# CSE 105
## THEORY OF COMPUTATION

"Winter" 2018

http://cseweb.ucsd.edu/classes/wi18/cse105-ab/

# Today's learning goals

Sipser Ch 7

- Distinguish between computability and complexity
- Articulate motivation questions of complexity

- Section 7.1: time complexity, asymptotic upper bounds.
- Section 7.2: polynomial time, P
- Section 7.3: NP, polynomial verifiers, nondeterministic machines.

# Complexity theory

*Chapter 7*

In the "real world", computers (and Turing machines) don't have infinite tape, and we can't afford to wait unboundedly long for an answer.

"Decidable" isn't good enough – we want "Efficiently decidable"

# Not just decidable …

- For a given **algorithm** working on a given **input**, how long do we need to wait for an answer?  How does the running time depend on the input in the worst-case? average-case? Expect to have to spend more time on larger inputs.

# Measuring time

applications of transition function

- For a given **algorithm** working on a given **input**, how long do we need to wait for an answer? Count steps! How does the running time depend on the input in the worst-case? average-case? Big-O

Can we detect problems that are **efficiently solvable**?

# Time complexity

For M a deterministic decider,

its **running time** or **time complexity** is the function

f: N → $R^+$ given by

f(n) = maximum number of steps M takes before halting,

over all inputs of length n.

worst-case analysis

Plus, instead of calculating precisely, **estimate** f(n) by using big-O notation.

TIME(n) = { L | L is dec by TM running in linear time }

# Time complexity classes

$\boxed{\textbf{TIME(t(n))}}$ = { L | L is decidable by a TM running in O(t(n)) }

- **Exponential**

Brute-force search

- **Polynomial**

$$P = \bigcup_k TIME(n^k)$$

Invariant under many models of TMs

- **Logarithmic**

May not need to read all of input

# The class P

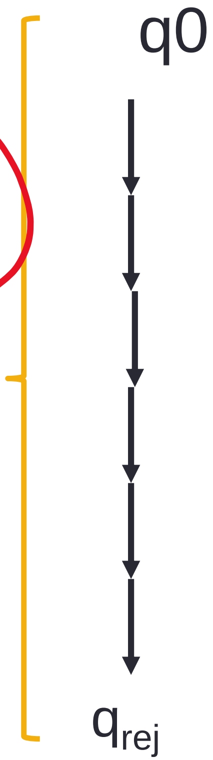
*Why is it okay to group all polynomial running times?*

- Contains all the "feasibly solvable" problems.
- Invariant for all the "usual" deterministic TM models
  - multitape machines (Theorem 7.8)
  - multi-write

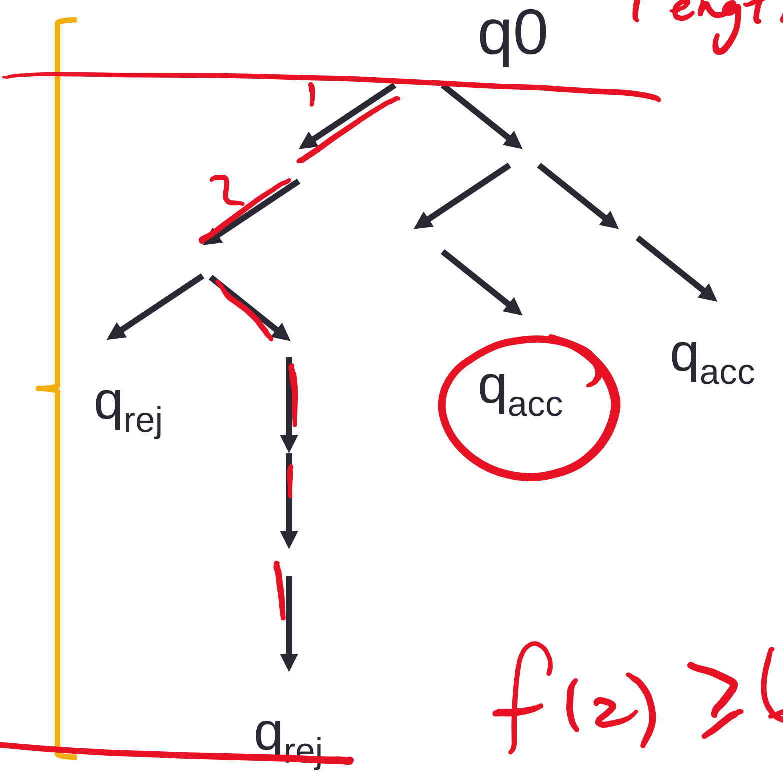
# Which Turing machine model?

input of length 2

**deterministic computation**




**non-deterministic computation**




$f(2) \geq 6$

# Time complexity

For M a deterministic decider, its **running time** or **time complexity** is the function f: N → $R^+$ given by

f(n) = maximum number of steps M takes before halting, over all inputs of length n.

For M a **nondeterministic decider**, its **running time** or **time complexity** is the function f: N → $R^+$ given by

f(n) = maximum number of steps M takes before halting on any branch of its computation, over all inputs of length n.

# Time complexity classes

**DTIME ( t(n) )** = { L | L is decidable by O( t(n) )
deterministic, single-tape TM }

is a tighter bound

**NTIME ( t(n) )** = { L | L is decidable by O( t(n) )
nondeterministic, single-tape TM }

Is DTIME($n^2$) is a subset of DTIME($n^3$)?

A. Yes
B. No
C. Not enough information to decide
D. I don't know

# Time complexity classes

**DTIME ( t(n) )** = { L | L is decidable by O( t(n) ) deterministic, single-tape TM }

**NTIME ( t(n) )** = { L | L is decidable by O( t(n) ) nondeterministic, single-tape TM }

if $L \in NTIME(f(n))$
then
$L \in DTIME(2^{f(n)})$

Is DTIME($n^2$) is a subset of NTIME($n^2$)?

A. Yes
B. No
C. Not enough information to decide
D. I don't know

# Time complexity classes

**DTIME ( t(n) )** = { L | L is decidable by O( t(n) )
deterministic, single-tape TM }

**NTIME ( t(n) )** = { L | L is decidable by O( t(n) )
nondeterministic, single-tape TM }

Is NTIME($n^2$) is a subset of DTIME($n^2$)?

A. Yes
B. No
C. Not enough information to decide
D. I don't know

# P vs. NP

Millenium Problem

$1000000

# "Feasible" i.e. P

$$P = \bigcup_k TIME(n^k)$$

- Can't use nondeterminism
- Can use multiple tapes

*Often need to be "more clever" than naïve / brute force approach*

Examples

PATH = {<G,s,t> | G is digraph with n nodes there is path from s to t}

Use breadth first search to show in P

RELPRIME = { <x,y> | x and y are relatively prime integers}

Use Euclidean Algorithm to show in P

L(G) = {w | w is generated by G}  where G is any CFG

Use Dynamic Programming to show in P

# "Verifiable" i.e. NP

- Can be **decided** by a **nondeterministic** TM in polynomial time
- Best known **deterministic** solution is brute-force
- Solution can be verified by a deterministic TM in polynomial time

P = NP?

# Examples in NP

Solution can be *verified* by a deterministic TM in polynomial time

HAMPATH = {<G,s,t> | G is digraph with n nodes there is path from s to t that goes through *every* node exactly once}

VERTEX-COVER = { <G,k> | G is an undirected graph with n nodes that has a k-node vertex cover}

CLIQUE = { <G,k> | G is an undirected graph with n nodes that has a k-clique}

SAT = { < X > | X is a satisfiable Boolean formula with n variables}

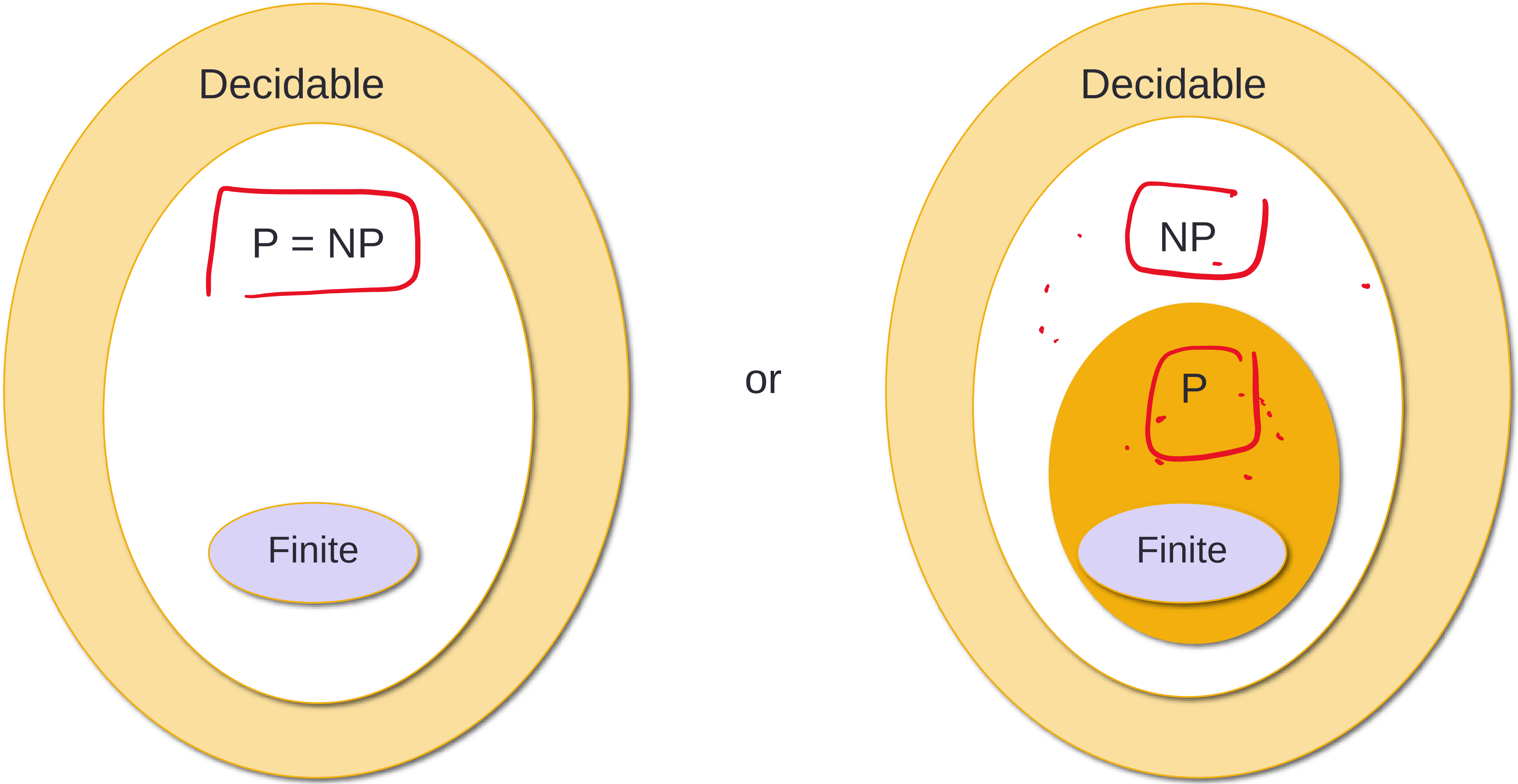
# Examples in NP

Claim: HAMPATH is in NP

Pf: Build nondeterministic decider

"On input <G, s, t>

1. Nondeterministically pick a permutation of vertices of G that starts w/ s & ends w/ t.
2. Check consecutive vertices in list are adj. If so, accept"


Decidable
P = NP
Finite
or
Decidable
NP
P
Finite

# P vs. NP

| **Problems in P** | **Problems in NP** |
|---|---|
| (Membership in any) CFL | **Any problem in P** |
| PATH | HAMPATH |
| $E_{DFA}$ | CLIQUE |
| $EQ_{DFA}$ | VERTEX-COVER |
| Addition, multiplication of integers | TSP |
| … | SAT |
| | … |

# Next time

Pre-class reading skim Chapter 7