Algorithms – Overview (1A)

Young Won Lim
3/29/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Informal Definitions of Algorithms

An informal definition could be
"a set of <u>rules</u> that precisely defines a <u>sequence</u> of <u>operations</u>."
which would include all computer programs,
including programs that do not perform numeric calculations.
Generally, a program is only an algorithm if it stops eventually.

A prototypical example of an algorithm is
the Euclidean algorithm to determine
the maximum common divisor of two integers;

https://en.wikipedia.org/wiki/Algorithm

Flow Chart – Euclid Algorithms

flow chart of an algorithm (Euclid's algorithm)
for calculating the greatest common divisor (gcd)

two numbers **a** and **b** in locations named **A** and **B**.

<u>successive</u> <u>subtractions</u> in two loops:

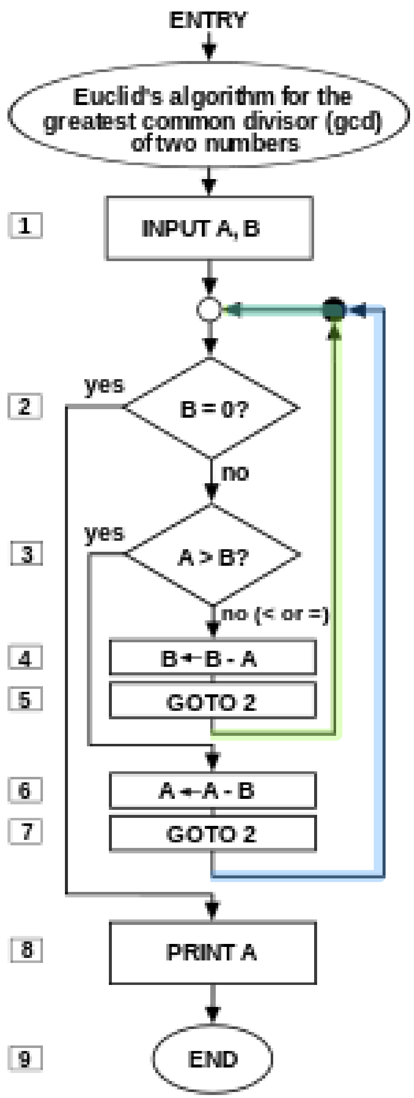
IF the test **B** ≥ **A**, THEN **B** ← **B − A**

Similarly, IF **A** > **B**, THEN **A** ← **A − B**.

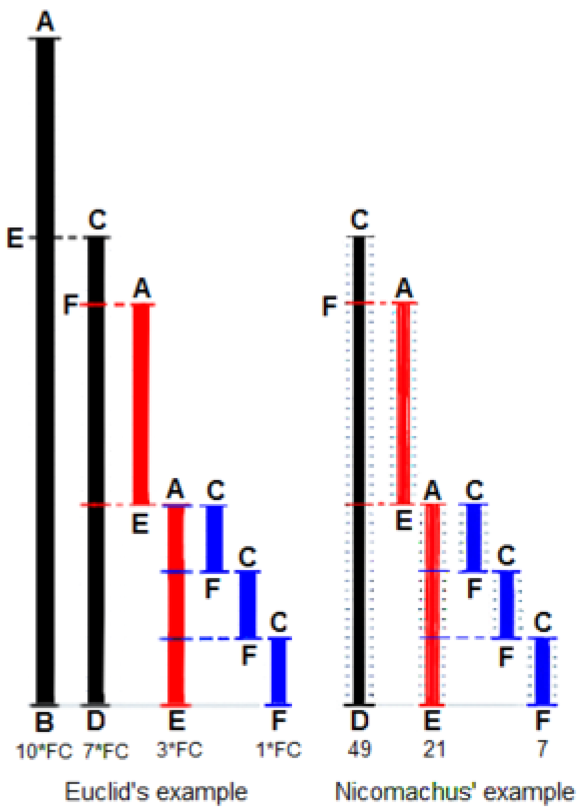
terminates when (the contents of) **B** is 0,
yielding the g.c.d. in **A.**

(Algorithm derived from Scott 2009:13;
symbols and drawing style from Tausworthe 1977).

https://en.wikipedia.org/wiki/Algorithm



Euclid Algorithm



Euclid's method for finding the greatest common divisor (GCD) of two starting lengths BA and DC, both defined to be multiples of a common "unit" length.

The length DC being shorter, it is used to "measure" BA, but only once because remainder EA is less than DC. EA now measures (twice) the shorter length DC, with remainder FC shorter than EA. Then FC measures (three times) length EA. Because there is no remainder, the process ends with FC being the GCD. On the right Nicomachus' example with numbers 49 and 21 resulting in their GCD of 7 (derived from Heath 1908:300).

https://en.wikipedia.org/wiki/Euclidean_algorithm

Best, Worst, Average Case Complexities

The best, worst and average case complexity refer to
three different ways of measuring the **time complexity**
(or any other complexity measure) of different inputs of the same size.

Since some inputs of the same size **n** may be faster to solve than others.

This complexity is only defined with respect to
a probability distribution over the inputs.

For instance, if all inputs of the same size
are assumed to be equally likely to appear,

the average case complexity can be defined
with respect to the uniform distribution over all inputs of size n.

https://en.wikipedia.org/wiki/Algorithm

Best, Worst, Average Case Complexities

Best-case complexity:
the complexity of solving the problem for the best input of size n.

Worst-case complexity:
the complexity of solving the problem for the worst input of size n.

Average-case complexity:
the complexity of solving the problem on an average.

https://en.wikipedia.org/wiki/Algorithm

Upper and Lower Bounds

To classify the **computation time**
(or similar resources, such as **space consumption**),
one is interested in proving upper and lower bounds
on the minimum amount of time required
by the most efficient algorithm solving a given problem.

The complexity of an algorithm is usually taken to be
its **worst-case complexity**, unless specified otherwise.

Analyzing a particular algorithm falls under the field of analysis of algorithms.
To show an upper bound T(n) on the time complexity of a problem,
one needs to show only that there is a particular algorithm
with running time at most T(n).

However, proving lower bounds is much more difficult,
since lower bounds make a statement about
all possible algorithms that solve a given problem.

https://en.wikipedia.org/wiki/Algorithm

Upper and Lower Bounds

The phrase "all possible algorithms" includes
not just the algorithms known today,
but any algorithm that might be discovered in the future.

To show a lower bound of T(n) for a problem requires
showing that no algorithm can have time complexity lower than T(n).

Upper and lower bounds are usually stated using the big O notation,
which hides constant factors and smaller terms.

This makes the bounds independent of the specific details
of the computational model used.

For instance, if $T(n) = 7n^2 + 15n + 40$,
in big O notation one would write $T(n) = O(n^2)$.

https://en.wikipedia.org/wiki/Algorithm

NP Problems

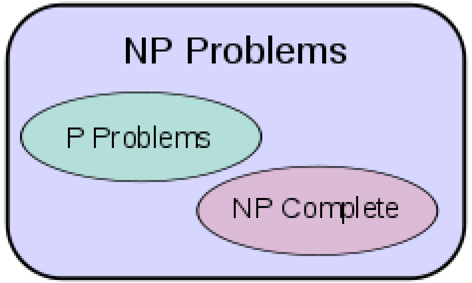
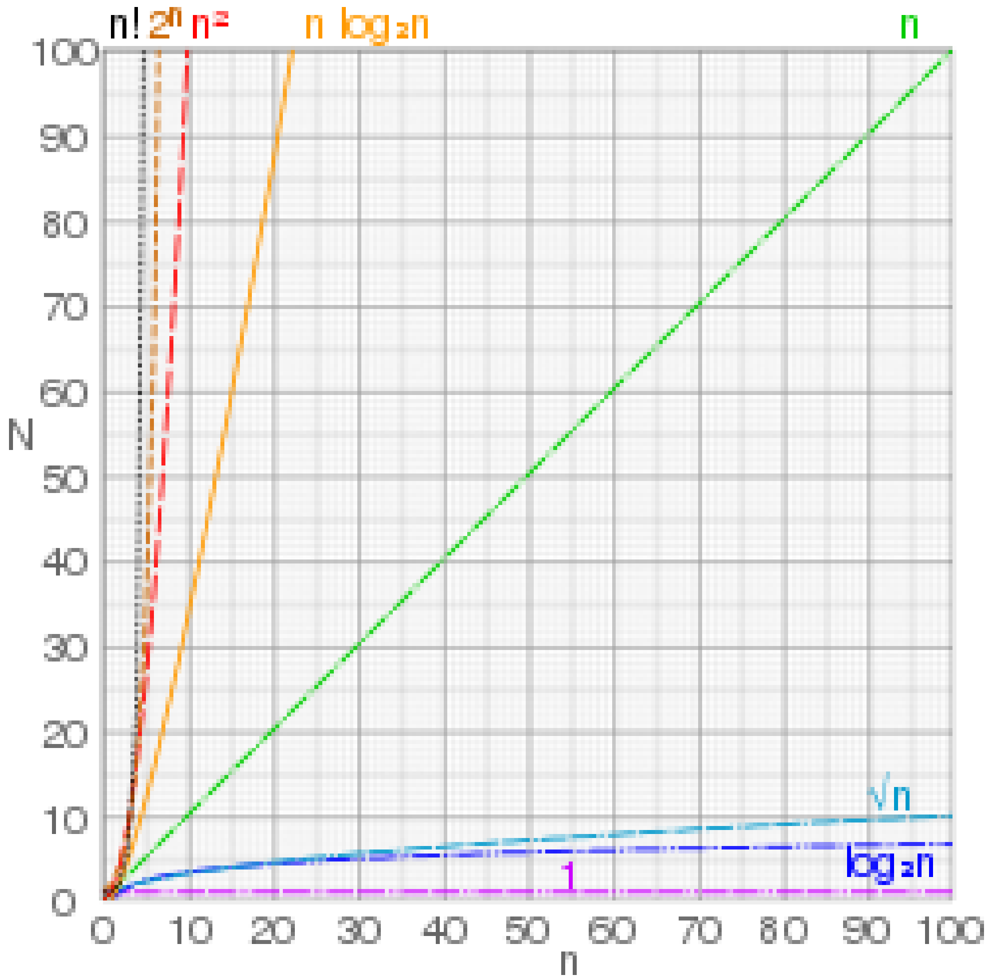


Diagram of complexity classes provided that P ≠ NP. The existence of problems in NP outside both P and NP-complete in this case was established by Ladner.[3]

https://en.wikipedia.org/wiki/Algorithm

Algorithm Analysis



https://en.wikipedia.org/wiki/Algorithm

Graphs of number of operations,
N vs input size n
for common complexities,
assuming a coefficient of 1

Big-O

Let *f* and *g* be two functions defined on some subset of the real numbers. One writes

$$f(x) = O(g(x)) \text{ as } x \to \infty$$

if and only if there is a positive constant *M* such that for all sufficiently large values of *x*, the absolute value of *f*(*x*) is at most *M* multiplied by the absolute value of *g*(*x*). That is, *f*(*x*) = *O*(*g*(*x*)) if and only if there exists a positive real number *M* and a real number x_0 such that

$$|f(x)| \le \; M|g(x)| \text{ for all } x \ge x_0 \,.$$

In many contexts, the assumption that we are interested in the growth rate as the variable *x* goes to infinity is left unstated, and one writes more simply that

$$f(x) = O(g(x)) \,.$$

The notation can also be used to describe the behavior of *f* near some real number *a* (often, *a* = 0): we say

$$f(x) = O(g(x)) \text{ as } x \to a$$

if and only if there exist positive numbers *δ* and *M* such that

$$|f(x)| \le \; M|g(x)| \text{ when } 0 < |x - a| < \delta \,.$$

If *g*(*x*) is non-zero for values of *x* sufficiently close to *a*, both of these definitions can be unified using the limit superior:

$$f(x) = O(g(x)) \text{ as } x \to a$$

if and only if

$$\limsup_{x \to a} \left| \frac{f(x)}{g(x)} \right| < \infty.$$

https://en.wikipedia.org/wiki/https://en.wikipedia.org/wiki/Big_O_notation

Upper and Lower Bounds

Product [edit]

$f_1 = O(g_1) \text{ and } f_2 = O(g_2) \Rightarrow f_1 f_2 = O(g_1 g_2)$

$f \cdot O(g) = O(fg)$

Sum [edit]

$f_1 = O(g_1) \text{ and } f_2 = O(g_2) \Rightarrow f_1 + f_2 = O(|g_1| + |g_2|)$

This implies $f_1 = O(g) \text{ and } f_2 = O(g) \Rightarrow f_1 + f_2 \in O(g)$, which means that $O(g)$ is a convex cone.

If f and g are positive functions, $f + O(g) = O(f + g)$

Multiplication by a constant [edit]

Let k be a constant. Then:

$O(kg) = O(g)$ if k is nonzero.

$f = O(g) \Rightarrow kf = O(g).$

https://en.wikipedia.org/wiki/https://en.wikipedia.org/wiki/Big_O_notation`

Big O Notation

$$witness\ (C, k)$$

$$f(x) \leq C\,g(x) \quad for\ \ x > k$$

$$f(x)\ is\ O(g(x))$$

Discrete Mathematics, Rosen

Big O Notation Examples

$$witness\ (C, k)$$

$$f(x) \le C\, g(x) \qquad for\ \ x > k$$

$$f(x)\ is\ O(g(x))$$

$$x^2 + 2x + 1\ is\ O(x^3)$$

$$1 < x \qquad 1 < x^2 \qquad x < x^2 \qquad\qquad x^2 + 2x + 1 < x^2 + 2x^2 + x^2 = 4x^2$$

$$x^2 + 2x + 1 < 4x^2$$

Discrete Mathematics, Rosen

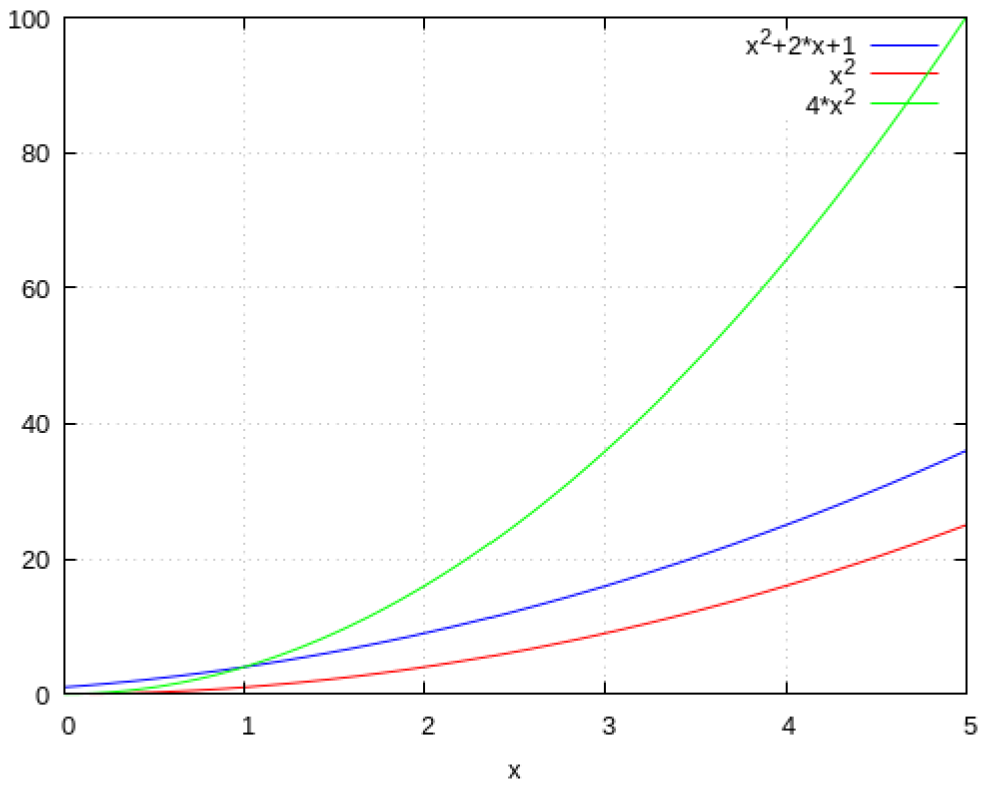
$f(x) = x^2 + 2x + 1$, $g(x) = x^2$

(%i23)

(%o23) $f(x) := 1 + 2\,x + x^2$

(%i24)

(%o24) $g(x) := x^2$



The same class function examples

(%i32)

(%o27) $f1(x) := 1+2\,x+x^2$

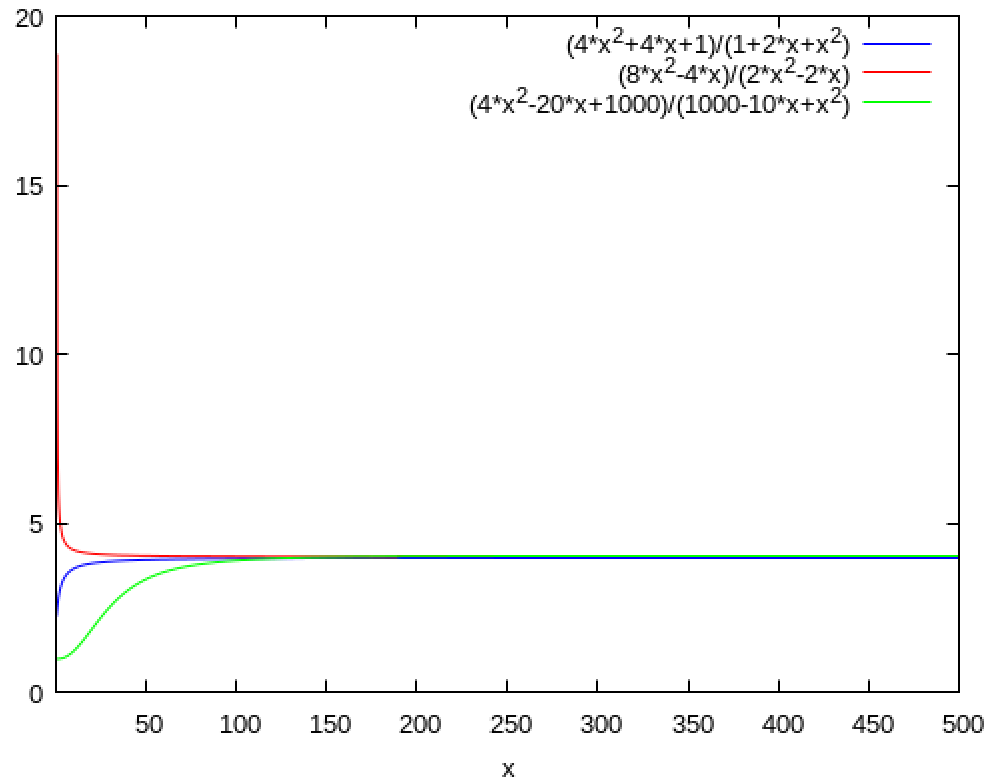
(%o28) $rf1(x) := \dfrac{f1(2\,x)}{f1(x)}$

(%o29) $f2(x) := 2\,x^2-2\,x$

(%o30) $rf2(x) := \dfrac{f2(2\,x)}{f2(x)}$

(%o31) $f3(x) := 1000-10\,x+x^2$

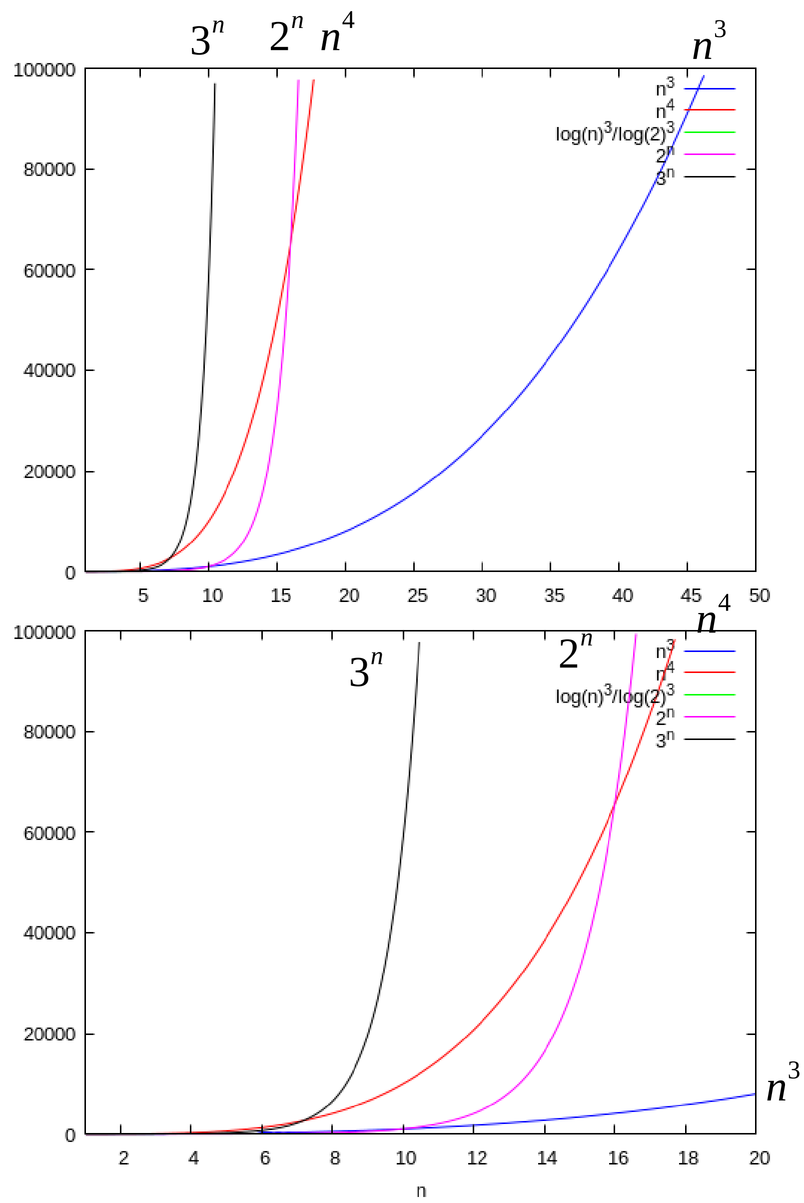
(%o32) $rf3(x) := \dfrac{f3(2\,x)}{f3(x)}$

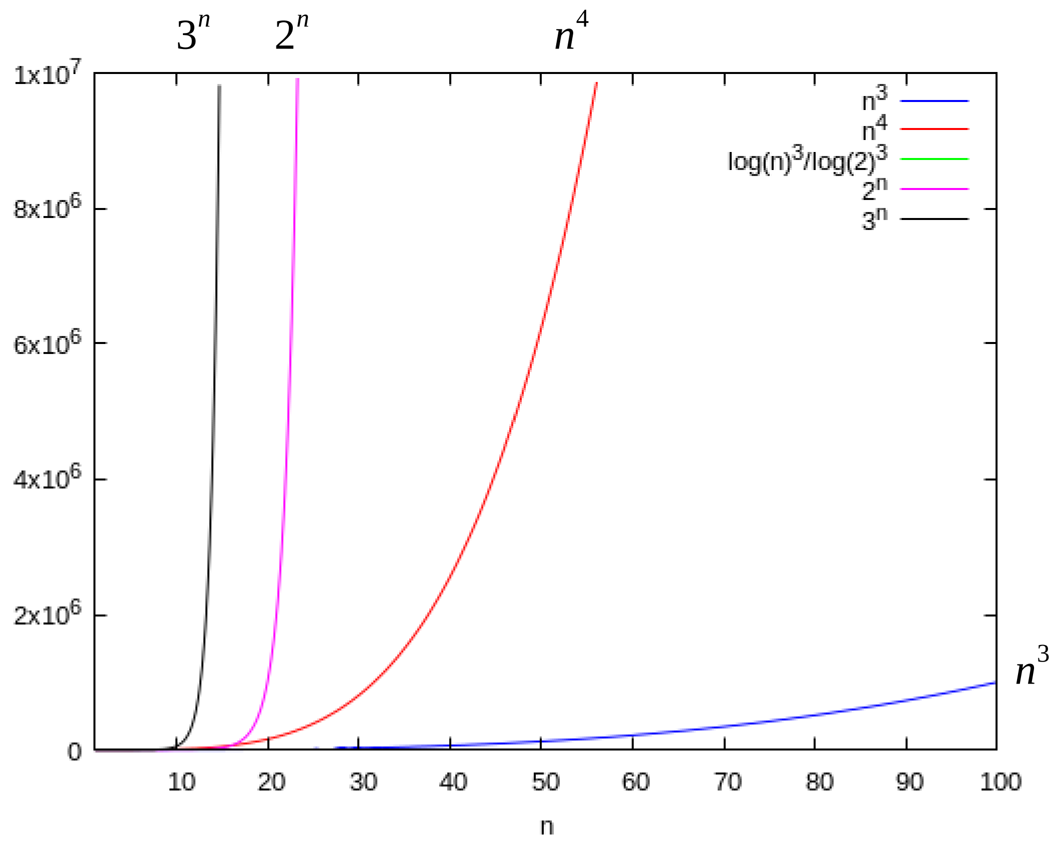


$x \rightarrow 2x$

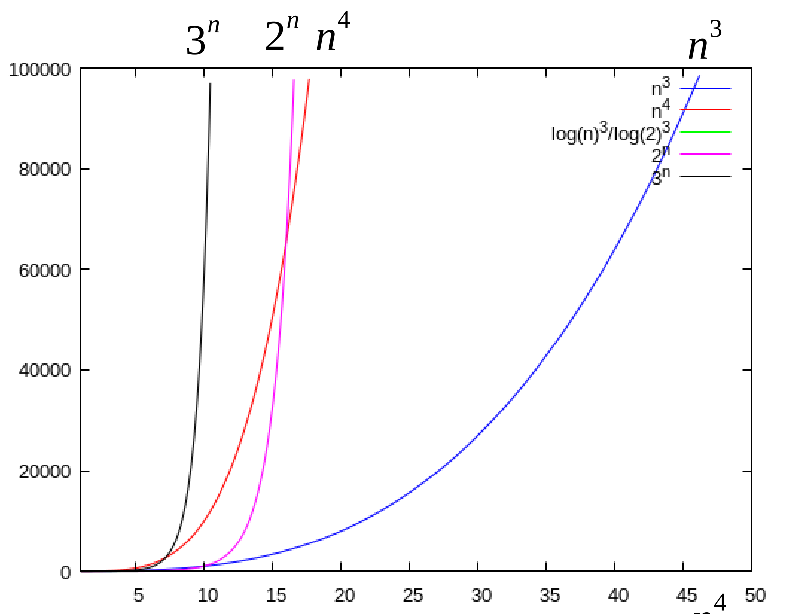
$f_1(2x) \rightarrow 4 f_1(x)$
$f_2(2x) \rightarrow 4 f_2(x)$
$f_3(2x) \rightarrow 4 f_3(x)$

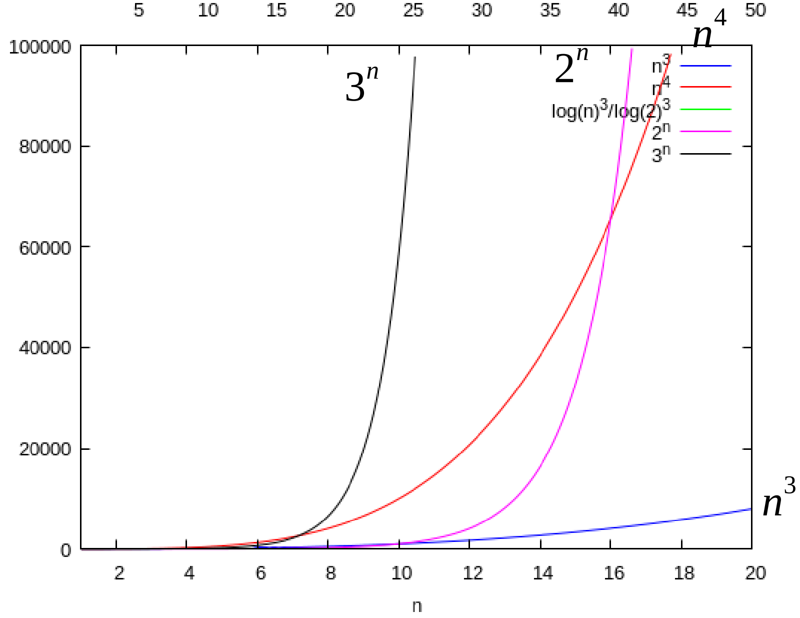
Growth Examples (1)

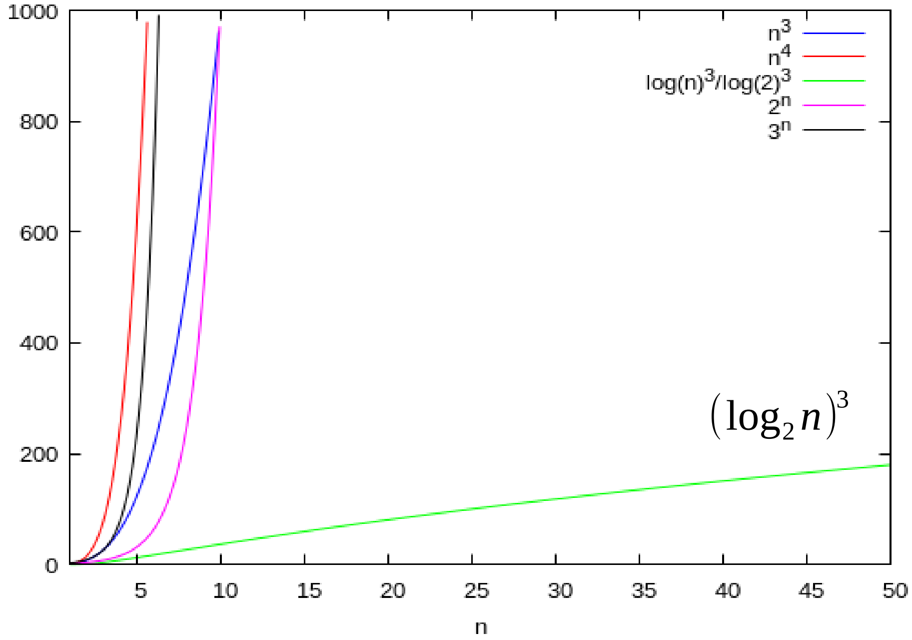




Growth Examples (2)







References

[1] http://en.wikipedia.org/
[2]