Lecture 14

Real Computers are *Finite*

Unbounded "memory" is critical to most undecidability pfs

Real computers are finite: n bits of state (registers, cache, RAM, HD, …) $\Rightarrow \leq 2^n$ configs – it's a DFA!

"Does M accept w" is *decidable*: run M on w; if it runs more that 2^n steps, it's looping. (Recall LBA pfs.)

BUT:

- 2^n is *astronomical:* a modest laptop has n = 100's of gigabits of state; # atoms in the universe ~ 2^{262}

Are "real" computer problems undecidable?

Options:

- 100 G is so much >> 262, let's say it's approximately unbounded ⇒ undecidable
- Explore/quantify the "computational difficulty" of solving the (decidable) "bounded memory" problem

1st is somewhat crude, but easy, and not crazy, given that we really don't have methods that are fundamentally better for 100Gb memories than for arbitrary algorithms

2nd is more refined but harder; goal of next few weeks is to develop theory supporting such aims

Measuring "Compute Time"

TM: simple, just count steps

Defn: If M is a TM deciding L, the *time complexity of M* is the function T(n) such that T(n) is the max number of steps taken by M on any input $w \in \Sigma^*$ of length n.

Why as a function of n? Mainly to smooth and summarize

Loosely, the *time complexity of L* is the least such T over all M deciding L.

(I say "loosely" because it may be that no one M is fastest on all inputs, but nevertheless we may be able to bound it.)

Example: $L = \{ a^n b^n \mid n \geq 0 \}$
(on a One-Tape TM)

A simple algorithm (zig-zag, cross off letters): $T(n) = \sim n^2$

Somewhat trickier: cross of 5 letters at a time: $T(n) = \sim n^2/5$

A more complex algorithm:

- On a "two-track" tape, drag along a binary counter: $T(n) = \sim n \log_2 n$

Slightly more work:

- As above, but a decimal counter: $T(n) = \sim n \log_{10} n$

More work still:

- As above, but use lots of states to count off 1st ten million a's & b's: $T(n) = \sim$ if $(n < 10^7)$ then n else $n \log_{10} n$

One conclusion:

Focus on growth rate, not const or small n. I.e., big-O

Complexity Classes

Defn:

TIME(T(n)) = the set of languages decidable by single-tape TMs in time O(T(n))

E.g. $\{ a^n b^n \mid n \geq 0 \} \in \text{TIME}(n \log n)$

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

(on a Two-Tape TM)

Counter on tape 2; +1 for every a; -1 for every b

Time: O(n) – faster than best 1-tape TM for L

(Analysis is a bit subtle. "+1/-1" take log n steps in worst case, but "carries/borrows" usually don't propagate very far. Can prove *amortized* cost of +1/-1 is only O(1) per operation.)

One Conclusion: "Time" is somewhat technology-sensitive

(In fact, gap between 1 tape and 2 is quadratic: $\{ww | w \in \Sigma^*\}$)

“Tapes are Lame”

Obviously, “real” computers have essentially constant-time access to *any* bit of memory, not sequential access as on a tape

Fast “random access” will allow faster algorithms for many problems, so time on a TM may seem a poor surrogate for time on real computers

How poor?

A Model of a “Real Computer”

“Random Access Machines” (RAMs)

- Memory is an array
- Unit time access to any word
- Basic, unit time ops like +, -, *, /, test-if-zero,…
- Programs

For comparison to TMs, perhaps have read-only “input tape” or other string-oriented input convention and special “accept/reject” operations. Program typically not in memory (but could be)

TM-time(T) $\subseteq$ RAM-time(T)
RAM-time(T) $\subseteq$ TM-time(T^3)

Proof: look at your homework #1 and see how long your simulations took.

TM by RAM is quick

RAM by TM is slower, but cubic is conservative. In time T, the RAM can touch at most T memory words, each word holds at most T bits, it takes time at most T^2 to slog through tape to fetch/store a word, etc.

A Church-Turing thesis for "time"?

Church-Turing thesis: all "reasonable" models of computation are equivalent – i.e. all give the same set of decidable problems

"Extended" Church Turing thesis: All "reasonable" models of computation are equivalent *up to a polynomial difference in time complexity*

E.g. from above, this is true of deterministic singe- and multi-tape TMs and RAMs

More on what "reasonable" means later…

The class P

Definition:

$P = \bigcup_{k \geq 1} TIME(n^k)$

I.e., the set of (decision) problems solvable by computers in *polynomial time*. I.e., $L \in P$ iff there is an algorithm deciding L in time $T(n) = O(n^k)$ for some fixed k (i.e., k is independent of the input).

Examples: sorting, shortest path, MST, connectivity, …

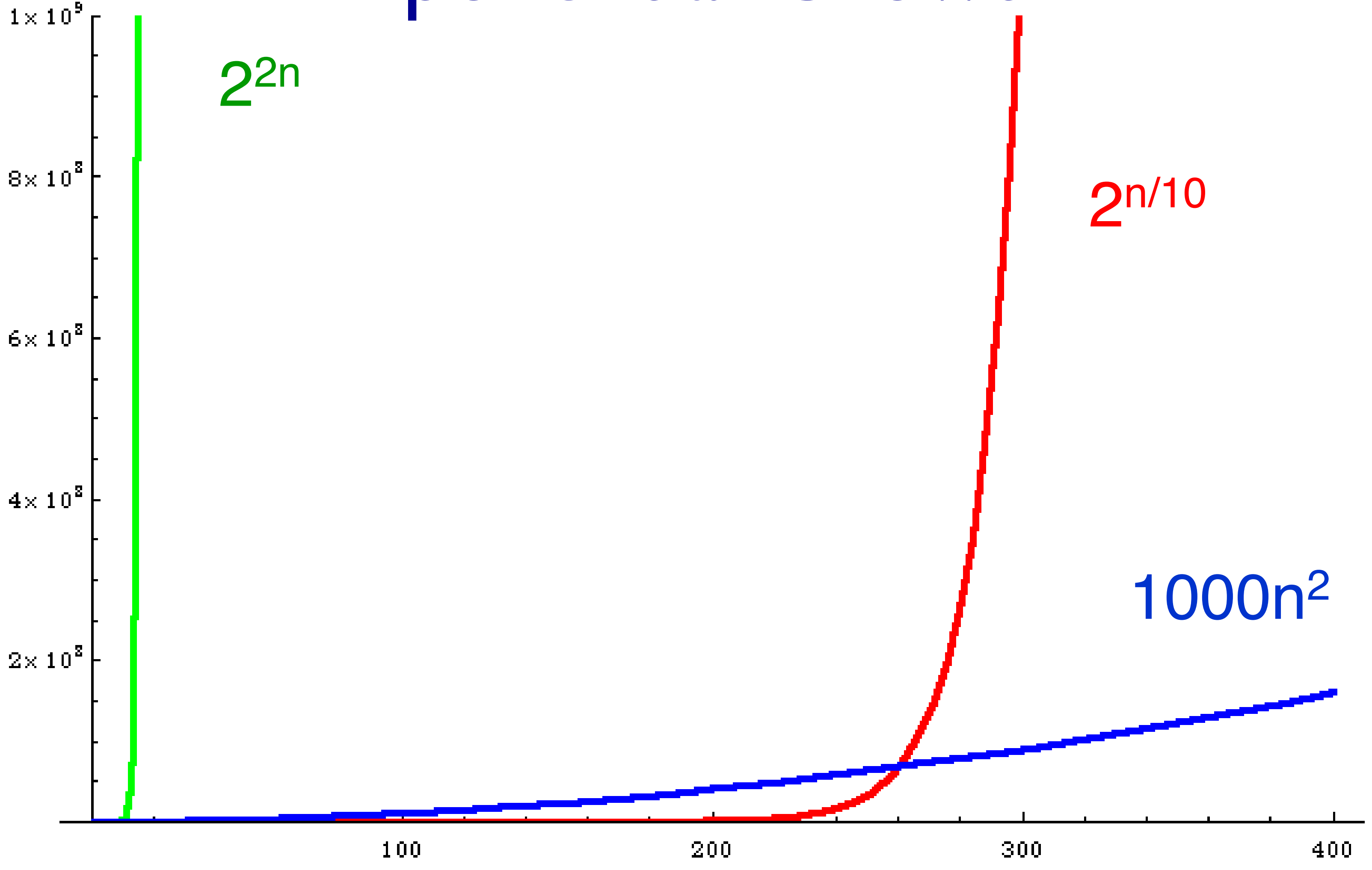
Why "Polynomial"?

Point is not that n^{2000} is a nice time bound, or that the differences among n and 2n and n^2 are negligible.

Rather, simple theoretical tools may not easily capture such differences, whereas exponentials are qualitatively different from polynomials and may be amenable to theoretical analysis.

- "My problem is in P" is a starting point for a more detailed analysis
- "My problem is not in P" may suggest that you need to shift to a more tractable variant

Polynomial vs Exponential Growth



Another view of Poly vs Exp

Next year's computer will be 2x faster. If I can solve problem of size n_0 today, how large a problem can I solve in the same time next year?

Complexity	Increase	E.g. $T=10^{12}$	
$O(n)$	$n_0 \rightarrow 2n_0$	10^{12}	2×10^{12}
$O(n^2)$	$n_0 \rightarrow \sqrt{2}\, n_0$	10^{6}	1.4×10^{6}
$O(n^3)$	$n_0 \rightarrow \sqrt[3]{2}\, n_0$	10^{4}	1.25×10^{4}
$2^{n/10}$	$n_0 \rightarrow n_0+10$	400	410
2^{n}	$n_0 \rightarrow n_0+1$	40	41