

Harvard CS 121 and CSCI E-121

Lecture 20: Polynomial Time

Harry Lewis

November 12, 20123

Review of Asymptotic Notation

For $f, g : \mathcal{N} \rightarrow \mathcal{R}^+$

- $f = O(g)$: $\exists c > 0$ s.t. $f(n) \leq c \cdot g(n)$ for all sufficiently large n .
- $f = \Omega(g)$: $g = O(f)$
- $f = \Theta(g)$: $f = O(g)$ and $g = O(f)$
- $f = o(g)$: $\forall c > 0$ we have $f(n) \leq c \cdot g(n)$ for all sufficiently large n . Equivalently, $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$.
- $f = \omega(g)$: $g = o(f)$. Equivalently, $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$.

Which of the following implies the other?

- $\lim_{n \rightarrow \infty} f(n)/g(n) = t$ for some $0 < t < \infty$.
- $f = \Theta(g)$.

TIME and Big-O

- Recall: Let $t : \mathcal{N} \rightarrow \mathcal{R}^+$. Then $\text{TIME}(t)$ is the class of languages L that can be decided by some multitape TM with running time $\leq t(n)$ for inputs of size n .
- “Table lookup” shows that using more states we can get $t(n) = n$ for finitely many n .
- Linear Speedup Theorem shows that using more states and a larger tape alphabet we can reduce $t(n)$ by a constant factor (as long as $t(n)$ is not too small).
- So $\text{TIME}(O(t(n))) = \text{TIME}(t(n))$ for $t(n)$ not too small (e.g. $t(n) \geq 1.1n$).
- What about more tapes?

Time-bounded Simulations

Q: How quickly can a 1-tape TM M_2 simulate a multitape TM M_1 ?

- If M_1 uses $f(n)$ time, then it uses $\leq f(n)$ tape cells
- M_2 simulates one step of M_1 by a complete sweep of its tape. This takes $\mathcal{O}(f(n))$ steps.

$\therefore M_2$ uses $\leq f(n) \cdot \mathcal{O}(f(n)) = \mathcal{O}(f^2(n))$ steps in all.

So $L \in \text{TIME}_{\text{multitape TM}}(f) \Rightarrow L \in \text{TIME}_{\text{1-tape TM}}(\mathcal{O}(f^2))$

Similarly $\mathcal{O}(f^k)$ for

- 2-D Tapes
- Random Access TMs ...

Basic thesis of complexity theory

Extended Church-Turing Thesis: Every “reasonable” model of computation can be simulated on a Turing machine with only a polynomial slowdown.

Counterexamples?

- Randomized computation.
- Parallel computation.
- Analog computers.
- DNA computers.
- Quantum computers.

Should qualify thesis with “sequential and deterministic”.

Polynomial Time

- **Def:** Let $P = \bigcup_p \text{TIME}(p)$, where p is a polynomial
$$= \bigcup_{k \geq 0} \text{TIME}(n^k)$$
- also known as PTIME or \mathcal{P}
- Coarse approximation to “efficient”:

Model Independence of P

Although P is defined in terms of TM time, **P is a stable class, independent of the computational model.**

(Provided the model is reasonable.)

Justification:

- If A and B are different models of computation, $L \in \text{TIME}_A(p_1(n))$, and B can simulate a time t computation of A in time $p_2(t)$, then $L \in \text{TIME}_B(p_2(p_1(n)))$.
- Polynomials are closed under composition, e.g.
 $f(n) = n^2, g(n) = n^3 + 1 \Rightarrow f(g(n)) = (n^3 + 1)^2 = n^6 + 2n^3 + 1$.

How much does representation matter?

- How big is the representation of an n -node directed graph?
 - ... as a list of edges?
 - ... as an adjacency matrix?
- How big is the representation of a natural number n ?
 - ... in binary?
 - ... in decimal?
 - ... in unary?

Describing & Analyzing Polynomial-Time Algorithms

- Due to Extended Church-Turing Thesis, we can use high-level descriptions.
- Freely use algorithms we've seen as subroutines, if we (or you) have analyzed their running time.
- Bound the total number of high-level steps (including # of loop iterations), and the running time of each step.
- Be careful about the size of data.
- “ $\text{poly}(n)$ executions of $\text{poly}(n)$ -time algorithms on $\text{poly}(n)$ -sized inputs takes time $\text{poly}(n)$ ”

A bogus polynomial-time algorithm

Consider the following algorithm on input an n -bit number z :

- Repeat n times: let $z \leftarrow z \times z$ (using grade-school multiplication algorithm)

“Proof” that this algorithm is polynomial time:

- The loop has n iterations.
- Each time we multiply, which takes time $O(n^2)$.
- Total time = $n \cdot O(n^2) = O(n^3)$.

Where is the error?

Another way of looking at P

- Multiplicative increases in time or computing power yield multiplicative increases in the size of problems that can be solved
- If L is in P, then there is a constant factor $k > 1$ such that
 - If you can solve problems of size s within a given amount of time
 - and you are given a computer that runs twice as fast, then
 - you can solve problems of size $k \cdot s$ on the new machine in the same amount of time.
- E.g. if L is decidable in $O(n^d)$ time, then with twice as much time you can solve problems $2^{1/d}$ as large

Exponential time

- $E = \cup_{c>0} \text{TIME}(c^n)$
- For problems in E , a multiplicative increase in computing power yields only an *additive* increase in the size of problems that can be solved.
- If L is in E , then there is a constant k such that
 - If you can solve problems of size s within a given amount of time
 - and you are given a computer that runs twice as fast, then
 - you can solve problems only of size $k + s$ on the new machine using the same amount of time.