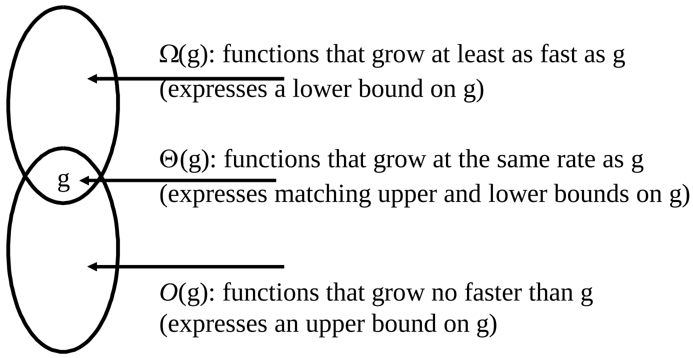
CS 520 Algorithm Analysis
Spring 2015
Lecture 03

I. Asymptotic notation in equations (Knuth) and text pages 49-50
  A. when the asymptotic notation stands alone on the RHS of an equation, the equal sign means set membership
  B. when asymptotic notation appears in a formula, we interpret it as standing for some anonymous function that we do not care to name
  C. the number of anonymous functions in an expression is understood to be equal to the number of times the asymptotic notation appears
  D. when the asymptotic notation appears on the LHS of an equation it means "No matter how the anonymous functions are chosen on the LHS of the equal sign, there is a way to choose the anonymous functions on the RHS of the equal sign to make the equation valid."
  E. the RHS provides a coarser level of detail than the LHS (Knuth follows this philosophy also)
  F. Text pg 44 Asymptotic notation, functions, and running times
    1. Usually use asymptotic notation to describe the running times of algorithms
    2. Usually the worst case running time
    3. Asymptotic can be used to describe memory use or just to describe functions
    4. Sometimes there will be context provided for interpreting the asymptotic notation (some input conditions, all input conditions, best case, worst case,..)

II. Chapter 3 Growth of Functions
  A. From Rawlings, *Compared to What? An introduction to the analysis of algorithms*, page 43
    1. Suppose you had a keyboard with a broken "≤" key, and you used the sequence "=*L*" instead
    2. How do you begin to interpret relations such as $1 = L(3)$ and $3 = L(5)$? Do you start thinking about $L()$ as a function?
    3. If you think of $L(5)$ as an unspecified number less than 5, you can do "arithmetic" with this function
      a) $1 + L(5) = L(6)$
      b) $L(3) + L(1) = L(4)$
      c) $L(3)\ L(4) = L(12)$
    4. You cannot state, though, that $L(5) - L(3) = L(2)$ since the indeterminacy about what is subtracted could make this almost any number
    5. What about $L(6) / L(3)$?
    6. Now think about L as a set of functions; $L(g)$ represents an unspecified function f and the only thing we know or care about f is that $f(n)$ is less than $g(n)$ for each n

B. 3.1 Asymptotic notation

1. usually defined in terms of functions whose domains are the set of natural numbers *N* = {0,1,...}, but can be extended to include *R* or limit to subsets of *N*



From *Computer Algorithms: Introduction to Design & Analysis, Third Edition*, Sara Baase and Allen Van Gelder, New York: NY, Addison Wesley, 2000, page 45, Figure 1.5

2. the set $O(g)$ (definition 1.14 page 45 Sara Baase & Alan Van Gelder, *Computer Algorithms: Introduction to Design & Analysis, Third Edition*)
   a) Let g be a function from the nonnegative integers into the positive real numbers. Then $O(g)$ is a set of functions f, also from the nonnegative integers into the positive real numbers, such that for some real constant $c > 0$ and some nonnegative integer constant n_0, $f(n) \le c\, g(n)$ for $n \ge n_0$.
3. Θ-notation
   a) a function $f \in \Theta(g)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = C$ for some constant C such that $0 < C < \infty$
   b) $\Theta(g(n)) = \left\{ \begin{matrix} f(n) : \exists \text{positive constants } c_1, c_2, \text{and } n_0 \\ \text{such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \, \forall \, n \ge n_0 \end{matrix} \right\}$
   c) asymptotically tight bound - for all values n greater than n_0 the value of $f(n)$ lies at or above the lower bounding function and at or below the upper bounding function
      (1) for $n \ge n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor
   d) assume that every function used within Θ-notation is asymptotically non-negative

e) throw away lower-order terms and ignoring the leading coefficient of the highest-order term

4. *O*-notation
   a) when we have only an asymptotic upper bound we use the *O*-notation
   b) upper bound on a function, to within a constant factor
   c) a function $f \in O(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C, \text{ where } C < \infty$, including the case in which the limit is 0
   d) Example
      (1) $f(n) = \frac{n^3}{2}$
      (2) $g(n) = 37n^2 + 120n + 17$
      (3) Show that $g(n) \in O(f(n))$ but $f(n) \notin O(g(n))$
      (4) $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{37n^2 + 120n + 17}{n^3/2} = \lim_{n\to\infty} \left(74/n + 240/n^2 + 34/n^3 \right) = 0$
      (5) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^3/2}{37n^2 + 120n + 17} = \frac{n^3}{74n^2 + 240n + 34} = \infty$
   e) If the limit of the ratio of *f* to *g* exists and it not ∞, then *f* grows no faster than *g*. If the limit is ∞, then *f* does grow faster than *g*.
   f) $O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$
   g) Polynomial of degree k will always be *O*(*nk*)
   h) when we have only an asymptotic upper bound we use the *O*-notation
   i) upper bound on a function, to within a constant factor
   j) definition page 47

5. Ω-notation
   a) a function $f \in \Omega(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C, \text{ where } C > 0$ including the case in which the limit is ∞
   b) $\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$
   c) provides an asymptotic lower bound; bounds the best case running times of an algorithm
   d) no matter what particular input of size *n* is chosen for each value of *n*, the running time on that set of inputs is at least a constant times *g*(*n*), for sufficiently large *n*.
   e) interpretation coaching again on page 49 "no matter what particular input of size *n* is chosen for each value of *n*, the running time on that set of inputs is a constant times *g*(*n*), for sufficiently large *n*

6. *o*-notation
   a) $o(g(n)) = \left\{ \begin{array}{l} f(n): \text{for any positive constant } c > 0, \\ \exists \text{ a constant } n_0 > 0 \text{ such that } 0 \le f(n) < cg(n)\ \forall\, n \ge n_0 \end{array} \right\}$
   b) in *O*-notation, the bound holds for some constant *c* > 0; in *o*-notation, the bound holds for all constants *c* > 0.
   c) limit of ratio of *f*(*n*) and *g(n*) as *n* approaches ∞ page 50
      (1) $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
7. ω-notation
   a) $\omega(g(n)) = \left\{ \begin{array}{l} f(n): \text{for any positive constant } c > 0, \\ \exists \text{ a constant } n_0 > 0 \text{ such that } 0 \le cg(n) < f(n)\ \forall\, n \ge n_0 \end{array} \right\}$
   b) limit of ratio of *f*(*n*) and *g*(*n*) as *n* approaches ∞ page 51
      (1) $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$
8. comparison of functions
   a) transitivity
   b) reflexitivity
   c) symmetry
   d) transpose symmetry
   e) trichotomy

C. 3.2 Standard notations and common functions (reference material for later use)
1. monotonicity
2. floors and ceilings
3. modular arithmetic
4. polynomials
   a) asymptotically positive polynomials
   b) polynomially bounded if $f(n) = n^{O(1)}$, which is equivalent to saying $f(n) = O(n^k)$ for some constant *k*
5. exponentials (especially Napierian-based exponentials)
6. logarithms
   a) use of base 2 when lg is used, base 10 when log, and base *e* when ln
   b) $f(n)$ is polylogarithmically bounded if $f(n) = \lg^{O(1)} n$
      $\lg^b n = o(n^a)\ \forall\, a > 0$
7. factorials
   a) Stirling's approximation
8. the iterated logarithm function, lg*n
9. Fibonacci numbers and the golden ratio
   a) 1202 Leonardo Pisano (Leonardo of Pisa), sometimes called Leonardo Fibonacci (Filius Bonaccii, son of Bonaccio [Knuth v1]

III. Computational Complexity Tarjan (2-7), Horowitz and Sahni (24-39, 501-513, 545-547), Knuth v1 (104-107)

A. Asymptotic Behavior Tarjan (3-4) Knuth v1 (104-107)

1. we can use static or dynamic measures of algorithmic complexity
   a) static - program length
   b) dynamic - run-time or storage space as a function of input size
      (1) most algorithms have a storage bound that is a linear function of input size
2. Asymptotic formulas represent the approximate value of a quantity instead of the exact value (ignore constant factors especially)
3. P. Bachman in 1892 introduced big-oh notation
   a) $H_n = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} = \sum_{1 \le k < n} \frac{1}{k} \text{ where } n > 0$

      $H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \varepsilon, 0 < \varepsilon < \frac{1}{256n^6}$
   b) $H_n = \ln n + \gamma + O\left(\frac{1}{n}\right)$

      $H_n \approx \ln n + \gamma$
   c) Euler's constant, $\gamma =$ .57721 56649, http://www.math.sfu.ca/~cbm/aands/page_68.htm .
4. $O(f(n))$ may be used whenever $f(n)$ is a function of the positive integer n
5. it stands for a quantity not explicitly known except that its magnitude is not very large
6. $O(f(n))$ means that there is a positive constant M such that the number x_n represented by $O(f(n))$ satisfies magnitude(x_n) ≤ M * magnitude($f(n)$) for all $n \ge n_0$ and M and n_0 are usually different for each appearance of O and are not specific
7. magnitude(H_n - ln n - γ) ≤ M/n
8. an example

   $$1^2 + 2^2 + ... + n^2 = \frac{1}{3}n\left(n + \frac{1}{2}\right)(n+1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$
   a) $1^2 + 2^2 + ... + n^2 = O(n^3) \because \left|\frac{1}{3}n^3\right| \le n^3$

      $1^2 + 2^2 + ... + n^2 = \left|\frac{1}{3}n^3\right| O(n^2) \because \left|\frac{1}{2}n^2\right| \le n^2$
9. algebra using big-oh notation
   a) one-way inequalities
   b) $\frac{1}{2}n^2 + n = O(n^2) \text{ but } O(n^2) = \frac{1}{2}n^2 + n \text{ is not correct because}$

      $\text{RHS could have been } \frac{1}{4}n^2$

c) $\alpha(n) = \beta(n)$ means $\alpha(n) \in \beta(n)$

if $\alpha(n) = \beta(n)$ and $\beta(n) = \gamma(n)$ then $\alpha(n) = \gamma(n)$

d) $O(f(n)) - O(f(n)) = O(f(n))$

e) items 4-9 page 106 of Knuth v1

10. $O(f)$ bounded above
11. $\Omega(f)$ bounded below
12. $\Theta(f)$ bounded above and below

B. HS(24-39)

1. asymptotic notation
   a) $f(n) = O(g(n)$ iff there exist two positive constants c and n_0 such that magnitude($f(n)$) ≤ c * magnitude($g(n)$)
   b) if $A(n) = a_m n^m + ... + a_1 n^1 + a_0 n^0$ is a polynomial of degree m then $A(n) = O(n^m)$
2. most common computing times for algorithms
   a) $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$
   b) see figure 1.9 page 29
3. $f(n) = \Omega(g(n))$ iff there exist positive constants c and $n0$ such that $n > n0$, magnitude($f(n)$) ≥ c * magnitude($g(n)$)
   a) bounded below ($\Omega(2^n)$) is said to require exponential time
4. $f(n) = \Theta(g(n))$ iff there exist positive constants c_1, c_2, and n_0 such that for all $n > n0$, $c1$ * magnitude($g(n)$) ≤ magnitude($f(n)$) ≤ $c2$ * magnitude($g(n)$)
5. $f(n) \sim o(g(n))$ iff limit $f(n)/g(n) \to 1$ as $n \to \infty$
6. performance profiling
7. Tarjan describes kinds of analysis
   a) worst case
   b) average case
   c) probabilistic
   d) amortization - where particular algorithms are repeatedly applied