BU CS 332 – Theory of Computation

https://forms.gle/YQaFs3aX9dSVod4e6

Lecture 18:

- Asymptotic Notation
- Time/Space Complexity
- Complexity Class P

Reading:

Sipser Ch 7.1, 7.2, 8.0

– OH today 5:30-6:30
CDS 1021

– Pause for eclipse ~3:25

Mark Bun

April 8, 2024

Where we are in CS 332

Automata	Computability	Complexity

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

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

Time and space complexity

Today: Start 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"?

Time and space complexity

Time complexity of a TM = Running time of an algorithm

= Max number of steps as a function of input length n

of (read, write, move) instructions TM executes

Space complexity of a TM = Memory usage of algorithm

= Max number of tape cells as a function of input length n

Example

In how much time/space can a basic single-tape TM decide $A = \{0^m 1^m \mid m \geq 0\}$?

Ex: Input 000 111

1. Linear scan of input to see all 0's before all 1's

2. a. ~~0~~00 ~~1~~11

   b. ~~00~~0 ~~11~~1

   c. ~~000~~ ~~111~~

3. No 0's or 1's left, so accept.

One particular TM M deciding this language:

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."

Example

M = "On input w:

1. Scan input and reject if not of the form 0^*1^*] $O(n)$
2. While input contains both 0's and 1's: ← Runs $O(n)$ times

   Cross off one 0 and one 1 ← $O(n)$ time to find and cross off
3. Accept if no 0's and no 1's left. Otherwise, reject."

What is the time complexity of M?

a) $O(1)$ [constant time]

b) $O(n)$ [linear time]

c) $O(n^2)$ [quadratic time]

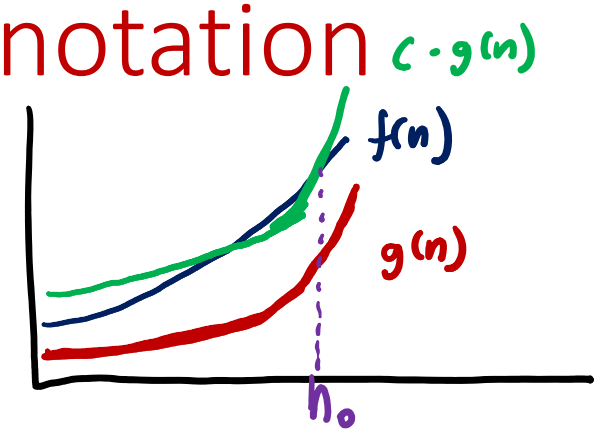
d) $O(n^3)$ [cubic time]

$$\underbrace{O(n)}_{\text{phase 1}} + \underbrace{O(n)}_{\text{times through phase 2 loop}} \cdot \underbrace{O(n)}_{\text{each invocation of Phase 2}} = O(n^2)$$

What is the space complexity of M?

Space: $O(n)$ (part of the tape input is written on)

Review of asymptotic notation



O-notation (upper bounds)

$f(n) = O(g(n))$ means:

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

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

Example: $2n^2 + 12 = O(n^3)$ $(c = 3, n_0 = 4)$

Proof. Let $c = 3$, $n_0 = 4$

If $n \geq 4$ then $2n^2 + 12 \leq 2n^2 + n^2$

(why?. Because $n \geq 4 \Rightarrow n^2 \geq 16 \geq 12$)

$\leq 3n^2$

Properties of asymptotic notation:

Transitive:

$f(n) = O(g(n))$ and $g(n) = O(h(n))$ means $f(n) = O(h(n))$

Ex: $n = O(n^2)$ $\quad n^2 = O(n^3)$ $\quad \Rightarrow \quad n = O(n^3)$

Not reflexive:

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

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

$2n^2 = O(n^3)$ but n^3 is not $O(n^2)$

Alternative (better) notation: $f(n) \in O(g(n)) = \{ h(n) \mid \exists\, c, n_0 \text{ s.t. } g(n) \le c \cdot h(n) \; \forall\, n \geq n_0 \}$

Examples

- $10^6 n^3 + 2n^2 - n + 10 = O(n^4)$ true, but not "optimal"

  $= O(n^3)$

- $\sqrt{n} + \underbrace{\log n}_{}= O(\sqrt{n})$

  logs grow slower than all polynomials

- $n(\log n + \sqrt{n}) = n \log n + \underbrace{n\sqrt{n}}_{\text{dominant}} = O(n\sqrt{n}) = O(n^{3/2})$.

Little-oh

If O-notation is like $\leq$, then o-notation is like $<$

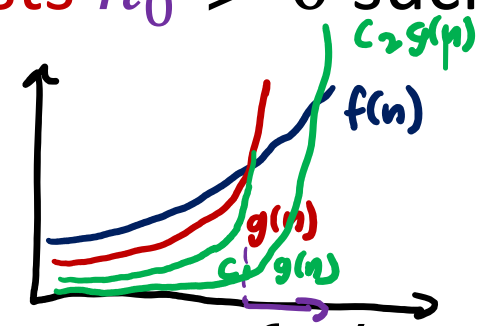
big-Oh: $\exists\, c\ \exists\, n_0$ s.t....

Little-oh: $\forall\, c\ \exists\, n_0$ s.t....

$f(n) = o(g(n))$ means:

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

$$f(n) \leq c g(n) \text{ for every } n \geq n_0$$



$\Leftrightarrow \forall\, c>0\ \exists\, n_0$ s.t. $\frac{f(n)}{g(n)} \leq c \quad \forall\, n \geq n_0$

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

Example: $\quad 2n^2 + 12 \ = o(n^3) \qquad (n_0 = \max\{4/c, 3\})$

Proof 1: Let $c>0$ be arbitrary

Choose $n_0 = \max\{\frac{4}{c}, 3\}$, $n \geq n_0$

Then $\underline{2n^2+12} \leq 4n^2$ (as $n \geq 3$ $\Rightarrow 12 \leq 2n^2$)

$\leq (cn)\cdot n^2$ (as $n \geq \frac{4}{c}$)

$= \underline{cn^3}$

Proof 2:

$$\lim_{n\to\infty} \frac{2n^2+12}{n^3} = \lim_{n\to\infty} \frac{2}{n} + \frac{12}{n^3}$$

$= 0.$

True facts about asymptotic expressions

Which of the following statements is true about the function $f(n) = 2^n$?

$f(n) = O(g(n))$ means

$f(n) \le \text{const} \cdot g(n)$

$f(n) = o(g(n))$ means

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

a) $f(n) = O(3^n)$ ✓

$2^n \le 3^n \quad \forall n$

b) $f(n) = o(3^n)$ ✓

$\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\underbrace{\frac{2}{3}}_{\le 1}\right)^n = 0$

c) $f(n) = O(n^2)$ ✗

Exponentials grow faster than polynomials

d) $n^2 = O(f(n))$ ✓

Asymptotic notation within expressions

Asymptotic notation within an expression is shorthand for "there exists a function satisfying the statement"

Examples:

- $n^{O(1)}$ means "$\exists$ f(n) s.t. $f(n) = O(1)$ s.t. this is $n^{f(n)}$"

  $\iff$ "n^c for some constant c" $\iff$ "some polynomial"

  e.g. $n^{100} = n^{O(1)}$ is a true statement because $100 = O(1)$.

- $n^2 + O(n)$ means $\exists$ f(n) s.t.. $n^2 + f(n)$

  $n^2 + 10000n = n^2 + O(n)$ because $10000n = O(n)$

  True: $n^2 + O(n) = O(n^2)$ $\quad$ $100n^2 = O(n^2)$, but $100n^2 \neq n^2 + O(n)$.

- $\big(1 + o(1)\big)n$ = $n + o(1)\cdot n$ means $n +$ something sublinear

  $\iff n + f(n)$ for $\frac{f(n)}{n} \to 0$.

FAABs: Frequently asked asymptotic bounds

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

$\log_a n = \frac{\log_b n}{\log_b a} = \text{const} \cdot \log_b n$ whenever a, b are constants.

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

$\frac{n^d}{b^n} \to 0$

- **Exponentials.** For all $b > 1$ and all $d > 0$, $n^d = o(b^n)$

$n^{999999} = o((1.0000001)^n)$

- **Factorial.** $n! = n(n-1)\cdots 1$ $\approx n^n$

By Stirling's formula,

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

$\approx n^n = (\underbrace{2^{\log n}}_{=n})^n = 2^{n \log n}$

Time and Space Complexity

Running time analysis

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

Formally: Let $f : \overset{\text{input length}}{\mathbb{N}} \to \overset{\text{\# of computational steps}}{\mathbb{N}}$. 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

- 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.

Time complexity classes

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

$\mathrm{TIME}(f(n))$ is a set ("class") of languages:

"complexity class"

= set of problems solvable in $O(f(n))$ time

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 $O(f(n))$

Ex: $\{0^m 1^m \mid m \geq 0\} \in \mathrm{TIME}(n^2)$

because there is an $O(n^2)$-time algorithm solving this problem.