BU CS 332 – Theory of Computation

Lecture 18:

- Time Complexity
- Complexity Class P

Reading:

Sipser Ch 7.1-7.2

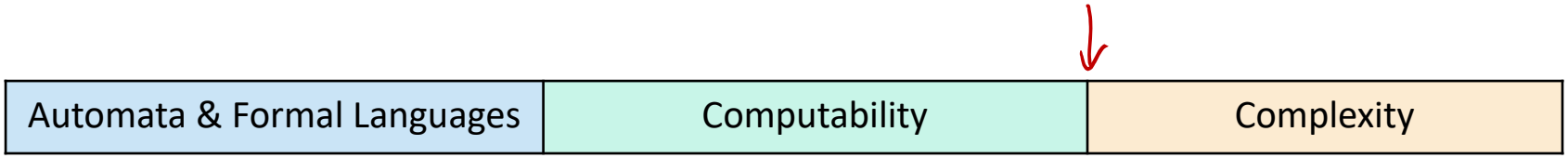
HW 7 due Monday April 13

Midterm (hopefully) graded by Thursday

Mark Bun

April 6, 2020

Where we are in CS 332



Previous unit: Computability theory
What kinds of problems can / can't computers solve?

Final unit: Complexity theory
What kinds of problems can / can't computers solve under constraints on their computational resources?

Restrict attention to "efficient algorithms"
i.e. algs. using limited time or space

First topic: Time complexity

Today: Answering the basic questions

1. How do we measure complexity? (as in CS 330)

2. Asymptotic notation (as in CS 330)

3. How robust is the TM model when we care about measuring complexity?

4. How do we mathematically capture our intuitive notion of "efficient algorithms"?

Running time analysis

For now: TM = basic single-tape TM

Time complexity of a TM (algorithm) = maximum number of steps it takes on a worst-case input

Running time is a function of n = input length

input lengths ↓ running time ↓

Formally: Let $f : \mathbb{N} \to \mathbb{N}$. A TM M runs in time $f(n)$ if on every input $w \in \Sigma^n$, M halts on w within at most $f(n)$ steps

(Implicit: M is a decider)

- Focus on worst-case running time: Upper bound of $f(n)$ must hold for all inputs of length n
- Exact running time $f(n)$ does not translate well between computational models / real computers. Instead focus on asymptotic complexity.

Example

How much time does it take for a basic single-tape TM to decide $A = \{0^m 1^m \mid m \geq 0\}$?

Input length = $n = 2m$

Let's analyze one particular TM M:

0 0 0 X X X

Iteration 1 (step 2)
Iteration 2 (step 2)
Iteration 3

M = "On input w:

$O(n)$ → 1. Scan input and reject if not of the form 0^*1^*

Execute this loop $O(n)$ times

2. While input contains both 0's and 1's: (check: $O(n)$ time)

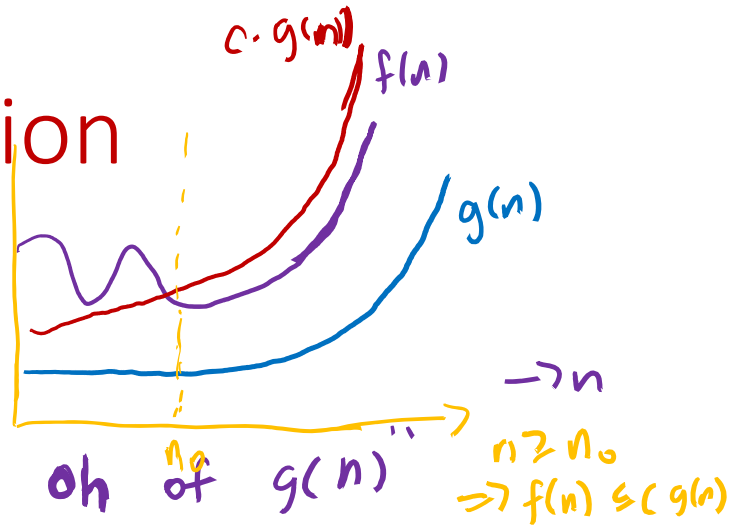
   Cross off one 0 and one 1 (crossing off: $O(n)$ time)

3. Accept if no 0's and no 1's left. Otherwise, reject."

Final check in $O(n)$

Total runtime: $\underbrace{O(n)}_{\text{step 1}} + O(n)\underbrace{[O(n)+O(n)]}_{\text{step 2}} + \underbrace{O(n)}_{\text{step 3}} = O(n^2)$

Review of asymptotic notation



O-notation (upper bounds)

$f, g : \mathbb{N} \to \mathbb{N}$

$f(n) = O(g(n))$ means: "$f(n)$ is oh of $g(n)$"

There exist constants $c > 0, n_0 > 0$ such that

$f(n) \leq cg(n)$ for every $n \geq n_0$

$2n^2 = O(n^3)$ (true) but also:

Example: $2n^2 = O(n^3)$ $(c = 2, n_0 = 0)$

Justification: $n \geq 0 \Rightarrow 2n^2 \leq 2n^3$

Caution: = does not mean "equals"

Not reflexive:

$f(n) = O(g(n))$ does not mean $g(n) = O(f(n))$

Example: $f(n) = 2n^2,\ g(n) = n^3$ $2n^2 = O(n^3)$

NOT the case that $n^3 = O(2n^2)$

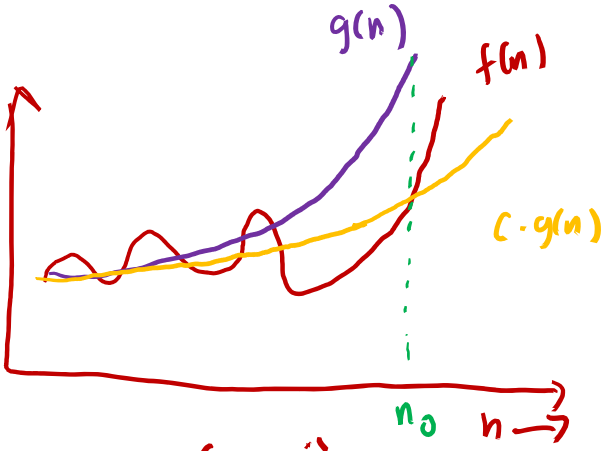
$O(g(n))$ = set of functions f w/ rate of growth $\leq$ that of g

Alternative (better) notation: $f(n) \in O(g(n))$

Examples

- $10^6\, n^3 + \ 2n^2 \ - n \ + 10 \ =$ $O(n^3)$
- $\sqrt{n} \ + \ \log n \ =$ $O(\sqrt{n})$
- $n\ (\log n + \sqrt{n}) \ =$ $O(n\sqrt{n}) = O(n^{1.5})$
- $n \ =$ $O(n)$ [but also $O(n^2)$, $O(2^n)$, ...]

Review of asymptotic notation



Ω-notation (lower bounds)

$f(n) = \Omega(g(n))$ means: "f is (big) omega of g"

There exist constants $c > 0, n_0 > 0$ such that

$f(n) \geq cg(n)$ for every $n \geq n_0$

Example: $\sqrt{n} = \Omega(\log n)$ $(c = 1, n_0 = 16)$

Justification: $n \geq 16 \Rightarrow \sqrt{n} \geq \log n$ [log = $\log_2$]

When should we use O vs. Ω?

Upper bounds: Use O

"The merge-sort algorithm uses at most $O(n \log n)$ comparisons in the worst case"

Lower bounds: Use Ω

"Every comparison-based sorting algorithm requires at least $\Omega(n \log n)$ comparisons in the worst case"

Review of asymptotic notation

Θ-notation (tight bounds)

$f(n) = \Theta(g(n))$ means: "f is theta of g"

$f(n) = O(g(n))$ AND $f(n) = \Omega(g(n))$

Example: $\frac{1}{2}n^2 - 1000n = \Theta(n^2)$

Generally, polynomials are easy:

$$a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d)$$

$a_d > 0$

Little-oh and little-omega

O-notation and Ω-notation are like $\leq$ and $\geq$;
o-notation and ω-notation are like $<$ and $>$

Asymptotic rate of growth of f is strictly slower than that of g

$f(n) = o(g(n))$ means: "f is little oh of g"

For every constant $c > 0$, there exists $n_0 > 0$ such that

$f(n) \leq c\,g(n)$ for every $n \geq n_0$

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

little omega (ω): reverse this to $>$

Example: $2n^2 = o(n^3)$ ($n_0 = 2/c$)

Justification: $n \geq n_0 = \frac{2}{c} \Rightarrow 2n^2 \leq c n^3$

$\lim_{n \to \infty} \frac{2n^2}{n^3} = \lim_{n \to \infty} \frac{2}{n} = 0$

A handy-dandy chart

Notation	… means …	Think…	Example	$\lim_{n\leftarrow\infty} \frac{f(n)}{g(n)}$
$f(n)=O(g(n))$	$\exists\ c>0, n_0>0, \forall\ n > n_0 :$ $f(n) < cg(n)$	Upper bound	$100n^2$ $= O(n^3)$	If it exists, it is $< \infty$
$f(n)=\Omega(g(n))$	$\exists c>0, n_0>0, \forall\ n > n_0 :$ $cg(n) < f(n)$	Lower bound	2^n $= \Omega(n^{100})$	If it exists, it is > 0
$f(n)=\Theta(g(n))$	both of the above: $f=\Omega(g)$ **and** $f = O(g)$	Tight bound	$\log(n!)$ $= \Theta(n \log n)$	If it exists, it is > 0 and $< \infty$
$f(n)=o(g(n))$	$\forall\ c>0, \exists\ n_0>0, \forall\ n > n_0 :$ $f(n) < cg(n)$	Strict upper bound	$n^2 = o(2^n)$	Limit exists, $=0$
$f(n)=\omega(g(n))$	$\forall\ c>0, \exists n_0>0, \forall\ n > n_0 :$ $cg(n) < f(n)$	Strict lower bound	n^2 $= \omega(\log n)$	Limit exists, $=\infty$

Asymptotic notation within expressions

Asymptotic notation within an expression is shorthand for an unspecified function satisfying the statement

Examples:

- $n^{O(1)}$ means n^c for some constant $c > 0$
- $n^2 + \Omega(n)$ means $n^2 + g(n)$ for some $g(n) = \Omega(n)$
- $\big(1 + o(1)\big)n$ means $(1 + \epsilon(n))n$ for some $\epsilon(n) \to 0$ as $n \to \infty$

FAABs: Frequently asked asymptotic bounds

- Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$
- Logarithms. $\log_a n = \Theta(\log_b n)$ for all constants $a, b > 0$

For every $c > 0$, $\log n = o(n^c)$

- Exponentials. For all $b > 1$ and all $d > 0$, $n^d = o(b^n)$
- Factorial. $n! = n(n-1)\cdots 1$

By Stirling's formula,

$$n! = (\sqrt{2\pi n})\left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}$$

Means: $\exists\ g(n) = \Theta(n \log n)$ s.t. $n! = 2^{g(n)} \Rightarrow \log(n!) = g(n)$
$\Rightarrow \log(n!) = \Theta(n \log n)$

Time Complexity

Time complexity classes

Let $f : \mathbb{N} \rightarrow \mathbb{N}$

$\mathrm{TIME}(f(n))$ is a class (i.e., set) of languages:

"complexity class"

A language $A \in \mathrm{TIME}(f(n))$ if there exists a basic single-tape (deterministic) TM M that

1) Decides A, and
2) Runs in time $O(f(n))$

Example

$A = \{0^m 1^m \mid m \geq 0\}$

M = "On input w:

1. Scan input and reject if not of the form 0^*1^*
2. While input contains both 0's and 1's:

   Cross off one 0 and one 1
3. Accept if no 0's and no 1's left. Otherwise, reject."

- M runs in time $O(n^2)$ $\Longrightarrow$ $A \in TIME(n^2)$
- Is there a faster algorithm?

Example

Iteration 1
Iteration 2
Iteration 3

$A = \{0^m 1^m \mid m \geq 0\}$

M' = "On input w:

1. Scan input and reject if not of the form 0^*1^*
2. While input contains both 0's and 1's:
   - Reject if the total number of 0's and 1's remaining is odd
   - Cross off every other 0 and every other 1
3. Accept if no 0's and no 1's left. Otherwise, reject."

execute this loop $O(\log n)$

cuts input size by a factor of 2

- Running time of M': $O(n) + O(\log n) \cdot O(n) + O(n) = O(n \log n)$
  step 1, step 2, step 3

  $\Rightarrow A \in TIME(n \log n)$
- Is there a faster algorithm?

Example

Running time of M': $O(n \log n)$

Theorem (Sipser, Problem 7.49): If L can be decided in $o(n \log n)$ time on a 1-tape TM, then L is regular

A not regular $\Rightarrow$ M' is (asymptotically) the fastest algorithm deciding A

Does it matter that we're using the 1-tape model for this result?

It matters: 2-tape TMs can decide A faster

M'' = "On input w:

1. Scan input and reject if not of the form 0^*1^* $O(n)$
2. Copy 0's to tape 2 $O(n)$
3. Scan tape 1. For each 1 read, cross off a 0 on tape 2 $O(n)$
4. If 0's on tape 2 finish at same time as 1's on tape 1, accept. Otherwise, reject." $O(1)$

Faster than what's possible on 1-tape

Analysis: A is decided in time $O(n)$ on a 2-tape TM

Moral of the story (part 1): Unlike decidability, time complexity depends on the TM model

How *much* does the model matter?

Theorem: Let $t(n) \geq n$ be a function. Every multi-tape TM running in time $t(n)$ has an equivalent single-tape TM running in time $O(t(n)^2)$

[converting from multi-tape to single-tape incurs ≤ quadratic slowdown]

Proof idea:

We already saw how to simulate a multi-tape TM with a single-tape TM

Need a runtime analysis of this construction

Moral of the story (part 2): Time complexity doesn't depend too much on the TM model (as long as it's deterministic, sequential)

Simulating Multiple Tapes

Implementation-Level Description

Simulating k tape TM

On input $w = w_1 w_2 \ldots w_n$

1. Format tape into $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$ $O(n + k)$
2. For each move of M: Outer loop: executed $\leq t(n)$ times

   Scan left-to-right, finding current symbols
   Scan left-to-right, writing new symbols,
   Scan left-to-right, moving each tape head
   } $O(k \cdot t(n))$ time

   If a tape head goes off the right end, insert blank
   If a tape head goes off left end, move back right
   } $O(1)$

How *much* does the model matter?

Theorem: Let $t(n) \geq n$ be a function. Every multi-tape TM running in time $t(n)$ has an equivalent single-tape TM running in time $O(t(n)^2)$

Proof: Time analysis of simulation

- Time to initialize (i.e., format tape): $O(n + k)$
- Time to simulate one step of multi-tape TM: $O(t(n))$

Reason: $\leq t(n)$ cells in use if runtime of multi-tape TM is $t(n)$

- Number of steps to simulate: $t(n)$

=> Total time: $O(n+k) + O(k \cdot t(n)) \cdot O(t(n))$

$= O(t(n)^2)$ [for constant k]

Extended Church-Turing Thesis

Every "reasonable"* model of computation can be simulated by a basic, single-tape TM with only a **polynomial** slowdown.

E.g., doubly infinite TMs, multi-tape TMs, RAM TMs

Does not include nondeterministic TMs (not reasonable)

Possible counterexamples? Randomized computation, parallel computation, DNA computing, quantum computation

(more believable: * deterministic, sequential)

Complexity Class $\mathbf{P}$

Complexity class P

Definition: P is the class of languages decidable in polynomial time on a basic single-tape (deterministic) TM

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

$= \mathrm{TIME}(n) \cup \mathrm{TIME}(n^2) \cup \mathrm{TIME}(n^3) \cup \ldots$

- Class doesn't change if we substitute in another reasonable deterministic model (Extended Church-Turing)
- Cobham-Edmonds Thesis: Captures ("Roughly") class of problems that are feasible to solve on computers

Examples of languages in P

- $PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a directed path from } s \text{ to } t\}$
- $A_{\text{DFA}} = \{\langle D, w\rangle \mid D \text{ is a DFA that accepts input } w\}$
- $RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}$
- $PRIMES = \{\langle x\rangle \mid x \text{ is prime}\}$

2006 Gödel Prize citation



The 2006 Gödel Prize for outstanding articles in theoretical computer science is awarded to Manindra Agrawal, Neeraj Kayal, and Nitin Saxena for their paper "PRIMES is in P."

In August 2002 one of the most ancient computational problems was finally solved….

- Every context-free language (section tomorrow)

Dynamic Programming