Tractability (Ch 2 [KTJ])

Goal: find algorithms that are "efficient" to solve (computational) problems.

* What do we mean by "efficient"?
  * problems are discrete: we are searching implicitly within a large set of combinations
  * we focus on running time (resource is time)
    However, there are other resources (memory = space)
    So, we analyze running time as a function of input $N$

** Ex: Stable Matching: implicitly looking through $n!$ perfect matchings!
  * input $N$ size preference lists $n$ men $n$ women
    
    $$N = 2n^2$$

* We analyze worst-case running time, i.e., worst possible running time of algorithm to all inputs of size $N$.
  (doing "average-case" is more tricky, what is a random input?)

* A benchmark to worst case running time of algorithm is brute force search (search over search space of possibilities)

** Ex: Stable Matching, we showed there are at most $n^2$ iterations of Algorithm. Later we show each iteration takes a constant # of computation steps.
  
  Brute force: looking at all $n!$ perfect matchings
  
  $n^2 << n!$

** Def: Algorithm is efficient if it is "qualitatively better" than brute force search.

** Vague.
An algorithm has polynomial running time if it is bounded by $C \cdot N^d$ for constants $C > 0$, $d > 0$.

- in such an algorithm doubling the input $N \to 2N$ results in multiplying bound by constant $C(2N)^d = 2^d \cdot C N^d$

**Def.** An algorithm is **efficient** if it has a polynomial running time.

More concrete definition

ex. The GS-algorithm for stable matchings is efficient.

**Order of growth**

need to express worst-case running time of Algorithm on input size $n$ as bounded by some function $f(n)$.

"There is no sense in being precise when you don’t even know what you are talking about." John von Neumann

- knowny worst-case running time is $\sqrt{3} n^2 - 3n + 81$ has drawbacks
  - is too precise
  - hard to get if algorithm is in pseudocode
  - difficult to compare to other algorithms with so many details

We want a more coarse bound like $\leq C \cdot n^2$

let $T(n)$ := worst-case running time algorithm on input size $n$

* $T(n)$ is $O(f(n))$ \( \implies T(n) = O(f(n)) \) \( \implies T(n) \text{ is order } f(n) \) means there exists $C > 0$ and $n_0$ s.t. for $n \geq n_0$ we have $T(n) \leq C \cdot f(n)$ \( \text{ (upper bounded by } f \text{ )} \)

ex. $T(n) = an^2 + bn + c$, $a, b, c > 0$ is $O(n^2)$ since for $n \geq 1$

$an^2 + bn + c \leq an^2 + bn^2 + cn^2 = (a + b + c) \cdot n^2$

Also $T(n)$ is $O(n^3)$, $O(n^4)$, ---

but $O(n^2)$ is a tight bound for $T(n)$. 
\* \( T(n) \) is \( \Omega(g(n)) \)  
(also \( T(n) = \Omega(g(n)) \) )

mean there exists constants \( c > 0 \) and index no \( s.t. \) for \( n \geq n_0 \)

\[ T(n) \geq c \cdot g(n) \]  
(\( T \) lower bounded by \( g \))

ex. \( a \cdot n^2 + b \cdot n + c \geq a \cdot n^2 \) for \( n \geq n_0 \)

\( (a, b, c \geq 0) \)

so \( a \cdot n^2 + b \cdot n + c = \Omega(n^2) \).

Also \( a \cdot n^2 + b \cdot n + c = \Omega(n) \).

Note \( T(n) = \Omega(g(n)) \) iff \( g(n) = O(T(n)) \)

\* \( T(n) \) is \( \Theta(f(n)) \)  
(also \( T(n) = \Theta(f(n)) \) )

mean both \( T(n) = O(f(n)) \) and \( T(n) = \Omega(f(n)) \)

i.e. \( f(n) \) is a tight bound for \( T(n) \).

ex. \( a \cdot n^2 + b \cdot n + c \), \( a, b, c > 0 \) is \( \Theta(n^2) \).

Note that \( \lim_{n \to \infty} \frac{a \cdot n^2 + b \cdot n + c}{n^2} = a \)

Prop: If \( f(n), g(n) \) are functions such that \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0 \)
then \( f(n) = \Theta(g(n)) \).

Pl: If limit exists then there is index no \( s.t. \) for \( n \geq n_0 \)

\[ \frac{1}{c} \leq \frac{f(n)}{g(n)} \leq c \Rightarrow \]

\[ f(n) \leq c \cdot g(n) \]

\( f(n) \geq \frac{1}{c} \cdot g(n) \)

ex (unrelated) \( \Pi(n) \) counts \# of primes \( \leq n \) then

\[ \Pi(n) = \Theta\left(\frac{n}{\log n}\right) \]
Properties (growth rate)

i) \( f = O(g), \ g = O(h) \) then \( f = O(h) \)

\[ \{ \text{(transitivity)} \] 

\[ \text{ii) } f = \Omega(g), \ g = \Theta(h) \text{ then } f = \Theta(h) \]

\[ \text{iii) } f = \Theta(g), \ g = \Theta(h) \text{ then } f = \Theta(h) \]

\[ \text{iv) } f = O(h), \ g = O(h) \text{ then } f + g = O(h) \]

\[ \text{for } i=1, \ldots, k \text{ if } f_i = O(h) \text{ then } f_1 + f_2 + \ldots + f_k = O(h) \]

\[ \text{v) If } g=O(f) \text{ then } f + g = \Theta(f) \]

Exercise True or False:

\[ \sum_{i=1}^{n} i = O(n) \]

\[ = \Omega(n) \]

\[ = \Theta(n^2) \]

Asymptotic bounds of:

polynomials

\[ f(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_1 n + a_0 \text{, } d \text{ degree} \]

Prop If \( f(n) \) is a polynomial in \( n \), degree \( d \), \( a_d > 0 \) then

\[ f(n) = O(n^d) \].

Prop Each term \( a_i n^i \leq |a_i| n^i \) so \( a_i n^i = O(n^i) \)

so \( f(n) = O(n^d) \) by property iv) above.

Prop If \( f(n) \) is a polynomial in \( n \), degree \( d \), \( a_d > 0 \) then

\[ f(n) = \Theta(n^d) \].

Exercise If algorithm has running time \( O(n \log n) \) then this also is polynomial time, since:

\[ \text{for } n \geq 1 \text{, } \log n \leq n \]

so for \( n \geq 1 \), \( n \log n \leq n^2 \).
Logarithms \( \log_b x = y \) means \( b^y = x \)

* \( \log_2 n + 1 \) is the number of bits used to represent \( n \) in binary.
* \( \log_b n + 1 \) is the number of "digits" \( v \) in \( n \) in base \( b \).

So \( \log_b n \) grows very slow:

For all \( b > 1 \) and \( x > 0 \) real \( \log_b(n) = O(n^x) \)

* We ignore the base \( b \) of the logarithm since they are related up to a constant
  \[ \log_a n = \frac{\log_b n}{\log_b a} \]
  i.e. \( \log_a n = \Theta(\log_b n) \)

So we use \( \log n \)

Exponentials \( f(n) = r^n \) for some \( r \) (normally \( r > 1 \))

For all \( r > 1 \) and all \( d > 0 \)
\( n^d = O(r^n) \)

* Sometimes we do not specify the base \( r \), although all exponentials are different. ( \( s > r > 1 \), \( r^n \neq \Theta(s^n) \)), since we just want to convey exponential growth.