PART 4:

Complexity Theory:

Complexity
Time Complexity Classes
Space Complexity Classes

Complexity Theory

- What makes some problems *computationally* hard and others easy?
- Classify solvable problems according to their degree of difficulty as **easy** ones and **hard** ones.
  - **Time Complexity**
  - **Space Complexity**
- **Intractability Theory**

Complexity Hierarchy of Decidable Languages

- The class of decidable languages
- The resources (time & space) required by the best decision procedures?

Tractability Hierarchy of Decidable Languages

- P
- NP
- PSPACE
- EXPTIME

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$

$$P \neq EXPTIME$$

$$P \subset EXPTIME$$

Analysis of Complexity

Decidable Languages

- A language is decidable!
- A problem is solvable!
- A function is computable!

Are All Decidable Languages Equal?

- L = (ab)*
- $WW^R = \{ww^R : w \in \{a, b\}^*\}$
- $WW = \{ww : w \in \{a, b\}^*\}$
- SAT = {*w* : *w* is a wff in Boolean logic and *w* is satisfiable}
- H = {<*M*, *w*> : Turing machine *M* halts on input string *w*}

Complexity Theory

- Are all decidable languages / problems/ functions equal?

- Find efficient algorithms for decidable languages/ problems/ functions!

✓ The Complexity Theory only applies to decidable languages.

Characterizing Problems as Languages

Describe all problems as languages to be decided via encoding!

- Decision problems
- Optimization problems

Problems as Languages

- CONNECTED = {<*G*> : *G* is an undirected graph and *G* is connected}.

- HAMILTONIANCIRCUIT = {<*G*> : *G* is an undirected graph that contains a *Hamiltonian circuit*}.

- PRIMES = {*w* : *w* is the binary encoding of a prime number}

- TSP-DECIDE = {<*G*, *cost*> : <*G*> encodes an undirected graph with a positive distance attached to each of its edges and *G* contains a Hamiltonian circuit whose total cost is less than <*cost*>}.

Measuring Time and Space Complexity

Choosing A Model of Computation

- We use Turing Machines!

Analyzing Time & Space Complexity

- "How long will it take *P* to run?"
- "How much space will *P* use?"

✓ We will state each answer as a function of some number that corresponds to a reasonable measure of the size of the input.

Measuring Time Requirements

timereq(M) is a function of n:

- If M is a ***deterministic*** TM that halts on all inputs, then:

  $timereq(M) = f(n)$ = the maximum number of steps that M executes on any input of length n.

- If M is a ***nondeterministic*** TM all of whose computational paths halt on all inputs, then:

  $timereq(M) = f(n)$ = the number of steps on the longest path that M executes on any input of length n.

Measuring Space Requirements

spacereq(M) is a function of *n*:

- If *M* is a ***deterministic*** TM that halts on all inputs, then:

  spacereq(*M*) = *f*(*n*) = the maximum number of tape squares that *M* reads on any input of length *n*.

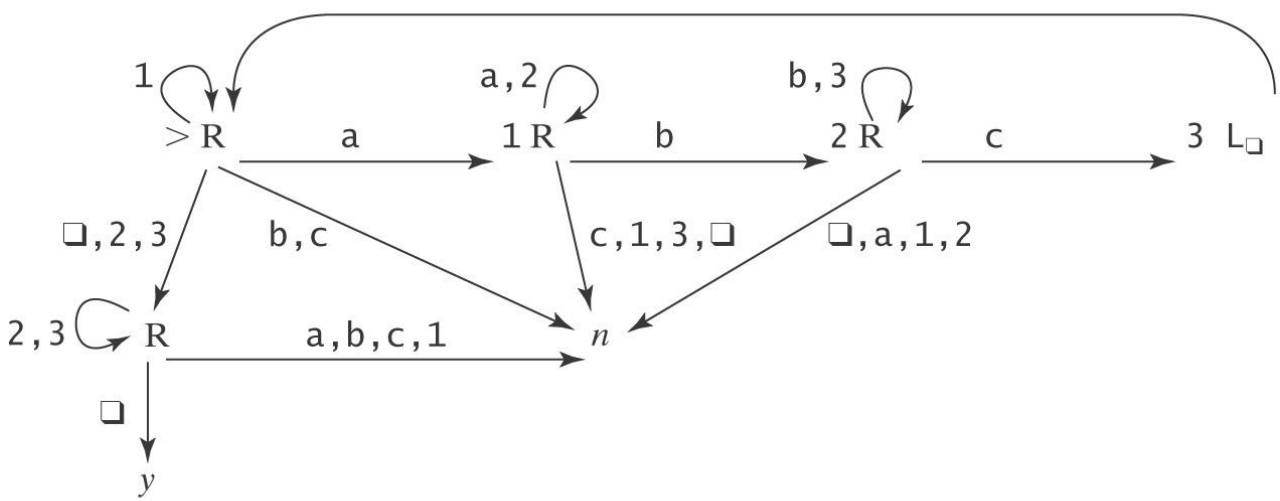
- If *M* is a ***nondeterministic*** TM all of whose computational paths halt on all inputs, then:

  spacereq(*M*) = *f*(*n*) = the maximum number of tape squares that *M* reads on any path that it executes on any input of length *n*.

Example 27.1

L = {$a^n b^n c^n$: $n \geq 0$}

Example: ❑aabbcc❑❑❑❑❑❑❑❑
Example: ❑aaccb❑❑❑❑❑❑❑❑❑



Example 17.8

$L = A^nB^nC^n = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \geq 0\}$ is **decidable**!

TM?

TM M: An informal description!

❑ ❑ ❑ ❑
❑abc ❑ ❑ ❑
❑aabbcc ❑ ❑ ❑
❑abccb ❑ ❑ ❑

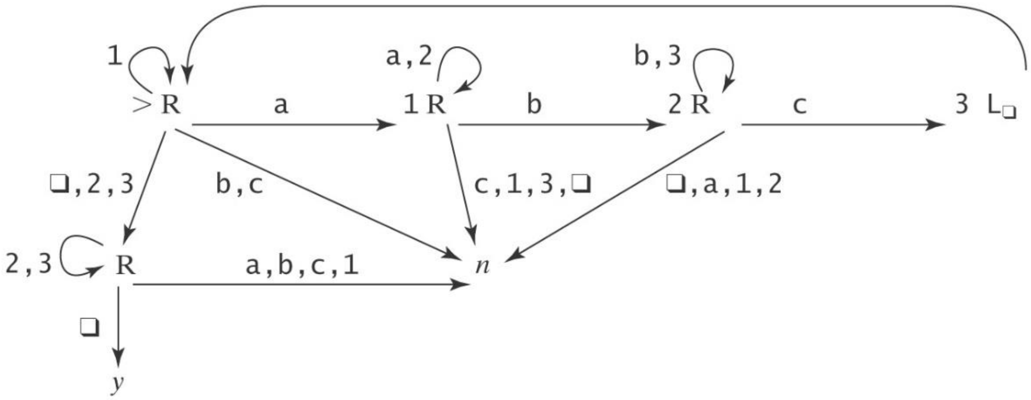
Example 17.8

L= $A^nB^nC^n$ = $\{a^n b^n c^n : n \geq 0\}$ is **decidable**!

TM?

TM M: A graphical notation!

❑ ❑ ❑ ❑
❑abc ❑ ❑ ❑
❑aabbcc ❑ ❑ ❑
❑abccb ❑ ❑ ❑



Example 27.1

1. Move right onto *w*. If the first character is ❑, halt and accept.
2. Loop:
    2.1. Mark off an `a` with a `1`.
    2.2. Move right to the first `b` and mark it off with a `2`. If there isn't one or if there is a c first, halt and reject.
    2.3. Move right to the first `c` and mark it off with a `3`. If there isn't one or there is an `a` first, halt and reject.
    2.4. Move all the way back to the left, then right again past all the `1`'s (the marked off `a`'s). If there is another `a`, go back to the top of the loop. If there isn't, exit the loop.

3. All `a`'s have found matching `b`'s and `c`'s and the read/write head is just to the right of the region of marked off `a`'s. Continue moving left to right to verify that all `b`'s and `c`'s have been marked. If they have, halt and accept. Otherwise halt and reject.

Example 27.1

If $w \in A^nB^nC^n$, the loop will be executed $n/3$ times:

- Each time through the loop, the average number of steps executed is $2(n/3 + n/3 + n/6)$.

Then *M* must make one final sweep all the way through *w*:

- That takes an additional *n* steps.

So the total number of steps *M* executes is:

$$2(n/3)(n/3 + n/3 + n/6) + n.$$

If $w \notin A^nB^nC^n$, the number of steps executed by *M* is lower.

So,

✓ $timereq(M) = 2(n/3)(n/3 + n/3 + n/6) + n$.

✓ The time required to run *M* on an input of length *n* grows as $\boldsymbol{n^2}$.

Example 27.1

M uses only those tape squares that contain its input string, plus the blank on either side of it.

So,

- ✓ *spacereq*(*M*) = n+2
- ✓ The space required to run *M* on an input of length *n* grows as ***n***.

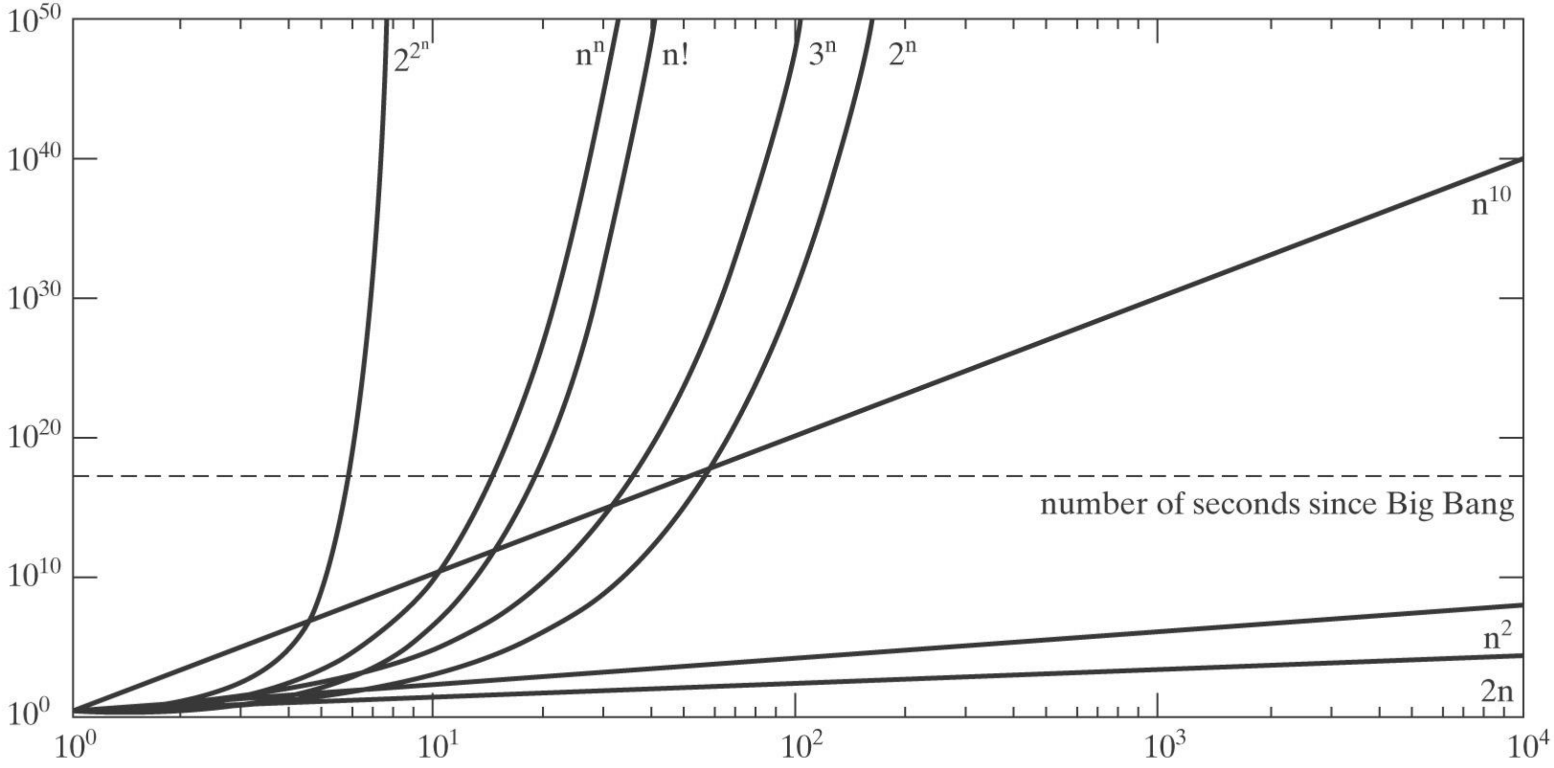
Asymptotic Analysis & Asymptotic Notations

Asymptotic Analysis

- We will ignore small inputs and exact execution counts!

- We will ask whether *P*'s execution time:
  - is constant (i.e., it is independent of n),
  - grows linearly with n,
  - grows faster than n but at a rate that can be described by some polynomial function of n (for example, n^2), or
  - grows at a rate that is faster than any polynomial function of n (for example 2^n).

Growth Rates of Functions



Asymptotic Dominance

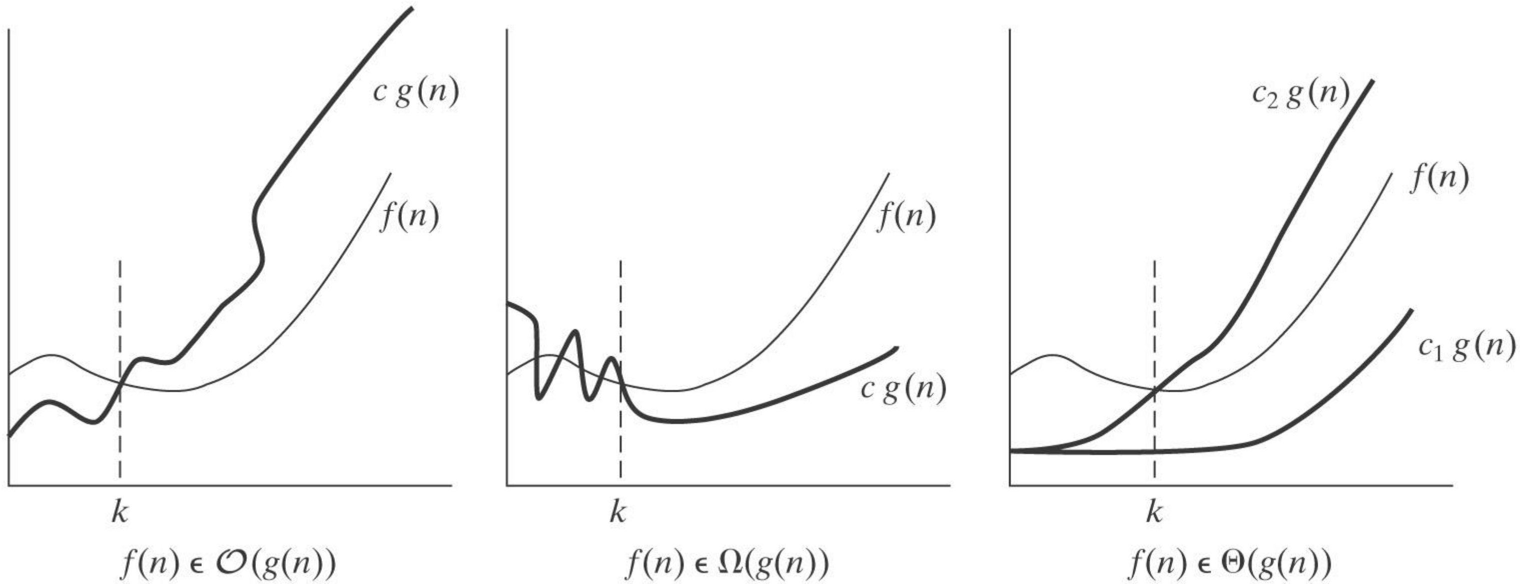
Suppose that *P*, on input of length *n*, executes:

$n^3 + 2n + 3$ steps.

As *n* increases, the n^3 term dominates the other two.

So, we characterize the time required to execute this program as $\boldsymbol{n^3}$.

Asymptotic Notations



Asymptotic Upper Bound - $\mathcal{O}$

Asymptotic upper bound: $f(n) \in \mathcal{O}(g(n))$ iff there exists a positive integer k and a positive constant c such that:

$$\forall n \geq k \; (f(n) \leq c\, g(n)).$$

Alternatively, if the limit exists:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

- f is "**big-Oh**" of g
- g asymptotically dominates or grows at least as fast as f does.
- g is an upper bound on the growth of f.

Asymptotic Dominance - $\mathcal{O}$

- $n^3 \in \mathcal{O}(n^3)$.
- $n^3 \in \mathcal{O}(n^4)$.
- $3n^3 \in \mathcal{O}(n^3)$.
- $n^3 \in \mathcal{O}(3^n)$.
- $n^3 \in \mathcal{O}(n!)$.
- $\log n \in \mathcal{O}(n)$.

Asymptotic Strong Upper Bound - o

Asymptotic strong upper bound: $f(n) \in o(g(n))$ iff, for every positive c, there exists a positive integer k such that:

$$\forall n \geq k \; (f(n) < c\, g(n)).$$

Alternatively, if the limit exists:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

- *f* is "**little-oh**" of *g*
- *g* grows strictly faster than *f* does.

Asymptotic Lower Bound - Ω

Asymptotic lower bound: $f(n) \in \Omega(g(n))$ iff there exists a positive integer k and a positive constant c such that:

$$\forall n \geq k \; (f(n) \geq c\, g(n)).$$

Alternatively, if the limit exists:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

- *f* is "**big-Omega**" of *g*
- *g* grows no faster than *f*.

Asymptotic Strong Lower Bound - ω

Asymptotic strong lower bound: $f(n) \in \omega(g(n))$
iff, for every positive c, there exists a positive integer k such that:

$$\forall n \geq k \ (f(n) > c\, g(n)).$$

Alternatively, if the required limit exists:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

- f is "**little-omega**" of g
- g grows strictly slower than f does.

Asymptotic Tight Bound - Θ

Asymptotic tight bound: $f(n) \in \Theta(g(n))$ iff there exists a positive integer k and positive constants c_1, and c_2 such that:

$$\forall n \geq k \; (c_1\, g(n) \leq f(n) \leq c_2\, g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

- *f* is "**Theta**" of *g*
- *g* is an asymptotically tight bound on the growth of *f.*

Asymptotic Tight Bound - Θ

- $f(n) \in \Theta(g(n))$ iff $f(n) \in \mathcal{O}(g(n))$, and $f(n) \in \Omega(g(n))$.
  - ✓ g(n) is both an upper bound and a lower bound of f(n)!
- $f(n) \in \Theta(g(n))$ iff $f(n) \in \mathcal{O}(g(n))$, and $g(n) \in \mathcal{O}(f(n))$.
  - ✓ f(n) and g(n) are upper bounds of each other!

$\mathcal{O}$ and Θ

Suppose P runs in time $f(n) = 2 + 4n$.

$2 + 4n \in \mathcal{O}(n)$.

$2 + 4n \in \mathcal{O}(n^2)$.

$2 + 4n \in \mathcal{O}(2^n)$,

Define Θ:

$f(n) \in \Theta(g(n))$ iff $f(n) \in \mathcal{O}(g(n))$ and $g(n) \in \mathcal{O}(f(n))$.

So:

$2 + 4n \in \Theta(n)$, but

$2 + 4n \notin \Theta(n^2)$ because $n^2 \notin \mathcal{O}(n)$.

Example 27.2

$timereq(M) = 3n^2+23n+100$

- $timereq(M) \in O(n^2)$?
- $timereq(M) \in O(n^3)$?
- $timereq(M) \in o(n^3)$?
- $timereq(M) \in \Omega(n)$?
- $timereq(M) \in \Omega(n^2)$?
- $timereq(M) \in \Theta(n^2)$?

Facts About $\mathcal{O}$

Theorem 27.1

1. $f(n) \in \mathcal{O}(f(n))$.

Facts About $\mathcal{O}$

2. Addition:

   2.1. $\mathcal{O}(f(n)) = \mathcal{O}(f(n) + c_0)$

   2.2. If $f_1(n) \in \mathcal{O}(g_1(n))$ and $f_2(n) \in \mathcal{O}(g_2(n))$,
   then $f_1(n) + f_2(n) \in \mathcal{O}(g_1(n) + g_2(n))$.

   2.3. $\mathcal{O}(f_1(n) + f_2(n)) = \mathcal{O}(max(f_1(n), f_2(n)))$.

Facts About $\mathcal{O}$

3. Multiplication:

   3.1. $\mathcal{O}(f(n)) = \mathcal{O}(c_0\, f(n))$.

   3.2. If $f_1(n) \in \mathcal{O}(g_1(n))$ and $f_2(n) \in \mathcal{O}(g_2(n))$,
   then $f_1(n)\, f_2(n) \in \mathcal{O}(g_1(n)\, g_2(n))$.

Facts About $\mathcal{O}$

4. Polynomials:

4.1. If $a \le b$ then $\mathcal{O}(n^a) \subseteq \mathcal{O}(n^b)$.

4.2. If $f(n) = c_j n^j + c_{j-1} n^{j-1} + \ldots c_1 n + c_0$,
then $f(n) \in \mathcal{O}(n^j)$.

Facts About $\mathcal{O}$

5. Logarithms:

   5.1. For a and $b > 1$, $\mathcal{O}(\log_a n) = \mathcal{O}(\log_b n)$.

   5.2. If $0 < a < b$ and $c > 1$,
   then $\mathcal{O}(n^a) \subseteq \mathcal{O}(n^a \log_c n) \subseteq \mathcal{O}(n^b)$

Facts About $\mathcal{O}$

6. Exponentials (dominate polynomials):

   6.1. If $1 < a \le b$ then $\mathcal{O}(a^n) \subseteq \mathcal{O}(b^n)$.

   6.2. If $a > 0$ and $b > 1$ then $\mathcal{O}(n^a) \subseteq \mathcal{O}(b^n)$.

   6.3. If $f(n) = c_{j+1}2^n + c_j n^j + c_{j-1}n^{j-1} + \ldots c_1 n + c_0$,
   then $f(n) \in \mathcal{O}(2^n)$.

   6.4. $\mathcal{O}(n^t 2^n) \subseteq \mathcal{O}(2^{(n^s)})$, for some $s>1$.

Facts About $\mathcal{O}$

7. Factorial dominates exponentials:

   If $a \geq 1$,
   then $\mathcal{O}(a^n) \subseteq \mathcal{O}(n!)$.

Facts About $\mathcal{O}$

8. Transitivity:

   If $f(n) \in \mathcal{O}(f_1(n))$ and $f_1(n) \in \mathcal{O}(f_2(n))$,
   then $f(n) \in \mathcal{O}(f_2(n))$.

Summarizing $\mathcal{O}$

$$\mathcal{O}(c) \subseteq \mathcal{O}(\log_a n) \subseteq \mathcal{O}(n^b) \subseteq \mathcal{O}(d^n) \subseteq \mathcal{O}(n!)$$

Example 27.3

$$timereq(M) = 2(n/3)(n/3 + n/3 + n/6) + n$$
$$= (5/9)n^2+n$$

- $timereq(M) \in O(n^2)$?
- $timereq(M) \in O(n^3)$?
- $timereq(M) \in o(n^3)$?

Common Algorithm Growth Rates

A constant growth rate O(1)

- A logarithmic logarithmic growth rate (log (log N))
- A logarithmic growth rate (log N)
- A logarithmic squared growth rate $(\log^2 N)$
- A linear growth rate O(N)
- A linear-logarithmic (?) growth rate O(N log N)
- A quadratic growth rate $O(N^2)$
- A cubic growth rate $O(N^3)$
- A polynomial growth rate $O(N^k)$ for a constant k.

Common Algorithm Growth Rates

- An exponential growth rate $O(2^N)$
- A factorial growth rate $O(N!)$

Tight Bound vs Loose Bound

$\boxed{f}$ = $O(1)$ $O(\log N)$ $O(N)$ $O(N \log N)$ $O(N^2)$ $O(2^N)$

Tight bound ⟷ Loose bound

$\Omega(1)$ $\Omega(\log N)$ $\Omega(N)$ $\Omega(N \log N)$ $\Omega(N^2)$ $\Omega(2^N)$ = $\boxed{f}$

Loose bound ⟷ Tight bound

Algorithmic Gaps

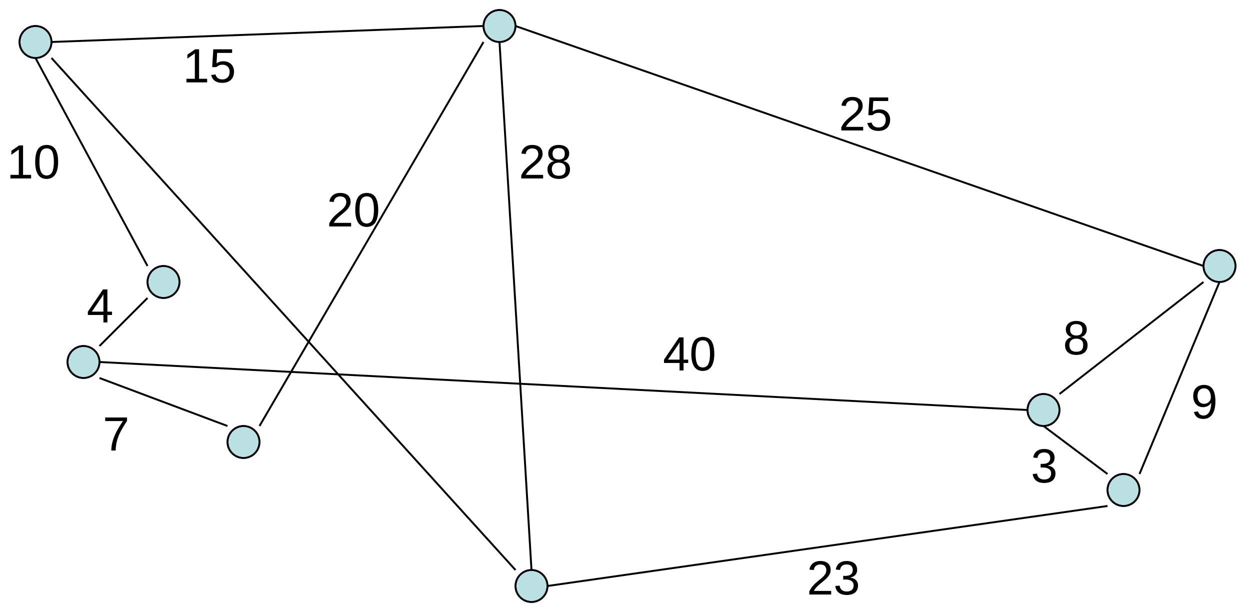
Given a problem L, we'd like to show:

1. Upper bound: There exists an algorithm that decides *L* and that has complexity C_1.

2. Lower bound**:** Any algorithm that decides *L* must have complexity at least C_2.

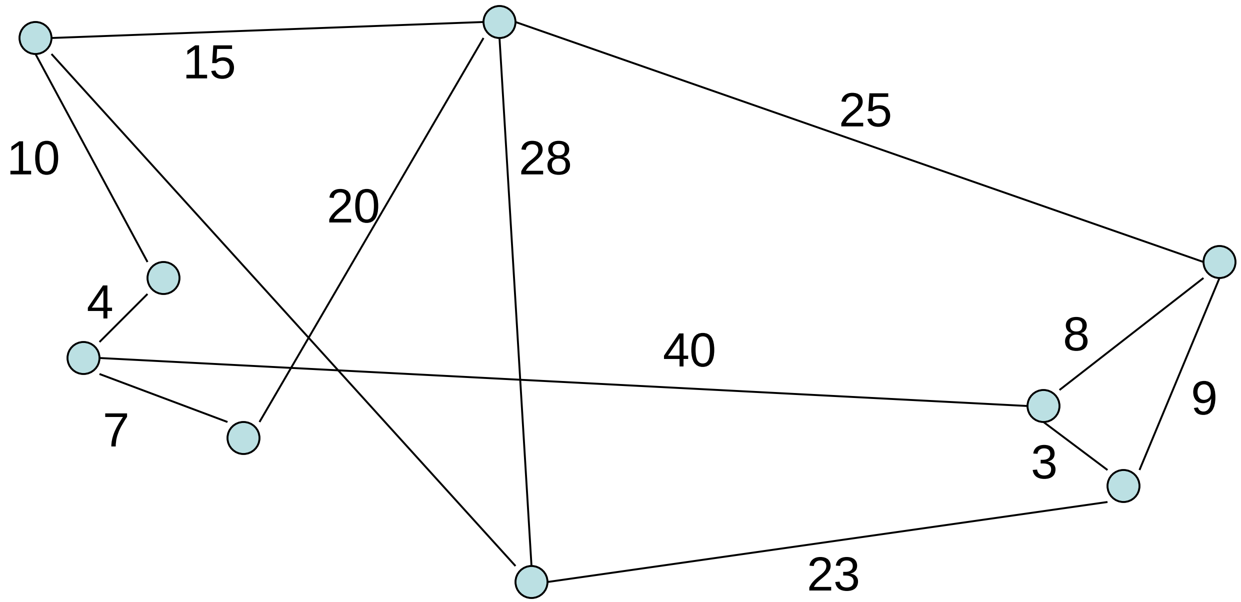
3. $C_1 = C_2$? If $C_1 = C_2$, we are done.
   Often, we're not done.
   For many interesting problems, not done!

The Traveling Salesman Problem



"Given n cities and the distances between each pair of them, find the shortest tour that returns to its starting point and visits each other city exactly once along the way. "

The Traveling Salesman Problem



Given n cities:

Choose a first city n

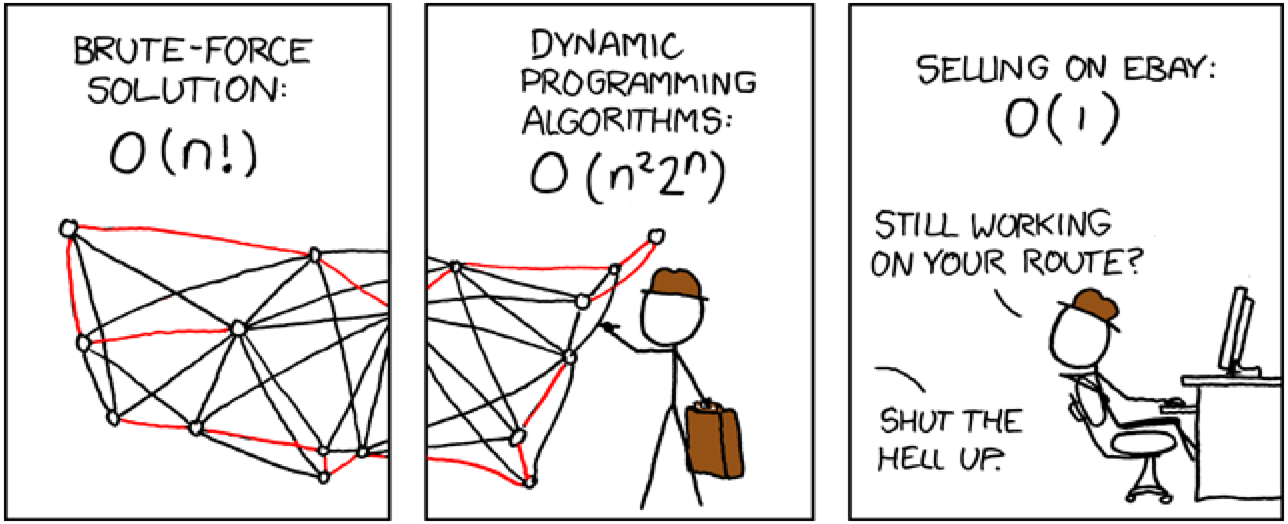
Choose a second n-1

Choose a third $\underline{n\text{-}2}$

… $\boldsymbol{n!}$

TSP is in P???

- Upper bound: *timereq* $\in \mathcal{O}(2^{(n^k)})$.
- Lower bound: Don't have a lower bound that says polynomial isn't possible.



Reading Assignment

Chapter 27:

Sections
27.1
27.2
27.3
27.4
27.5
27.6
27.7

In-Class Exercises

Chapter 27:

1
6
7