Class 21: Introducing Complexity

cs302: Theory of Computation
University of Virginia Computer Science

David Evans
http://www.cs.virginia.edu/evans

Exam 2

Good News

- 96% of you got 1a (a language is a set of strings) correct
- Most people got most credit for:
  - 2a (design a TM)
  - 2b (cyclical TM)
  - 3a (one-way simulation proof claiming equivalence)

Confusing News?

For question 1b (“Explain the essence of the *Church-Turing* Thesis in a way that would be understandable to a typical fifth grader”) more than half of you assumed a 5th grader knows what a Turing machine is (and about ¼ assumed they know Lambda calculus also!)

> Coming up with a good answer for this question with time pressure is tough. A good answer would either explain C-T thesis without needing TMs (using things a 5th grader already understands), or include an explanation of what a TM is. You can submit a new answer Tuesday. *Or*, find/make a 5th grader who understands TMs well enough to actually understand your answer.

Bad News

- Only 25/81 (>= 8 on 4b) and 24/81 (>= 8 on 4c) of you were able to get close to a convincing reduction proof.

> These were pretty tough questions, so many its actually good news that ~30% got them.

- But, to solve complexity problem, you will need to do tougher reduction proofs!

> Practicing more now would be a good idea!

Good/Bad News

- You have an opportunity to improve your score on Exam 2 by submitting improved answers to these questions
- Good news: I will provide some hints how to get started next.
- Bad news: Since I have provided hints, and you have as much time as you need, I expect very clear, convincing, correct answers.

4b

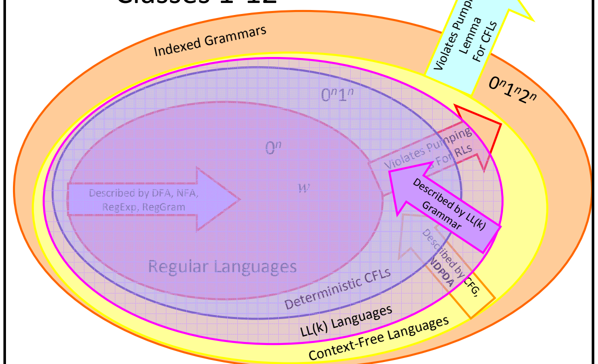
$NOTSUB_{TM}$ = { <A, B> | A and B are descriptions of TMs and there is some string which is accepted by A that is not accepted by B }

4c

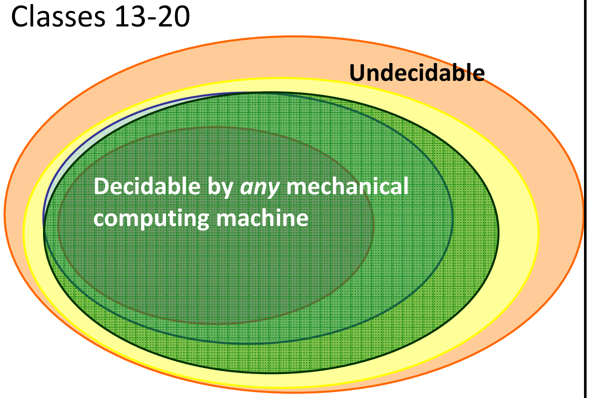
$L_{BusyBee}$ = {<M, w, k> | M describes a TM, k is the number of different FSM states M enters before halting on w }

Computability and Complexity

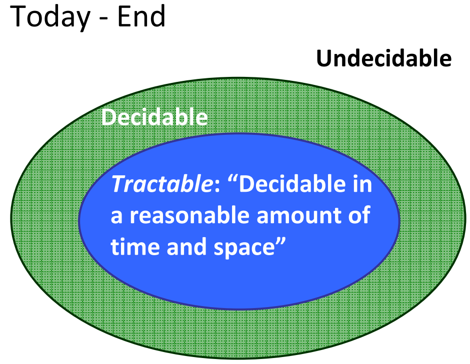
Classes 1-12



Classes 13-20



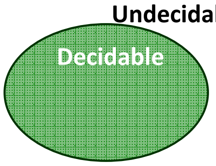
Today - End



Computability / Complexity

Computability

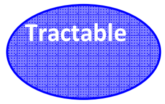
Undecidable



~1800s – 1960s
1900: Hilbert's Problems
1936: Turing's *Computable Numbers*
1957: Chomsky's *Syntactic Structures*
(Mostly) "Dead" field

Complexity

Intractable



1960s – 2150?
1960s: Hartmanis and Stearns: Complexity class
1971: Cook/Levin, Karp: *P=NP*?
1976: Knuth's O, Ω, Θ
Very Open and Alive

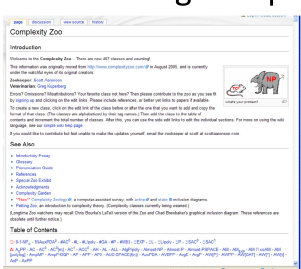
Complexity Classes

- **Computability** Classes: sets of problems (languages) that can be solved (decided/recognized) by a given machine
- **Complexity** Classes: sets of problems (languages) that can be solved (decided) by a given machine (usually a TM) within a limited amount of time or space

How many complexity classes are there?

Infinitely many! "Languages that can be decided by some TM using less than 37 steps" is a complexity class

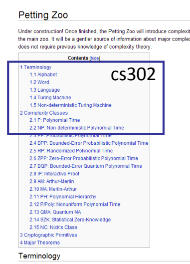
Interesting Complexity Classes



467 "interesting" complexity classes (and counting)!

http://qwiki.stanford.edu/wiki/Complexity_Zoo

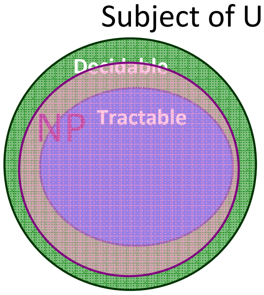
The "Petting Zoo"



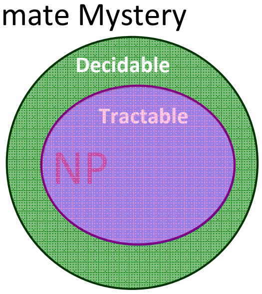
"Under construction! Once finished, the Petting Zoo will introduce complexity theory to newcomers unready for the terrifying and complex beasts lurking in the main zoo."

We will only get to the entrance of the "Petting Zoo". But, even here there are "terrifying and complex beasts lurking"!

The Most Terrifying Beast: Subject of Ultimate Mystery



Option 1: There are problems in Class NP that are not tractable



Option 2: All problems in Class NP are tractable

P = NP ?

- We need a couple more classes before explaining this (but will soon)
- This is an open question: no one knows the answer
  - If you can answer it, you will receive fame, fortune, and an A+ in cs302!
  - But, you should get some insight into what an answer would look like, and what it would mean

Order Notation

$$\boldsymbol{O}(f), \Omega(f), o(f), \boldsymbol{\Theta}(f)$$

Warning: you have probably seen some of these notations before in cs201 and cs216. What you learned about them there was probably (somewhat) useful but incorrect. (Note: if you learned them in cs150, then you learned them correctly.)

Order Notation

- $O(f)$, $\Omega(f)$, $o(f)$, $\Theta(f)$
- These notations define **sets of functions**
  - Generally: functions from positive integer to real
- We are interested in how the **size** of the **outputs** relates to the **size** of the **inputs**

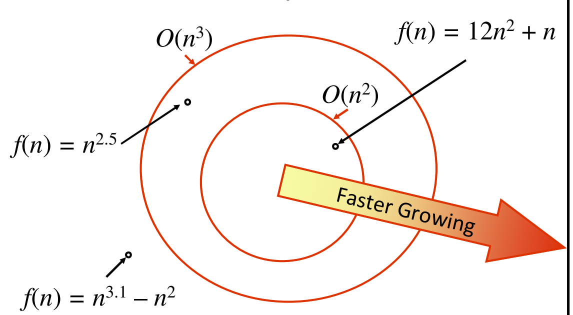
Big O

- Intuition: the set $O(f)$ is the set of functions that *grow* **no faster than** f
  - More formal definition coming soon
- Asymptotic growth rate
  - As input to f approaches infinity, how fast does value of f increase
  - Hence, only the fastest-growing term in f matters:

$O(12n^2 + n) \subset O(n^3)$

$O(n) \equiv O(63n + \log n - 423)$

Examples



Formal Definition

$f \in O(g)$ means:

There are *positive* constants c and n_0 such that

$$f(n) \leq cg(n)$$

for all values $n \geq n_0$.

O Examples

$f(n) \in O(g(n))$ means: there are positive constants c and n_0 such that $f(n) \leq cg(n)$ for all values $n \geq n_0$.

$x \in O(x^2)$?	Yes, $c = 1$, n_0=2 works fine.
$10x \in O(x)$?	Yes, $c = 11$, n_0=2 works fine.
~~$x^2 \in O(x)$?~~	No, no matter what c and n_0 we pick, $cx^2 > x$ for big enough x

Lower Bound: Ω (Omega)

$f(n)$ is $\Omega(g(n))$ means:

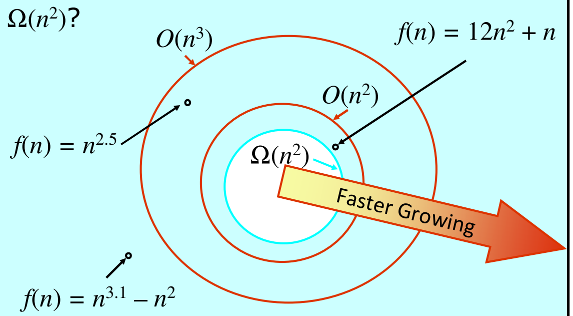
There are positive constants c and n_0 such that

$$f(n) \geq cg(n)$$

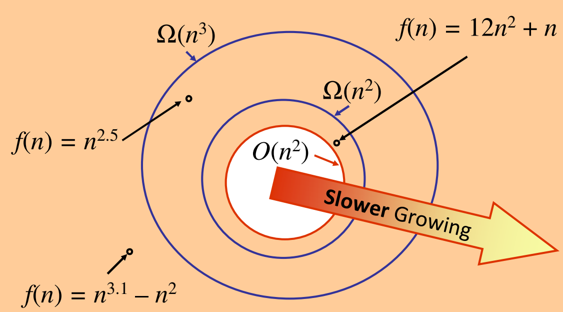
for all $n \geq n_0$.

Difference from O – this was $\leq$

Where is $\Omega(n^2)$?



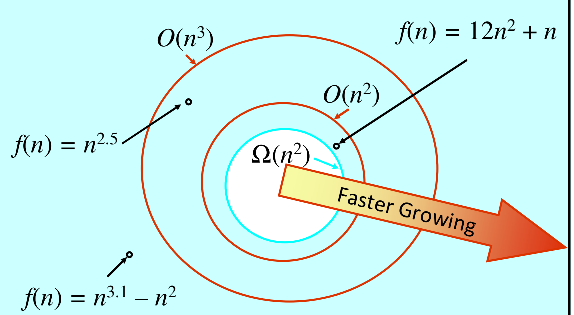
Inside-Out



Recap

- Big-O: the set $O(f)$ is the set of functions that *grow* **no faster than** f
  - There exist positive integers c, $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- Omega (Ω): the set $\Omega(f)$ is the set of functions that *grow* **no slower than** f
  - There exist positive integers c, $n_0 > 0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.

What else might be useful?



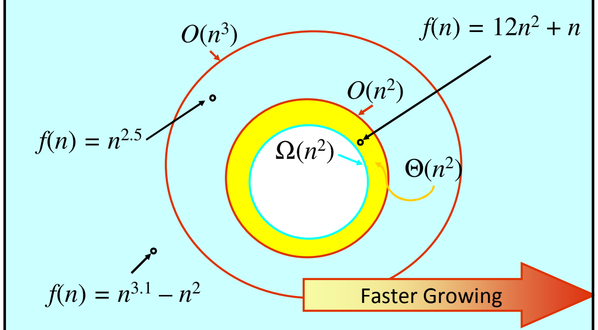
Theta ("Order of")

- Intuition: the set $\Theta(f)$ is the set of functions that *grow* **as fast as** f
- Definition: $f(n) \in \Theta(g(n))$ if and only if both:
  1. $f(n) \in O(g(n))$

  and 2. $f(n) \in \Omega(g(n))$
  - Note: we do not have to pick the same c and n_0 values for 1 and 2
- When we say, "f is order g" that means

  $f(n) \in \Theta(g(n))$

Tight Bound Theta (Θ)



Summary

- Big-O: there exist c, $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- Omega (Ω): there exist c, $n_0 > 0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.
- Theta (Θ): both O and Ω are true

When you were encouraged to use Big-*O* in cs201/cs216 to analyze the running time of algorithms, what should you have been using?

Algorithm Analysis

- In Big-*O* notation, what is the running time of algorithm *X*?

$$O(n^{n^{n^n}})$$

This is surely correct, at least for all algorithms you saw in cs201/cs216.

Should ask: In Theta notation, what is the running time of algorithm *X*?

Given an algorithm, should always be able to find a tight bound.

Complexity of Problems

So, why do we need O and Ω?

We care about the complexity of ***problems*** not **algorithms**. The complexity of a problem is the complexity of the best possible algorithm that solves the problem.

Revised exam answers are due at beginning of class Tuesday.