BU CS 332 – Theory of Computation

https://forms.gle/MabPvkC4K9jNHD257

Lecture 18:

- Asymptotic Notation
- Time/Space Complexity

Reading:

Sipser Ch 7.1, 8.0

Mark Bun

November 11, 2021

Where we are in CS 332

Automata	Computability	Complexity

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?

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 <u>function</u> of input length n

Space complexity of a TM = Memory usage of algorithm

= Max number of tape cells as a <u>function</u> of input length n

Example

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

Let's analyze one particular TM M:

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."

Input: $w = \overbrace{000}^{n}111$

1. In one linear pass, check $w \in L(0^*1^*)$
2. First loop: ø00 y11
   Second loop: øø0 yy1
   Third loop: øøø yyy
3. Accept

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

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]

Time complexity:

$\underbrace{O(n)}_{1} + \underbrace{O(n) \cdot O(n)}_{2} + \underbrace{O(n)}_{3}$

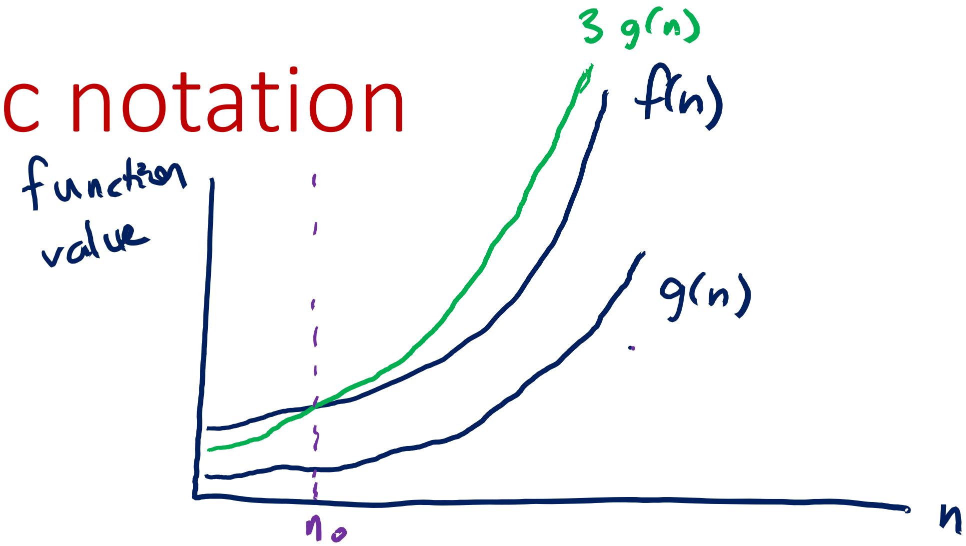
$= O(n^2)$

What is the space complexity of M?

n

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: $\overbrace{2n^2 + 12}^{f(n)} = O(\overbrace{n^3}^{g(n)})$ $(c = 3, n_0 = 4)$

If $n \geq n_0 = 4$

$f(n) = 2n^2 + 12 \leq 2n^2 + n^2 \leq 3n^2 \leq 3n^3 = 3g(n)$

$n \geq 4 \Rightarrow n^2 \geq 12$; $n \geq 1$

Properties of asymptotic notation:

Transitive:

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

Ex: $n^2 = O(n^3)$ and $n^3 = O(n^4)$ => $n^2 = O(n^4)$

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 $O(2n^2)$

Alternative (better) notation: $f(n) \in O(g(n)) = \{h(n) \mid \exists c\ \exists n_0\ h(n) \le c\,g(n)\ \forall n \ge n_0\}$

Examples

- $10^6 n^3 + 2n^2 - n + 10 = O(n^3)$

  [also $O(n^4)$, $O(n^{27})$, $O(2^n)$, ...]

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

  ↑ $= n^{1/2}$

  also $O(\sqrt{n})$

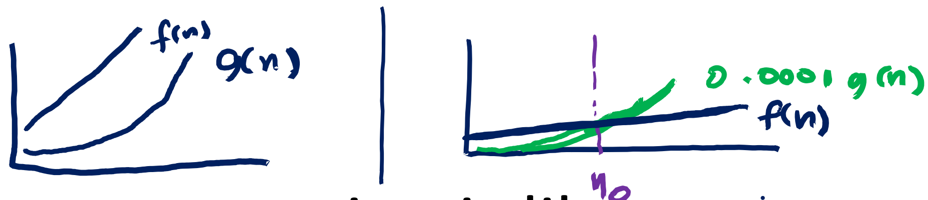
  In general: logs dominated by polynomials

  $\log n = O(n^c)$ $\forall$ constants c

- $n(\log n + \sqrt{n}) = O(n^{1.5})$ (i.e. $O(n\sqrt{n})$)

- $n = O(n)$, $O(n^2)$, $O(n^{1.43})$, $O(2^n)$, ...

Little-oh



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

Big Oh: $\exists c > 0 \ \exists n_0 > 0 \ldots$ | Little oh: $\forall c > 0 \ \exists n_0 > 0 \ldots$

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

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

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

$\Leftrightarrow \forall c \ \exists n_0 \ \frac{f(n)}{g(n)} \leq c \ \forall n \geq n_0 \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

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

Proof 1: Let $c > 0$ be arbitrary.

Choose $n_0 = \max\{4/c, 3\}$

Let $n \geq n_0$. Then

$2n^2 + 12 \leq 2n^2 + 2n^2 = 4n^2 \leq (cn) \cdot n^2 = cn^3$

($n \geq n_0 \geq 3$; $n \geq n_0 \geq \frac{4}{c}$)

Proof 2:

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

$= \lim_{n \to \infty} \frac{2}{n} + \frac{12}{n^3} = 0.$

True facts about asymptotic expressions

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

a) $f(n) = O(3^n)$ ✓

In general: $f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$

⇑

b) $f(n) = o(3^n)$ ✓

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

c) $f(n) = O(n^2)$ ✗

d) $n^2 = O(f(n))$ ✓

All polynomials grow slower than all exponentials

Asymptotic notation within expressions

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

Examples:

- $n^{O(1)}$ means: $n^{f(n)}$ for some $f(n) = O(1)$
  $\Leftrightarrow$ n^c for some constant c $\Leftrightarrow$ "some" polynomial.
- $n^2 + O(n)$ means: $n^2 + f(n)$ for some $f(n) = O(n)$
- $(1 + o(1))n$ means: $n + n \cdot f(n)$ where $f(n) = o(1)$
  $= n + n \cdot f(n)$ where $f(n) \to 0$ as $n \to \infty$

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 $a, b > 0$

because $\log_a n = \frac{\log_b n}{\log_b a}$ | When we write "$O(\log n)$" doesn't matter what base of log is.

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,

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

" $n^{O(n)} = 2^{\log n \cdot O(n)}$

Time and Space Complexity

Running time analysis

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

input length ↓ time, i.e. step bound ↙

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

- 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 $f : \mathbb{N} \rightarrow \mathbb{N}$ (represents a runtime upper bound)

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

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

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

Ex: $\text{TIME}(n^2) = \{A \mid A$ is a language that can be decided in time $O(n^2)\}$

Time class containment

Ex: n n^2

If $f(n) = O(g(n))$, then which of the following statements is always true?

a) $\mathrm{TIME}(f(n)) \subseteq \mathrm{TIME}(g(n))$

b) $\mathrm{TIME}(g(n)) \subseteq \mathrm{TIME}(f(n))$

c) $\mathrm{TIME}(f(n)) = \mathrm{TIME}(g(n))$

d) None of the above

not necessarily true

$\mathrm{TIME}(n) \neq \mathrm{TIME}(n^2)$

If $A \in \mathrm{TIME}(f(n))$

$\Rightarrow \exists$ a TM M deciding A in time $O(f(n))$

$\Rightarrow \exists$ a TM M deciding A in time $O(g(n))$

$\Rightarrow A \in \mathrm{TIME}(g(n))$

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 $O(n^2)$ $\Rightarrow$ $A \in TIME(n^2)$
- Is there a faster algorithm?

Example

Ex: Input 0000 1111

First loop: 0000 1111

Second loop: 0000 1111

Third loop: 0000 1111

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

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: ← $O(\log n)$
   - Reject if the total number of 0's and 1's remaining is odd
   - Cross off every other 0 and every other 1 ← $O(n)$
3. Accept if no 0's and no 1's left. Otherwise, reject."

- Running time of M': $O(n \log n) \Rightarrow A \in TIME(n \log n)$

- 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

Contrapositive: L non-regular $\Rightarrow$
L cannot be decided in time faster than $n \log n$

$L = \{0^m 1^m \mid m \geq 0\}$ non-regular $\Rightarrow$ M' is "best" algorithm

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^*
2. Copy 0's to tape 2
3. Scan tape 1. For each 1 read, cross off a 0 on tape 2
4. If 0's on tape 2 finish at same time as 1's on tape 1, accept. Otherwise, reject."

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 $O(t(n)^2)$

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)