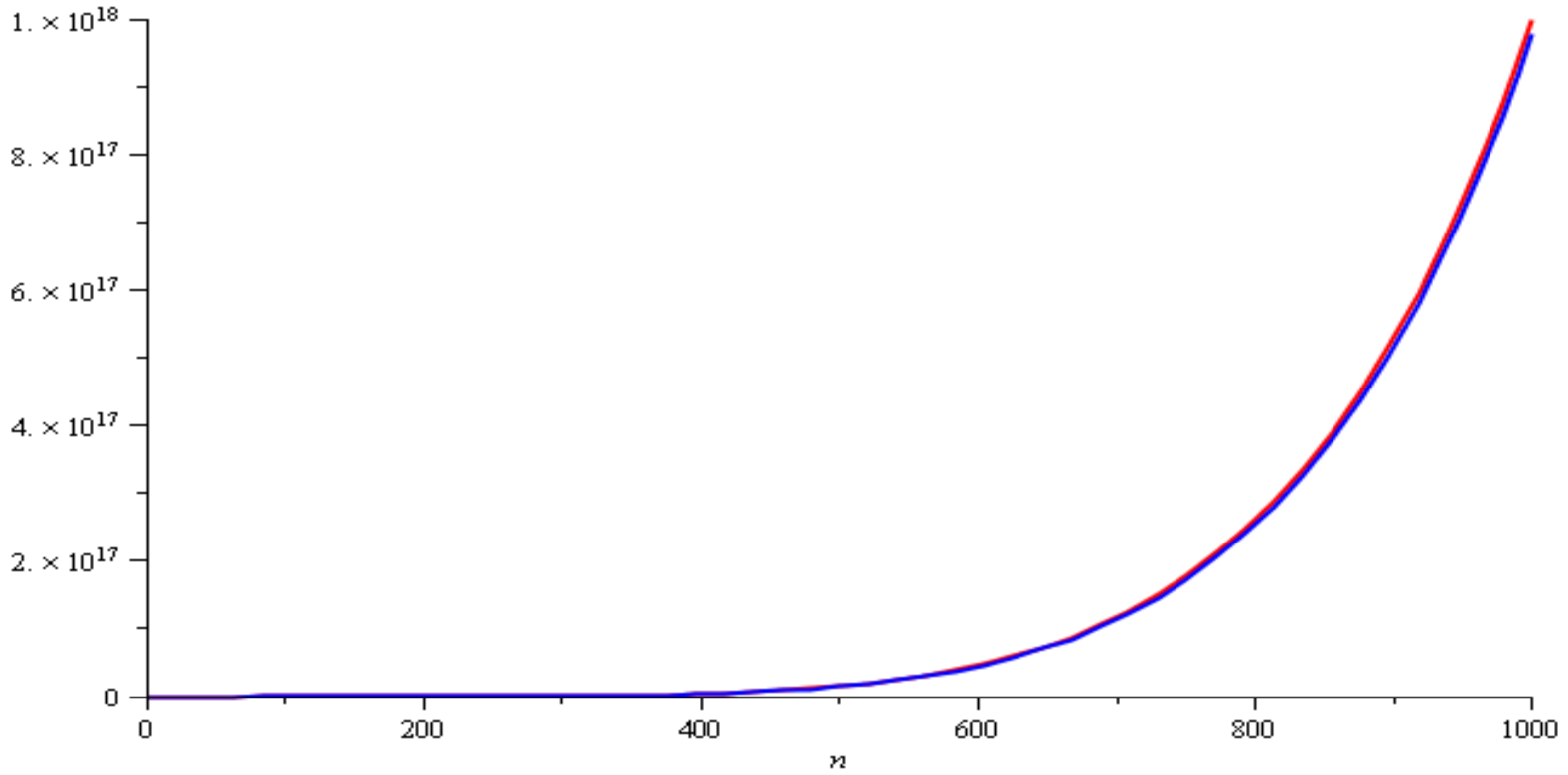
$f(n) = n^6$ and $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$

Around $n = 0$, they are very different



Polynomial Growth

Still, around $n = 1000$, the relative difference is less than 3%



Polynomial Growth

The justification for both pairs of polynomials being similar is that, in both cases, they each had the same leading term:

n^2 in the first case, n^6 in the second

Suppose however, that the coefficients of the leading terms were different

– In this case, both functions would exhibit the same rate of growth, however, one would always be proportionally larger

Asymptotic Analysis

- Goal: to simplify analysis of running time by getting rid of ”details”, which may be affected by specific implementation and hardware
  - like “rounding”: $1{,}000{,}001 \approx 1{,}000{,}000$
  - $3n^2 \approx n^2$
- Capturing the essence: how the running time of an algorithm increases with the size of the input *in the limit*.
  - Asymptotically more efficient algorithms are best for all but small inputs

Asymptotic Analysis

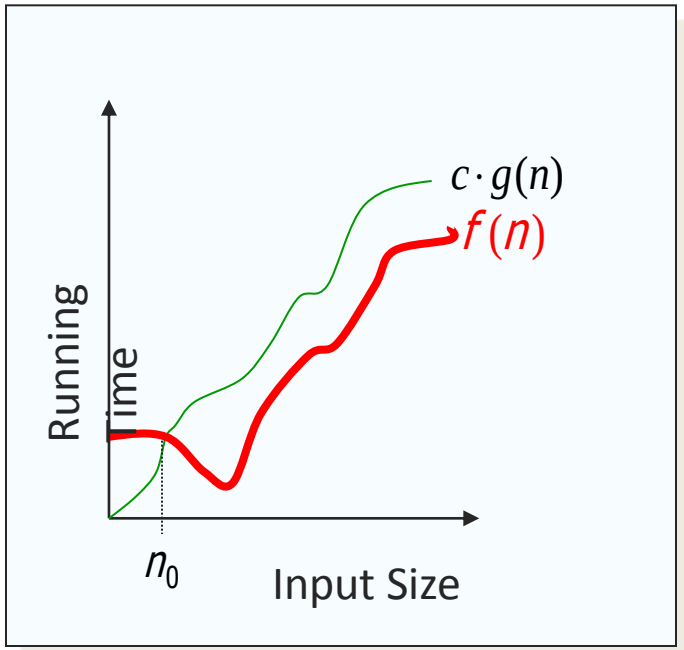
Given an algorithm:

- We need to be able to describe these values mathematically
- We need a systematic means of using the description of the algorithm together with the properties of an associated data structure
- We need to do this in a machine-independent way

For this, we need Landau symbols and the associated asymptotic analysis

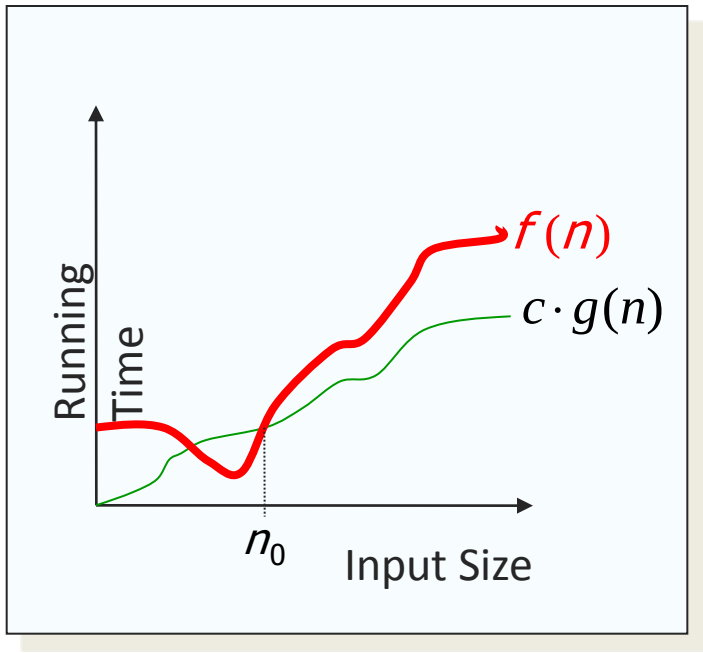
Asymptotic Notation - Landau Symbols

- The "big-Oh" *O*-Notation
  - asymptotic upper bound
  - *f(n) = O(g(n)),* if there exists constants *c* and n_0, *s.t.* **f(n) ≤ c g(n)** for $n \geq n_0$
  - *f(n)* and *g(n)* are functions over non-negative integers
- Used for *worst-case* analysis
- The function $\mathrm{f}(n)$ has a rate of growth no greater than that of $\mathrm{g}(n)$



Asymptotic Notation (2)

- The "big-Omega" Ω–Notation
  - asymptotic lower bound
  - $f(n) = \Omega(g(n))$ if there exists constants c and n_0, *s.t.* $\mathbf{c\ g(n) \leq f(n)}$ for $n \geq n_0$
- Used to describe *best-case* running times or lower bounds of algorithmic problems
  - E.g., lower-bound of searching in an unsorted array is $\Omega(n)$.

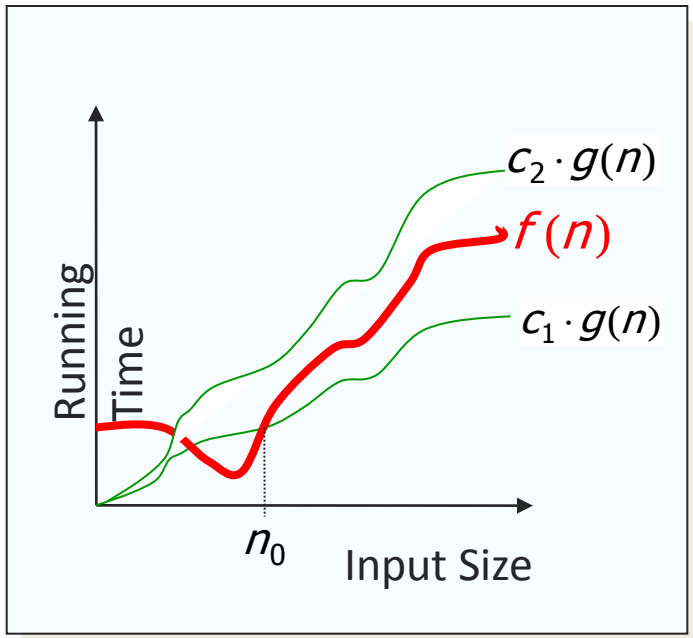


Asymptotic Notation (3)

- Simple Rule: Drop lower order terms and constant factors.
  - 50 n log n is O(n log n)
  - $7n - 3$ is O(n)
  - $8n^2 \log n + 5n^2 + n$ is O(n^2 log n)
- Note: Even though (50 n log n) is O(n^5), it is expected that such an approximation be of as small an order as possible

Asymptotic Notation (4)

- The "big-Theta" Θ–Notation
  - asymptoticly tight bound
  - $f(n) = \Theta(g(n))$ if there exists constants c_1, c_2, and n_0, *s.t.* $\mathbf{c_1\ g(n) \leq f(n) \leq c_2\ g(n)}$ for $n \geq n_0$
- $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- $O(f(n))$ is often misused instead of $\Theta(f(n))$
- The function $f(n)$ has a rate of growth equal to that of $g(n)$



Θ–Notation: Exercise

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

We must determine positive constants c_1, c_2, and n_0:

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

for all $n \ge n_0$. Dividing by n^2 yields

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2 .$$

By choosing c_1=1/14, c_2=1/2 and n_0=7.

$$6n^3 \neq \Theta(n^2)$$

Suppose for the purpose of contradiction that c_2, and n_0 exists such that: $6n^3 \le c_2 n^2$ for all $n \ge n_0$. Dividing by n^2 yields: $n \le c_2/6$

Which cannot possibly hold for arbitrarily large n, since c_2 is constant.

Asymptotic Notation (5)

- Two more asymptotic notations
  - "Little-Oh" notation $f(n)=o(g(n))$
    non-tight analogue of Big-Oh
    - For every c, there should exist n_0, s.t. $\mathbf{f(n) \leq c\, g(n)}$ for $n \geq n_0$
    - Used for **comparisons** of running times. If $f(n)=o(g(n))$, it is said that $g(n)$ *dominates* $f(n)$.

  "Big-Oh" : For **some** c, there should exist n_0, s.t. $\mathbf{f(n) \leq c\, g(n)}$ for $n \geq n_0$

  - "Little-omega" notation $f(n)=\omega(g(n))$
    non-tight analogue of Big-Omega

Asymptotic Notation (6)

- Analogy with real numbers
  - $f(n) = O(g(n)) \cong f \leq g$
  - $f(n) = \Omega(g(n)) \cong f \geq g$
  - $f(n) = \Theta(g(n)) \cong f = g$
  - $f(n) = o(g(n)) \cong f < g$
  - $f(n) = \omega(g(n)) \cong f > g$
- Abuse of notation: $f(n) = O(g(n))$ actually means $f(n) \in O(g(n))$

Landau Symbols

Graphically, we can summarize these as follows:

We say

$$f(n) =$$

$\mathbf{O}(g(n))$ $\Omega(g(n))$

$\mathbf{o}(g(n))$ $\Theta(g(n))$ $\omega(g(n))$

if

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

0 $0 < c < \infty$ ∞

Landau Symbols

Some other observations we can make are:

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$
$$f(n) = \mathbf{O}(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$
$$f(n) = \mathbf{o}(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

Big-Θ as an Equivalence Relation

If we look at the first relationship, we notice that $f(n) = \Theta(g(n))$ seems to describe an equivalence relation:

1. $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
2. $f(n) = \Theta(f(n))$
3. If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, it follows that $f(n) = \Theta(h(n))$

Consequently, we can group all functions into equivalence classes, where all functions within one class are big-theta Θ of each other

Big-Θ as an Equivalence Relation

For example, all of

n^2 $\quad 100000\,n^2 - 4\,n + 19$ $\quad n^2 + 1000000$

$323\,n^2 - 4\,n\ln(n) + 43\,n + 10$ $\quad 42n^2 + 32$

$n^2 + 61\,n\ln^2(n) + 7n + 14\ln^3(n) + \ln(n)$

are big-Θ of each other

E.g., $42n^2 + 32 = \Theta(\,323\,n^2 - 4\,n\ln(n) + 43\,n + 10\,)$

Big-Θ as an Equivalence Relation

The most common classes are given names:

$\Theta(1)$	constant
$\Theta(\ln(n))$	logarithmic
$\Theta(n)$	linear
$\Theta(n \ln(n))$	"n log n"
$\Theta(n^2)$	quadratic
$\Theta(n^3)$	cubic
2^n, e^n, 4^n, ...	exponential

Logarithms and Exponentials

Recall that all logarithms are scalar multiples of each other

- Therefore $\log_b(n) = \Theta(\ln(n))$ for any base b

Alternatively, there is no single equivalence class for exponential functions:

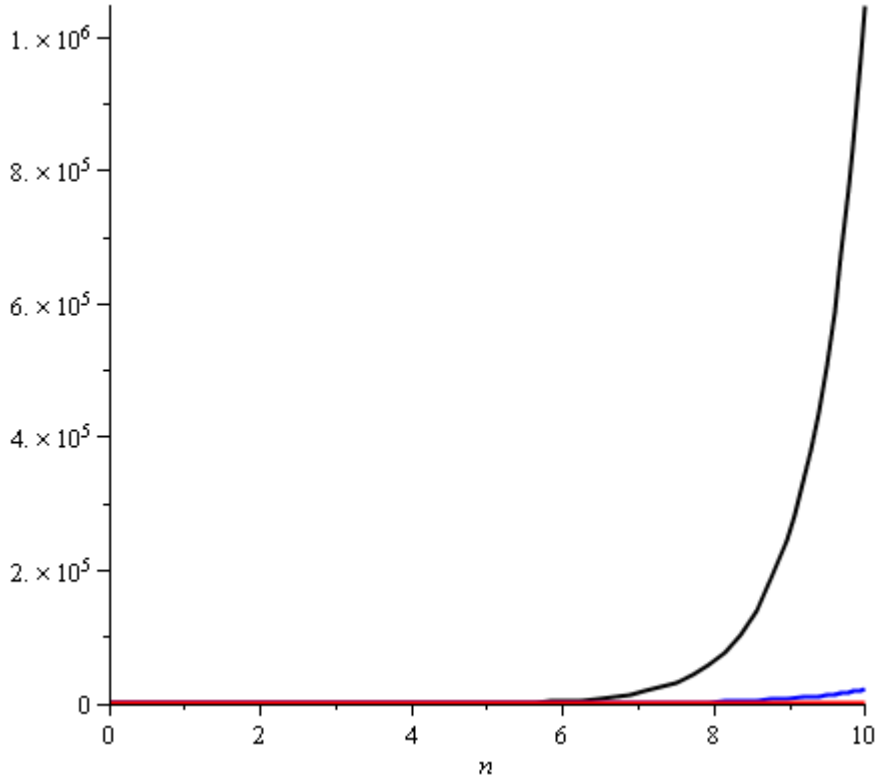
$$\lim_{n\to\infty} \frac{a^n}{b^n} = \lim_{n\to\infty} \left(\frac{a}{b}\right)^n = 0$$

- If $1 < a < b$,
- Therefore $a^n = \mathbf{o}(b^n)$

However, we will see that it is almost universally undesirable to have an exponentially growing function!

Logarithms and Exponentials

Plotting 2^n, e^n, and 4^n on the range $[1, 10]$ already shows how significantly different the functions g



Note:

$2^{10} = 1024$

$e^{10} \approx 22\,026$

$4^{10} = 1\,048\,576$

Little-o as a Weak Ordering

We can show that, for example

$$\ln(n) = \mathbf{o}(n^p)$$

for any $p > 0$

Proof: Using l'Hôpital's rule, we have

$$\lim_{n\to\infty} \frac{\ln(n)}{n^p} = \lim_{n\to\infty} \frac{1/n}{pn^{p-1}} = \lim_{n\to\infty} \frac{1}{pn^p} = \frac{1}{p} \lim_{n\to\infty} n^{-p} = 0$$

Conversely, $1 = \mathbf{o}(\ln(n))$

Little-o as a Weak Ordering

Other observations:

- If p and q are real positive numbers where $p < q$, it follows that

$$n^p = \mathbf{o}(n^q)$$

- For example, matrix-matrix multiplication is $\Theta(n^3)$ but a refined algorithm is $\Theta(n^{\lg(7)})$ where $\lg(7) \approx 2.81$

- Also, $n^p = \mathbf{o}(\ln(n)n^p)$, but $\ln(n)n^p = \mathbf{o}(n^q)$
  - n^p has a slower rate of growth than $\ln(n)n^p$, but
  - $\ln(n)n^p$ has a slower rate of growth than n^q for $p < q$

Little-o as a Weak Ordering

If we restrict ourselves to functions $f(n)$ which are $\Theta(n^p)$ and $\Theta(\ln(n)n^p)$, we note:

- It is never true that $f(n) = \mathbf{o}(f(n))$
- If $f(n) \neq \Theta(g(n))$, it follows that either

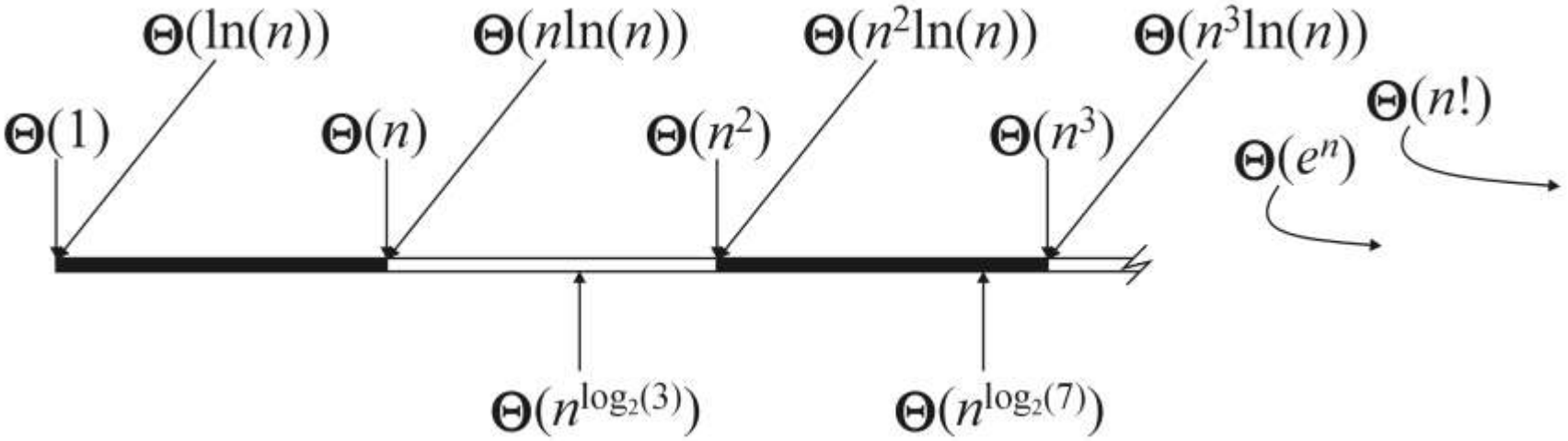
$$f(n) = \mathbf{o}(g(n)) \text{ or } g(n) = \mathbf{o}(f(n))$$

- If $f(n) = \mathbf{o}(g(n))$ and $g(n) = \mathbf{o}(h(n))$, it follows that $f(n) = \mathbf{o}(h(n))$

This defines a weak ordering!

Little-o as a Weak Ordering

Graphically, we can shown this relationship by marking these against the real line



Algorithms Analysis

We will use Landau symbols to describe the complexity of algorithms

- E.g., adding a list of n doubles will be said to be a $\Theta(n)$ algorithm

An algorithm is said to have *polynomial time complexity* if its run-time may be described by $O(n^d)$ for some fixed $d \geq 0$

- We will consider such algorithms to be *efficient*

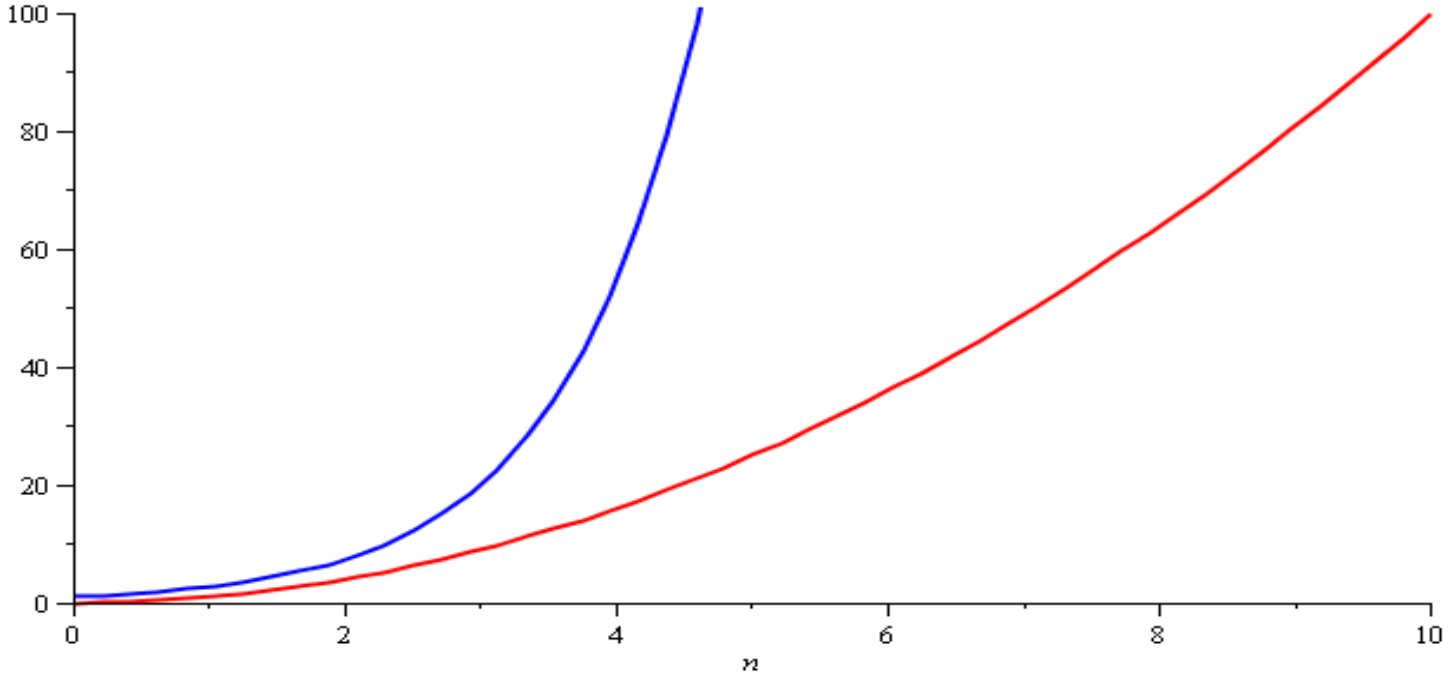
Problems that have no known polynomial-time algorithms are said to be *intractable*

- Traveling salesman problem: find the shortest path that visits n cities
- Best run time: $\Theta(n^2 2^n)$

Algorithm Analysis

In general, you don't want to implement exponential-time or exponential-memory algorithms

- Warning: don't call a quadratic curve "exponential", either...please



Exercises

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		

Exercises

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n \log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$

Exercises

Statement	Is it TRUE or FALSE?	If it is FALSE then write the correct formula
Rule of sums: $O(f+g) = O(f) + O(g)$		
Rule of products: $O(f \cdot g) = O(f) \cdot O(g)$		
Transitivity: if $g = O(f)$ and $h = O(f)$ then $g = O(h)$		

Exercises

Statement	Is it TRUE or FALSE?	If it is FALSE then write the correct formula
Rule of sums: $O(f+g) = O(f) + O(g)$	FALSE	$O(f+g) = \max\{O(f), O(g)\}$
Rule of products: $O(f \cdot g) = O(f) \cdot O(g)$	TRUE	
Transitivity: if $g = O(f)$ and $h = O(f)$ then $g = O(h)$	FALSE	if $g = O(f)$ and $f = O(h)$ then $g = O(h)$

Exercises

Running time $T(n)$ of processing n data items with a given algorithm is described by the recurrence:

$$T(n) = k \cdot T\left(\frac{n}{k}\right) + c \cdot n; \;\; T(1) = 0.$$

Derive a closed form formula for $T(n)$ in terms of c, n, and k. What is the computational complexity of this algorithm in a "Big-Oh" sense? *Hint*: To have the well-defined recurrence, assume that $n = k^m$ with the integer $m = \log_k n$ and k.

Exercises

$$
\begin{aligned}
T(k^m) &= k \cdot T(k^{m-1}) + c \cdot k^m \\
k \cdot T(k^{m-1}) &= k^2 \cdot T(k^{m-2}) + c \cdot k^m \\
&\dots \quad \dots \quad \dots \\
k^{m-1} \cdot T(k) &= k^m \cdot T(1) + c \cdot k^m \quad \text{so that} \quad T(k^m) = c \cdot m \cdot k^m
\end{aligned}
$$

$T(n) = c \cdot n \cdot \log_k n.$

$O(n \log n).$

Exercises

1- Let processing time of an algorithm of Big-Oh complexity O(f(n)) be directly proportional to f(n). Let three such algorithms A, B, and C have time complexity O(n2), O(n1.5), and O(n log n), respectively. During a test, each algorithm spends 10 seconds to process 100 data items. Derive the time each algorithm should spend to process 10,000 items.

2- Software packages A and B have processing time exactly $T_{EP} = 3n^{1.5}$ and $T_{WP} = 0.03n^{1.75}$, respectively. If you are interested in faster processing of up to $n = 10^8$ data items, then which package should be choose?

Exercises

1- Let processing time of an algorithm of Big-Oh complexity O(f(n)) be directly proportional to f(n). Let three such algorithms A, B, and C have time complexity O(n2), O(n1.5), and O(n log n), respectively. During a test, each algorithm spends 10 seconds to process 100 data items. Derive the time each algorithm should spend to process 10,000 items.

	Complexity	Time to process 10,000 items
A1	$O(n^2)$	$T(10,000) = T(100) \cdot \frac{10000^2}{100^2} = 10 \cdot 10000 = 100,000$ sec.
A2	$O(n^{1.5})$	$T(10,000) = T(100) \cdot \frac{10000^{1.5}}{100^{1.5}} = 10 \cdot 1000 = 10,000$ sec.
A3	$O(n \log n)$	$T(10,000) = T(100) \cdot \frac{10000 \log 10000}{100 \log 100} = 10 \cdot 200 = 2,000$ sec.

Exercises

2- Software packages A and B have processing time exactly $T_{EP} = 3n^{1.5}$ and $T_{WP} = 0.03n^{1.75}$, respectively. If you are interested in faster processing of up to $n = 10^8$ data items, then which package should be choose?

In the Big-Oh sense, the package **A** is better. But it outperforms the package **B** when $T_A(n) \leq T_B(n)$, that is, when $3n^{1.5} \leq 0.03n^{1.75}$. This inequality reduces to $n^{0.25} \geq 3/0.03(= 100)$, or $n \geq 10^8$. Thus for processing up to 10^8 data items, the package of choice is **B**.