Fundamentals

Mathematical Background

- The *floor* of X is the largest integer ≤ X
- The *ceiling* of X is the smallest integer ≥ X
- The *logarithm* of X *to base* Y is the value Z that satisfies the equation $\log_y X = Z$, $X = Y^Z$

Probabilities

- A *probability* is a number between 0 and 1
- If something will never occur, it has a probability of 0
- If something always occurs, it has a probability of 1
- If there are N equally likely events, each one has a probability of $\frac{1}{N}$

Summations

- A *summation* is a compact way to express the sum of a series of values

- The sum of the numbers from 1 to 7 would be written as:

$$\sum_{i=1}^{7} i$$

- There are closed forms for many summations

Analysis of Algorithms

- Investigation of an algorithm's efficiency with respect to:
  - Running time: how fast an algorithm runs
  - Memory space: extra space the algorithm requires
  - Efficiency: a function of input size n

- Comparison of two or more algorithms that solve the same problem

- Independent of the computer because a faster computer does not make an algorithm more efficient

- Independent of initializations because those may be done faster

Space Complexity

- The amount of space needed for an algorithm to complete its task
- Algorithms are classified as either *in place* or needing *extra space*
  - Depends on whether the algorithm moves data around within its current storage or copies information to a new space while it's working
- Use of extra space can be critical in software designed for small embedded systems

In-place Algorithm

- Transforms a data structure using a small, constant amount of extra storage space.
- The input is usually overwritten by the output as the algorithm executes.
- For example, to reverse an array of *n* items

```
function reverse(a[0..n])
    allocate b[0..n]
    for i from 0 to n
        b[n - i] = a[i]
    return b
```

```
function reverse-in-place(a[0..n])
    for i from 0 to floor(n/2)
        swap(a[i], a[n-i])
```

Measuring Running time

- *Time* is not merely CPU clock cycles, we want to study algorithms independent of implementations, platforms, and hardware.

- We measure *time* by "the number of operations as a function of an algorithm's *input size*."

Input Size

- For a given problem, we characterize the input size, *n*, appropriately:
  - *Sorting*: the number of items to be sorted
  - *Graphs*: the number of vertices and/or edges
  - *Numerical*: the number of bits needed to represent a number

- The choice of an input size greatly depends on the *basic operation*, the most important operation that contributes the most to the total running time.

What to Count

- Comparisons
  - Equal, greater, not equal, …
- Arithmetic
  - Additions
    - add, subtract, increment, decrement
  - Multiplications
    - multiply, divide, modulus, remainder

Measuring Running time

- Compute the number of times the basic operation is executed, $C(n)$
- Estimate running time: $T(n) \approx c_{op}C(n)$
  - c_{op}: execution time of a basic operation on a particular computer

Order of Growth

- Order of growth within a constant multiple as $n \to \infty$

- Answers questions of type:
  - How much faster will algorithm run on computer that is twice as fast?

  - How much longer does it take to solve problem of double input size?

Order of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

https://www.wolframalpha.com/input/?i=N%5E2

Order of Growth

- Assume MHz machine: 10^6 operations/second

Minute	Hour	Day	Month	Year
10^8 ops	10^9 ops	10^{11} ops	10^{12} ops	10^{13} ops

- Clearly, any algorithm requiring more than 10^{14} operations is impractical

Order of Growth

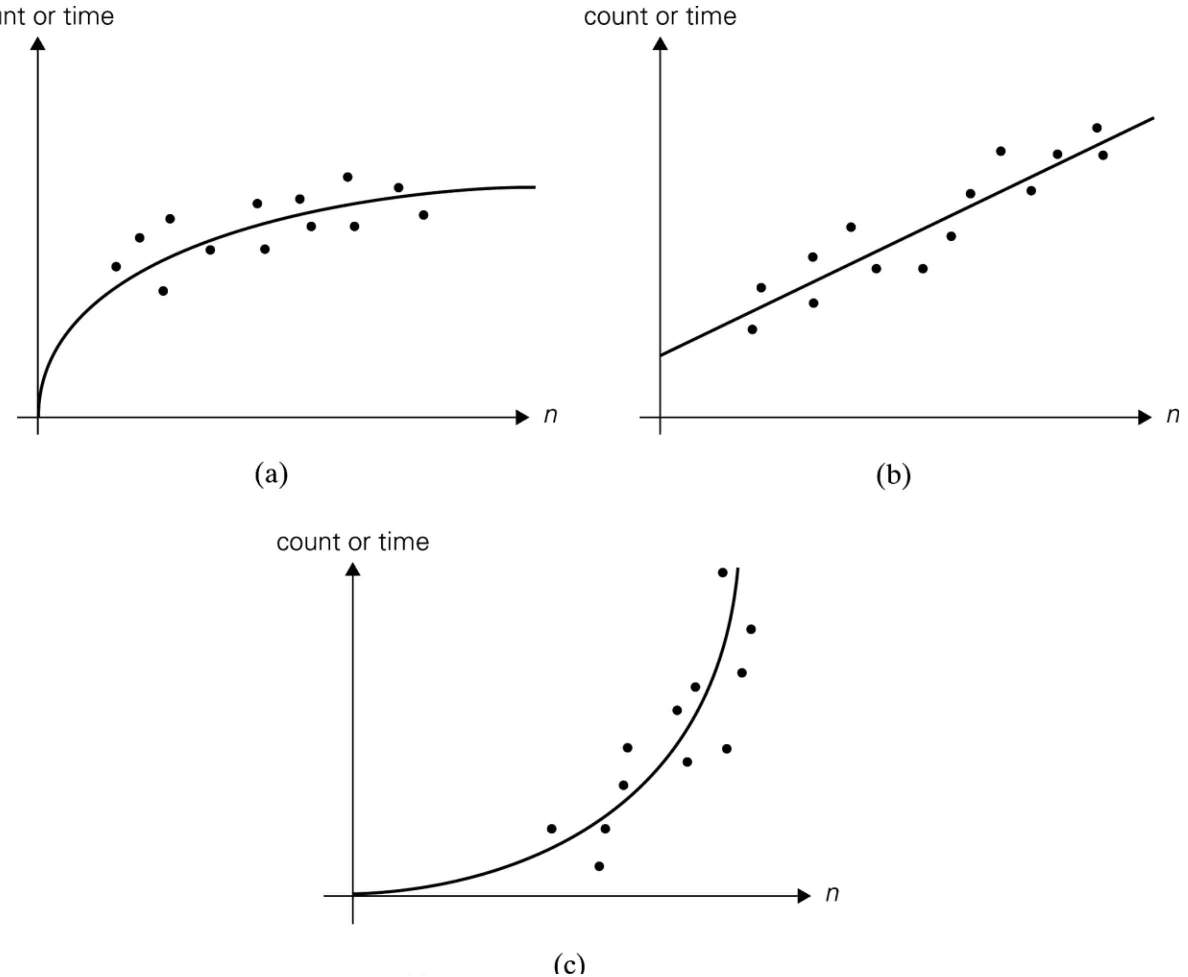
n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

< day   < month   > year

Empirical Analysis of Time Efficiency

- Select a specific (typical) sample of inputs
  - Use physical unit of time (e.g., milliseconds)
  - Count actual number of basic operation's executions
- Analyze the empirical data

Empirical Analysis of Time Efficiency



Typical scatterplots: (a) logarithmic; (b) linear; (c) one of the convex functions

Algorithm’s Efficiencies

- Running time depends on
  - An input size
  - The specifics of a particular input
  - The algorithm itself

Cases to Consider

- Best Case
  - The least amount of work done for any input set
  - $C_{best}(n)$ – minimum over inputs of size n.

- Worst Case
  - The most amount of work done for any input set
  - $C_{worst}(n)$ – maximum over inputs of size n.

- Average Case
  - The amount of work done averaged over all of the possible input sets
  - $C_{avg}(n)$ – “average” over inputs of size n.

Average Case

- Number of times the basic operation will be executed on typical input
- NOT the average of worst and best case
- Determining the average case:
  - Find the number of input set classes m
  - Find the probability that the input will be from each of these classes p_i
  - Find the amount of work done for each class t_i

- The average case is given by:

$$A(n) = \sum_{i=1}^{m} p_i * t_i$$

Types of formulas

- Exact formula

  e.g., $C(n) = \frac{n(n-1)}{2}$

- Formula indicating order of growth with specific multiplicative constant

  e.g., $C(n) \approx 0.5\, n^2$

- Formula indicating order of growth with unknown multiplicative constant

  e.g., $C(n) \approx cn^2$

Asymptotic Notations

- A way of comparing functions that ignores constant factors and small input sizes

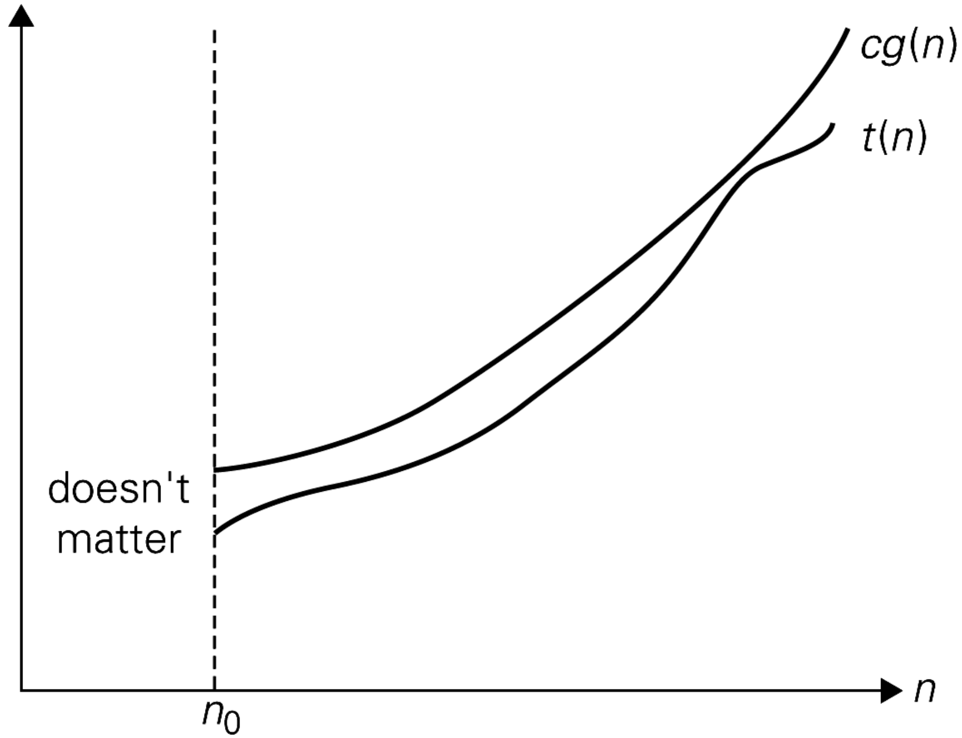
- $t(n) \in O(g(n))$: function $t(n)$ grows *no faster* than $g(n)$

- $\mathrm{t(n)} \in \Theta(g(n))$: function $t(n)$ grows *at same rate* as $g(n)$

- $\mathrm{t(n)} \in \Omega(g(n))$: function $t(n)$ grows at least as fast as $g(n)$

Big-oh

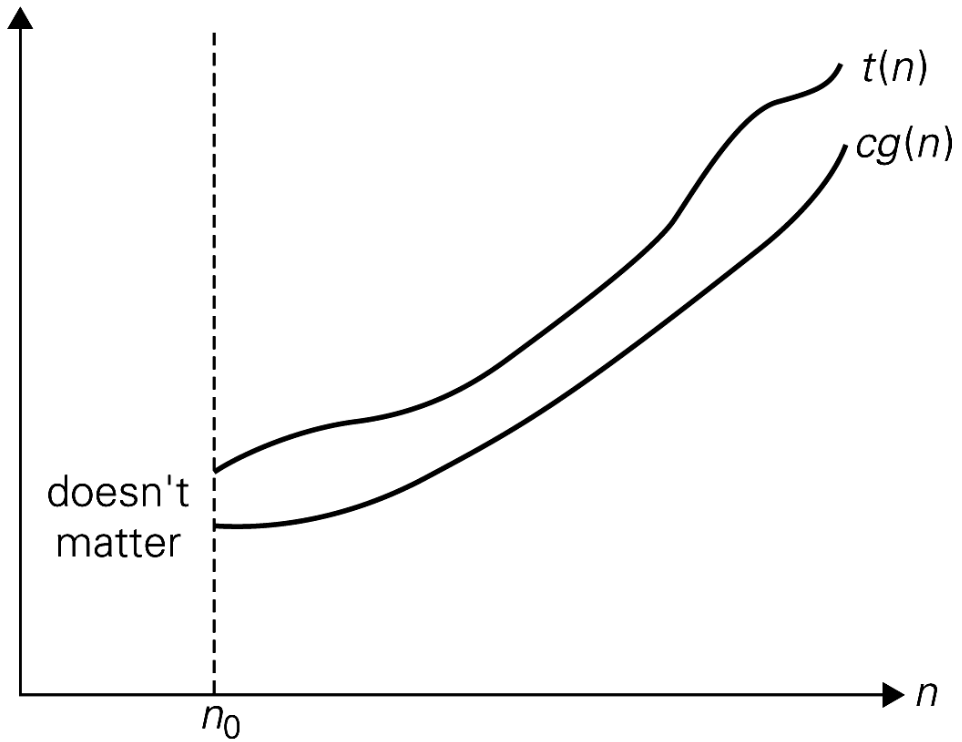
- function $t(n)$ grows *no faster* than $g(n)$



Big-oh notation: $t(n) \in O(g(n))$

Big-omega

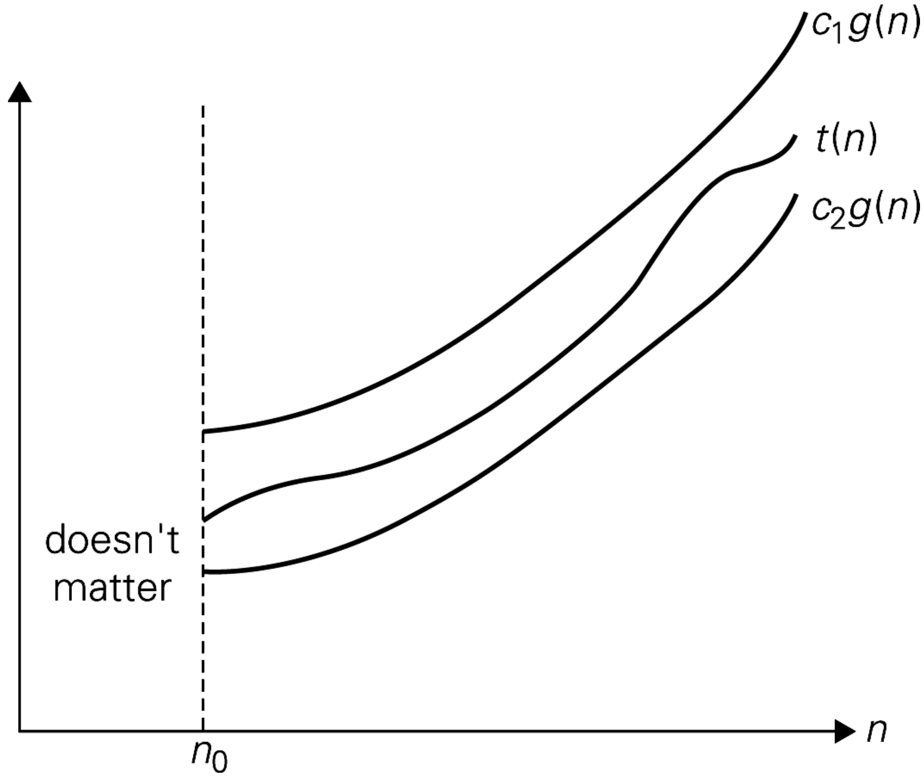
- function $t(n)$ grows <u>*at least as fast*</u> as $g(n)$



Big-omega notation: $t(n) \in \Omega(g(n))$

Big-theta

- function $t(n)$ grows *at same rate* as $g(n)$



Big-theta notation: $t(n) \in \Theta(g(n))$

$$t(n) \epsilon\, \Theta\big(g(n)\big) \wedge t(n) \epsilon\, O\big(g(n)\big) \rightarrow t(n)\ \epsilon\ \Omega(g(n))$$

Asymptotic Properties

- Reflexivity:
  - $f(n) \in O(f(n))$
  - $f(n) \in \Theta(f(n))$
  - $f(n) \in \Omega(f(n))$

- Transitivity:
  - $f(n) \in O\big(g(n)\big) \wedge g(n) \in O\big(h(n)\big) \rightarrow f(n) \in O(h(n))$
  - $f(n) \in \Theta(g(n)) \wedge g(n) \in \Theta(h(n)) \rightarrow f(n) \in \Theta(h(n))$
  - $f(n) \in \Omega(g(n)) \wedge g(n) \in \Omega(h(n)) \rightarrow f(n) \in \Omega(h(n))$

Asymptotic Properties

- Symmetry:
  - $f(n) \in \Theta(g(n))$ iff $g(n) \in \Theta(f(n))$

- Transpose Symmetry:
  - $f(n) \in O(g(n))$ iff $g(n) \in \Omega(f(n))$

Growth Composition

- Theorem:
  - $If\ t_1(n) \in O(g_1(n))\ \ and\ t_2(n) \in O(g_2(n)),$
  - $then\ t_1(n)\ +\ t_2(n) \in O(\max(g_1(n), g_2(n)))$

- If an algorithm runs in two or more stages, we’re only interested in the most expensive stage

- these assertions are true for Θ and Ω

Using Limits

$$\lim_{n \to \infty}\left(\frac{t(n)}{g(n)}\right) = \begin{cases} 0 & \text{order of growth of } t(n) < \text{ order of growth of } g(n) \\ & t(n) \in O(g(n)) \\ c > 0 & \text{order of growth of } t(n) = \text{ order of growth of } g(n) \\ & t(n) \in \Theta(g(n)) \\ \infty & \text{order of growth of } t(n) > \text{ order of growth of } g(n) \\ & t(n) \in \Omega(g(n)) \end{cases}$$

Orders of Growth

- All logarithmic functions $log_a n$ belong to the same class

$$log_a n \in \theta(\log n), for\ all\ a$$

- All polynomials of the same degree k belong to the same class:

$$a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 \in \theta\ (n^k)$$

- Exponentials a^n have different orders of growth for different a's

$$\log n < n^\alpha\ (\alpha > 0) < a^n < n! < n^n$$

L'Hôpital's Rule

- If the derivatives t', g' exist, then

$$\lim_{n->\infty} \frac{t(n)}{g(n)} = \lim_{n->\infty} \frac{t'(n)}{g'(n)}$$

CALCULUS

DERIVATIVES AND LIMITS

DERIVATIVE DEFINITION

$$\frac{d}{dx}(f(x)) = f'(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

BASIC PROPERTIES

$$(cf(x))' = c(f'(x))$$

$$(f(x) \pm g(x))' = f'(x) \pm g'(x)$$

$$\frac{d}{dx}(c) = 0$$

MEAN VALUE THEOREM

If f is differentiable on the interval (a, b) and continuous at the end points there exists a c in (a, b) such that

$$f'(c) = \frac{f(b)-f(a)}{b-a}$$

PRODUCT RULE

$$(f(x)g(x))' = f(x)'g(x) + f(x)g(x)'$$

QUOTIENT RULE

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$$

POWER RULE

$$\frac{d}{dx}(x^n) = nx^{n-1}$$

CHAIN RULE

$$\frac{d}{dx}\left(f(g(x))\right) = f'(g(x))g'(x)$$

COMMON DERIVATIVES

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(\sin x) = \cos x$$

$$\frac{d}{dx}(\cos x) = -\sin x$$

$$\frac{d}{dx}(\tan x) = \sec^2 x$$

$$\frac{d}{dx}(\sec x) = \sec x \tan x$$

$$\frac{d}{dx}(\csc x) = -\csc x \cot x$$

$$\frac{d}{dx}(\cot x) = -\csc^2 x$$

$$\frac{d}{dx}(\sin^{-1} x) = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}(\cos^{-1} x) = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}(\tan^{-1} x) = \frac{1}{1+x^2}$$

$$\frac{d}{dx}(a^x) = a^x \ln(a)$$

$$\frac{d}{dx}(e^x) = e^x$$

$$\frac{d}{dx}(\ln(x)) = \frac{1}{x}, x > 0$$

$$\frac{d}{dx}(\ln|x|) = \frac{1}{x}$$

$$\frac{d}{dx}(\log_a(x)) = \frac{1}{x\ln(a)}$$

CHAIN RULE AND OTHER EXAMPLES

$$\frac{d}{dx}([f(x)]^n) = n[f(x)]^{n-1}f'(x)$$

$$\frac{d}{dx}(e^{f(x)}) = f'(x)e^{f(x)}$$

$$\frac{d}{dx}(\ln[f(x)]) = \frac{f'(x)}{f(x)}$$

$$\frac{d}{dx}(\sin[f(x)]) = f'(x)\cos[f(x)]$$

$$\frac{d}{dx}(\cos[f(x)]) = -f'(x)\sin[f(x)]$$

$$\frac{d}{dx}(\tan[f(x)]) = f'(x)\sec^2[f(x)]$$

$$\frac{d}{dx}(\sec[f(x)]) = f'(x)\sec[f(x)]\tan[f(x)]$$

$$\frac{d}{dx}(\tan^{-1}[f(x)]) = \frac{f'(x)}{1+[f(x)]^2}$$

$$\frac{d}{dx}\left(f(x)^{g(x)}\right) = f(x)^{g(x)}\left(\frac{g(x)f'(x)}{f(x)} + \ln(f(x))\,g'(x)\right)$$

PROPERTIES OF LIMITS

These properties require that the limit of $f(x)$ and $g(x)$ exist

$$\lim_{x\to a}[cf(x)] = c\lim_{x\to a} f(x)$$

$$\lim_{x\to a}[f(x) \pm g(x)] = \lim_{x\to a} f(x) \pm \lim_{x\to a} g(x)$$

$$\lim_{x\to a}[f(x)g(x)] = \lim_{x\to a} f(x)\lim_{x\to a} g(x)$$

$$\lim_{x\to a}\left[\frac{f(x)}{g(x)}\right] = \frac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)} \text{ if } \lim_{x\to a} g(x) \neq 0$$

$$\lim_{x\to a}[f(x)]^n = \left[\lim_{x\to a} f(x)\right]^n$$

LIMIT EVALUATION METHOD – FACTOR AND CANCEL

$$\lim_{x\to -3}\frac{x^2-x-12}{x^2+3x} = \lim_{x\to -3}\frac{(x+3)(x-4)}{x(x+3)} = \lim_{x\to -3}\frac{(x-4)}{x} = \frac{7}{3}$$

L'HOPITAL'S RULE

$$\text{If } \lim_{x\to a}\frac{f(x)}{g(x)} = \frac{0}{0} \text{ or } \frac{\pm\infty}{\pm\infty} \text{ then } \lim_{x\to a}\frac{f(x)}{g(x)} = \lim_{x\to a}\frac{f'(x)}{g'(x)}$$

LIMIT EVALUATIONS AT +-∞

$$\lim_{x\to\infty} e^x = \infty \text{ and } \lim_{x\to -\infty} e^x = 0$$

$$\lim_{x\to\infty} \ln(x) = \infty \text{ and } \lim_{x\to 0^+} \ln(x) = -\infty$$

If $r > 0$ then $\lim_{x\to\infty}\frac{c}{x^r} = 0$

If $r > 0$ & x^r is real for $x < 0$ then $\lim_{x\to -\infty}\frac{c}{x^r} = 0$

$\lim_{x\to\pm\infty} x^r = \infty$ for even r

$\lim_{x\to\infty} x^r = \infty$ & $\lim_{x\to -\infty} x^r = -\infty$ for odd r

Stirling's Formula

$$n! \approx \sqrt{2\pi n} \times (\frac{n}{e})^n$$

Basic Efficiency Classes

Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 5.5). Note that a logarithmic algorithm cannot take into account all its input (or even a fixed fraction of it): any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	"*n-log-n*"	Many divide-and-conquer algorithms (see Chapter 4), including mergesort and quicksort in the average case, fall into this category.
n^2	*quadratic*	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on n-by-n matrices are standard examples.
n^3	*cubic*	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.
2^n	*exponential*	Typical for algorithms that generate all subsets of an n-element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	*factorial*	Typical for algorithms that generate all permutations of an n-element set.