Basics of Complexity

"Complexity" = resources

- time
- space
- ink
- gates
- energy

Complexity is a function

- Complexity = *f* (input size)
- Value depends on:
  - problem encoding
    - adj. list vs. adj matrix
  - model of computation
    - Cray vs TM ~$O(n^3)$ difference

TM time complexity

Model: *k*-tape deterministic TM (for any *k*)

DEF: *M* is *T(n)* time bounded iff for every *n*, for every input *w* of size *n*, *M(w)* halts within *T(n)* transitions.

- *T(n)* means max {*n*+1, *T(n)*}
  (so every TM spends at least linear time).
- worst case time measure
- *L* recursive → for some function *T*, *L* is accepted by a *T(n)* time bounded TM.

TM space complexity

Model: "Offline" *k*-tape TM.

read-only input tape

k read/write work tapes initially blank

DEF: M is $S(n)$ space bounded iff for every n, for every input w of size n, $M(w)$ halts having scanned at most $S(n)$ work tape cells.

- Can use less than linear space
- If $S(n) \geq \log n$ then wlog M halts
- worst case measure

Complexity Classes

Dtime($T(n)$) =

{L | exists a deterministic $T(n)$ time-bounded TM accepting L}

Dspace($S(n)$) =

{L | exists a deterministic $S(n)$ space-bounded TM accepting L}

E.g., Dtime(n), Dtime(n^2), Dtime($n^{3.7}$), Dtime(2^n), Dspace($\log n$), Dspace(n), ...

Linear Speedup Theorems

"Why constants don't matter": justifies O( )

If $T(n)$ > linear*, then for *every* constant c > 0,

$$\mathrm{Dtime}(T(n)) = \mathrm{Dtime}(cT(n))$$

For *every* constant c > 0,

$$\mathrm{Dspace}(S(n)) = \mathrm{Dspace}(cS(n))$$

(Proof idea: to compress by factor of 100, use symbols that jam 100 symbols into 1. For time speedup, more complicated.)

* $T(n)/n \to \infty$

Tape Reduction

- If L is accepted by a $S(n)$ space-bdd k-tape TM, then L is also by a $S(n)$ space-bdd 1-tape TM.

  Idea: M' simulates M on 1 tape using k tracks

- If L is accepted by a $T(n)$ time-bdd k-tape TM, then L is also accepted by:
  - A $(T(n))^2$ time-bdd 1-tape TM [proved earlier]
  - A $T(n) \log T(n)$ time-bdd 2-tape TM [very clever]

Time & Space Hierarchies

With more time or space, we can compute more

If $\inf_{n \to \infty} S_1(n)/S_2(n) = 0$ (e.g., $S_1 = o(S_2)$)

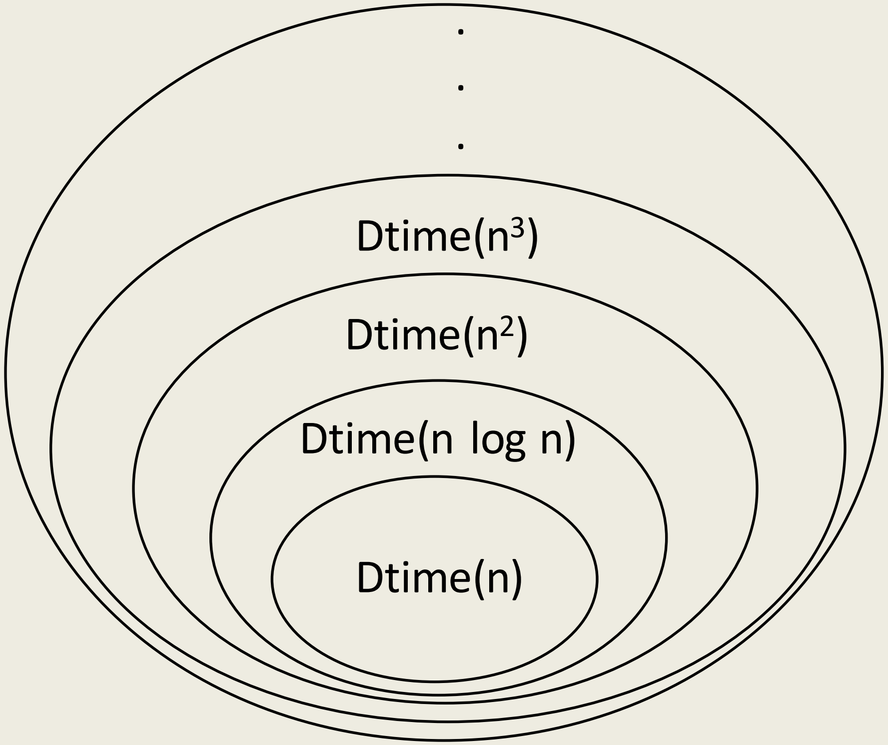
Then $\mathrm{Dspace}(S_1(n)) \subset \mathrm{Dspace}(S_2(n))$

If $\inf_{n \to \infty} T_1(n) \log T_1(n) / T_2(n) = 0$

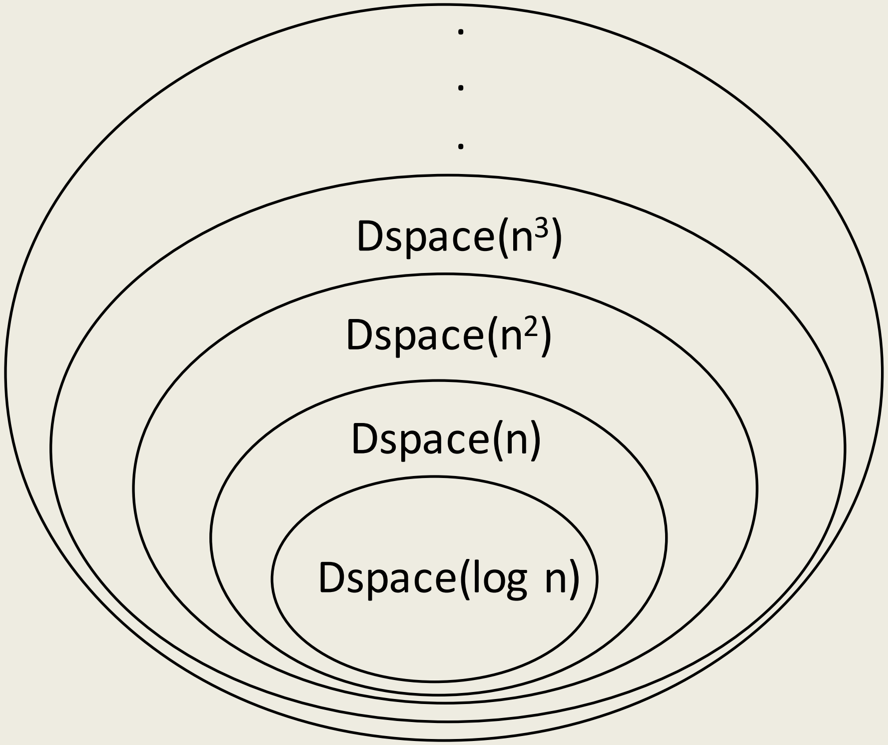
Then $\mathrm{Dtime}(T_1(n)) \subset \mathrm{Dtime}(T_2(n))$

also requires that S_1, S_2, and T_2 are “constructible”

Time & Space Hierarchies



TIME



SPACE

Relationships between Time & Space

- Dtime($f(n)$) $\subseteq$ Dspace($f(n)$)

  You can only use as much space as you have time

Different constant c for each L
Equivalently, $2^{O(f(n))}$

- Dspace($f(n)$) $\subseteq$ Dtime($c^{f(n)}$)

  [if f is constructible and $f(n) \geq \log n$]

If you only have $f(n)$ space, the number of IDs is bounded by $c^{f(n)}$ before you start looping, so may as well halt. [exercise: what is c ?]

Goal: define "efficient" computation

$$P = \bigcup_{k \geq 0} \mathrm{Dtime}(n^k)$$

"Deterministic Polynomial Time"

Union over all polynomials p of Dtime($p(n)$))

Worst-case

Advantages

- easy to analyze
- gives guarantee
- don’t have to decide what “typical” inputs are

Disadvantages

- bizarre inputs created by bored mathematicians proving lower bounds can force algorithms to take longer than any input you’re ever liable to see

Reasons why P is a bad def

- Worst case
- Asymptotic
- Ignores constants: $10^{100}n$ *versus* $10^{-100}2^n$

Reasons why P is a good def

- Model invariance (RAM, TM, Cray, ...)
- Invariant to input encoding
- poly(poly(n)) = poly(n), so “efficient” composes
- Typical algs found are $O(n^{\text{small-constant}})$
- Moderate growth rate of polys vs. exps...

Understatement: Exponentials are Big

1,000,000,000,000,000 operations per second					
n	**n^2**	**n^3**	**n^5**	**2^n**	**n!**
10	1E-13	1E-12	1E-10	1.024E-12	
20	4E-13	8E-12	3.2E-09	1.04858E-09	
30	9E-13	2.7E-11	2.43E-08	1.07374E-06	
40	1.6E-12	6.4E-11	1.024E-07	0.001099512	
50	2.5E-12	1.25E-10	3.125E-07	1.125899907	
60	3.6E-12	2.16E-10	7.776E-07		
70	4.9E-12	3.43E-10	1.6807E-06		
80	6.4E-12	5.12E-10	3.2768E-06		
90	8.1E-12	7.29E-10	5.9049E-06		
100	1E-11	1E-09	0.00001		

Death of Sun: 5 GigaYears

Understatement: Exponentials are Big

1,000,000,000,000,000 operations per second					
n	**n^2**	**n^3**	**n^5**	**2^n**	**n!**
10	1E-13	1E-12	1E-10	1.024E-12	3.6288E-09
20	4E-13	8E-12	3.2E-09	1.04858E-09	2432.902008
30	9E-13	2.7E-11	2.43E-08	1.07374E-06	$\boxed{\text{8 GigaYears}}$
40	1.6E-12	6.4E-11	1.024E-07	0.001099512	2.5E+25 Years
50	2.5E-12	1.25E-10	3.125E-07	1.125899907	silly
60	3.6E-12	2.16E-10	7.776E-07	19 min	silly
70	4.9E-12	3.43E-10	1.6807E-06	13 days	silly
80	6.4E-12	5.12E-10	3.2768E-06	38 years	silly
90	8.1E-12	7.29E-10	5.9049E-06	39K years	silly
100	1E-11	1E-09	0.00001	40M years	silly

Death of Sun: 5 GigaYears