Cormen Ch. 3 - Growth of Functions

- running time growth = asymptotic efficiency
- interested in input size vs. output size as it approaches a limit when the input is increased without bound

Asymptotic notation

- input domain is natural numbers N = {0, 1, 2, 3, ...}
- sometimes extended to real numbers R

Asymptotic notation, functions, and running times

- generally reduce to largest element for running time
- can be used to characterize memory usage and other algorithmic details

O(theta)-notation

- for a given function, g(n), denote O(g(n)) as the set of functions: O(g(n)) = {f(n): for positive constants c1, c2, and n0 such that 0 <= c1 *g(n) <= f(n) <= c2* g(n) for all n >= n0}
- this means that the O(theta)-descriptive function should always bound f(n) between c1 *g(n) and c2* g(n)
- f(n) is equal to g(n) to within a constant factor, i.e. g(n) is an asymptotically tight bound
- f(n) must be asymptotically non-negative, i.e. positive for sufficiently large n
- asymptotically positive - positive for all sufficiently large n

O(omicron)-notation

- O(theta) bounds from above and below
- O(omicron) is used when there is only an asymptotic upper bound
- for a given function, g(n), denote O(g(n)) as the set of functions: O(g(n)) = {f(n): for positive constants c and n0 such that 0 <= f(n) <= c*g(n) for all n >= n0}
- weaker than O(theta) notation
- inclusion in a set defined by O(theta) notation implies inclusion in a related set for O (omicron)

O(omega)-notation

- provides only an asymptotic lower bound
- for a given function, g(n), denote O(g(n)) as the set of functions: O(g(n)) = {f(n): for positive constants c and n0 such that 0 <= c*g(n) <= f(n) for all n >= n0}

- Theorem 3.1 ->
  - For any two functions f(n) and g(n), we have f(n) = O(theta)(g(n)) if and only if f(n) = O (omicron)(g(n)) and f(n) = �O(omega)(g(n))
- omega notation means the minimum cost (running time/memory) will be at least some constant times the given bound g(n) for all sufficiently large n

Asymptotic notation in equations and inequalities

- asymptotic notation is beneficial for communicating about and comparing equations/ algorithms because it removes inessential terms for that sake of focusing on the main factoring term

o-notation (little-o)

- used to denote an upper bound that is not asymptotically tight
- for a given function, g(n), denote o(g(n)) as the set of functions: o(g(n)) = {f(n): for positive constants c > 0 and n0 > 0 such that 0 <= f(n) <= c*g(n) for all n >= n0}
- similar to O(omicron)-notation
  - omicron specifies that the conditions hold for SOME constant c > 0
  - little-o holds for ALL constants c > 0
- implies limit of f(n) / g(n) as n -> inf is 0

w-notation (lower-case theta)

- same as little-o but specifies a lower bound that is not asymptotically tight

Comparing functions

- transitivity holds for all notations
  - f(n) = O(g(n)) and g(n) = O(h(n)) implies f(n) = O(h(n))
- reflexivity holds for all big-O notations
  - f(n) = O(f(n))
- symmetry holds only for O(theta)-notation
  - f(n) = O(g(n)) if and only if g(n) = O(f(n))
- transpose symmetry holds for O(omega) and w-notation