SO FAR

- Computational model (Turing machine)
- Complexity class P: efficiently solvable (decision) problems
- Complexity class NP: efficiently verifiable (decision) problems
- NP-complete problems (in particular, SAT)
- Complement class coNP
- EXP

NEXT UP

- NEXP ; If EXP ≠ NEXP, then P ≠ NP; padding
- Is it true that more time makes it possible to solve strictly more problems?
- If P ≠ NP, are there complexity classes strictly between P and NP?
- Diagonalization
- Oracles

NONDETERMINISTIC EXPONENTIAL TIME AND PADDING

DEFINITION $\boxed{NEXP = \bigcup_{k \in \mathbb{N}} NTIME(2^{n^k})}$

Clearly $P \subseteq NP \subseteq EXP \subseteq NEXP$

Why care about such classes?

THEOREM $\boxed{\text{If } EXP \neq NEXP, \text{ then } P \neq NP}$

Proof By contraposition. Suppose $P = NP$;
prove $EXP = NEXP$

Consider any language $L \in NTIME(2^{n^k})$

There is a nondeterministic TM M deciding L.

Let $L_{pad} = \{ \langle x, 1^{2^{|x|^k}} \rangle \mid x \in L \}$

Claim: $L_{pad} \in NP$

- First check input is on correct form
- Then run M

This can be done in polynomial time in the size of the input, since we have padded the input to be exponentially large

By assumption, $L_{pad} \in NP \Rightarrow L_{pad} \in P$, decided by M^*, say.

But then $L \in EXP$

- Given x, construct $\langle x, 1^{2^{|x|^k}} \rangle$.
- Then run M^* on this input, and answer as M^* does

Runs in exponential time ▨

PADDING: Useful technique to "scale up" or "scale down" results between weaker and stronger complexity classes

> What does all of this mean? Section 2.7 in Arora-Barak is highly recommended reading

How to prove that two complexity classes are different?

Find language in one class that is not in the other

Every language L in complexity class $\mathcal{C}$ is decided by some Turing machine M_L running within resource bound specified by $\mathcal{C}$

Separate $\mathcal{C}_1$ and $\mathcal{C}_2$ by finding Turing machine M running within resource bound specified by $\mathcal{C}_1$ that differs from every Turing machine in $\mathcal{C}_2$ on at least one input

Then

$$L = \{ x \mid M(x) = 1 \}$$

is a language separating $\mathcal{C}_1$ and $\mathcal{C}_2$

$$L \in \mathcal{C}_1 \setminus \mathcal{C}_2$$

Essentially only known tool to do this:

DIAGONALIZATION

DIAGONALIZATION

Recall: Turing machine specified by
- finite alphabet Σ (symbols)
- finite set of possible states Q
- transition function (program) mapping
  $Q \times \{$ symbols read on tapes $\}$ to
  $Q \times \{$ symbols written on tapes $\} \times \{$ head movements $\}$

Can map Σ and Q to binary encoding
Can agree on encoding of Turing machines as (finite) binary strings

Let's use encoding such that:

(a) Encoding ended by "stop marker" binary code; padding with more bits after stop marker has no effect but encodes same machine

(b) "syntax error" encoding identified with trivial Turing machine that immediately halts and rejects

THEN

① Every string $x \in \{0,1\}^*$ represents TM M_x
Given $i \in \mathbb{N}^+$, write M_i for TM encoded by i in binary

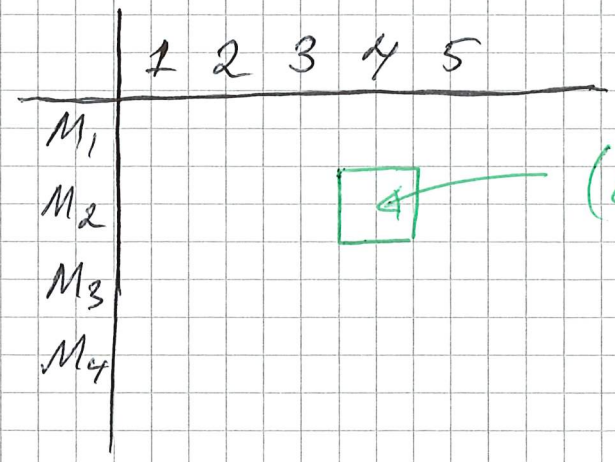
② Every Turing machine M is represented by infinitely many strings / integers

③ This representation is EFFICIENT — given x, can simulate M_x on universal TM U with at most logarithmic overhead

Consider table with rows and columns indexed by integers

Rows $\Leftrightarrow$ Turing machines

Columns $\Leftrightarrow$ Inputs



Construct Turing machine by walking diagonally downwards right, ensuring at least one mismatch per row $\Rightarrow$ Contradiction, such Turing machine cannot exist

TIME HIERARCHY THEOREM

If f, g are time-constructible functions satisfying

$$f(n) \log f(n) = o(g(n))$$

then $\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$

Time-constructible? Given 1^n, possible to write $f(n)$ in binary to tape in time $O(f(n))$. Any natural function is time-constructible (as long as $f(n) \geq n$)

$n \log n$, n^2, 2^n, et cetera

Will prove:

TIME HIERARCHY THEOREM, VANILLA VERSION

$$DTIME(n) \subsetneq DTIME(n^{1.5})$$

Proof Let D be the following Turing machine:

On input x, run universal TM U for $|x|^{1.4}$ steps to simulate execution of M_x on x

If U halts with output $b \in \{0,1\}$
    output opposite answer $1-b$
else
    output 0

How set time bound for TM? E.g.:

- compute $|x|$
- then compute $|x|^{1.4}$ and store on counter tape
- fill dedicated work tape with special marker symbol until counter decreased to 0
- now move back to start of work tape, start simulation, and at every step move right on "timer tape"
- if ever see non-marker symbol on timer tape, terminate simulation and output 0.

D decides some language, namely

$$L_D = \{ x \mid D(x) = 1 \}$$

D runs in time $\sim n^{1.4} \log n$

Hence $L_D \in DTIME(n^{1.5})$

We claim $L_D \notin DTIME(n)$

For contradiction, assume $\exists M^*$ that on any x runs in $\leq c \cdot |x|$ steps and outputs $D(x)$ [where c is a constant]

M^* can be simulated on any x in time $O(|x| \log |x|)$ by U.

Fix large enough N such that $n^{1.4}$ is larger than this if $n \geq N$.

Pick some x of length $\geq N$ such that $M_x = M^*$ (possible by ② above)

Then:

- On input x, D will simulate M^* on x
- M^* will have time to terminate and output $M^*(x)$
- By construction, we have $D(x) = 1 - M^*(x)$
- But M^* was supposed to decide L_D, so $D(x) = M^*(x)$

Contradiction. Hence no such M^* exists, QED

There is also a time hierarchy theorem for nondeterministic computation

NONDETERMINISTIC TIME HIERARCHY THEOREM

If f, g are time constructible functions such that $f(n+1) = o(g(n))$, then

$$NTIME(f(n)) \subsetneq NTIME(g(n))$$

- The proof is much more subtle
- We will skip this — see Arora-Barak for details

WHAT IS BETWEEN P and NP?

Most problems in NP that we study are known to be either in P or NP-complete

QUESTION: Is that the case for all problems in NP ?!

Results of that flavour are known as DICHOTOMY THEOREMS

ANSWER: If P=NP, then yes (trivially)
If P≠NP, then no, in a very strong sense

LADNER'S THEOREM

If P≠NP, then there exists a strict, infinite hierarchy of complexity classes between P and NP

GUIDED EXERCISE FOR PROBLEM SET:

- Prove vanilla version of this statement
- Most of the details can be found in Arora-Barak
- Want you to go through the proof and make sure you understand it
- Write nice, complete exposition, aimed at students finishing ADS, say
- Opportunity to practise writing and presentation skills

What lies between P and NP?

If $P = NP$, nothing (clearly)

But what if $P \neq NP$?

LADNER'S THEOREM

If $P \neq NP$, then there exists a strict, infinite hierarchy of complexity classes between P and NP

Guided exercise for problem set

- Pose vanilla version of this statement
- Most of details can be found in textbook
- Want you to go through the proof and make sure you understand it
- Write nice, complete exposition aimed at student finishing 420, say
- So practice also writing and presentation skills.

LADNER'S THEOREM, VANILLA VERSION

If $P \neq NP$, then there exists a language $L \in NP \setminus P$ that is not NP-complete

Caveat: This language L looks quite contrived...
But interesting to know it exists.

Main idea: Padding.

Let $P: \mathbb{N} \to \mathbb{N}$ be some function such that $P(n)$ is computable in time polynomial in n.

Define SAT_P to be (CNF)SAT with all size-n formulas φ padded with $n^{P(n)}$ 1's

$$SAT_P = \{\varphi 0 1^{n^{P(n)}} \mid \varphi \in CNFSAT \text{ and } n = |\varphi|\}$$

That is: given string x, scan from back until first 0. Let φ be everything before that 0. Set $n = |\varphi|$ = length of this. Have string 1^k after 0.
$x \in SAT_P$ if (a) $\varphi \in CNFSAT$ and (b) $k = n^{P(n)}$

OBSERVATIONS

- If $P(n) \in O(1)$, then SAT_P NP-complete
- If $P(n) = \Omega(n/\log n)$, then $SAT_P \in P$

Proof: Problem set

Want to choose padding function in some clever way so that SAT_P is too hard to be in P (assuming $P \neq NP$) but too easy to be NP-complete (because the padding gives extra time)

Here is our padding function

H(n)

```
if n ≤ 4
    return 1
else
    i := 0;  failed := TRUE
    while i < log log n and failed
        failed := FALSE;  i := i + 1;
        for all x ∈ {0,1}* with |x| ≤ log n
            simulate M_i on x for i · |x|^i steps
            if M_i didn't terminate
                failed := TRUE
            else
                let b := output of M_i(x)
                split x = φ 0 1^k and
                    let s := |φ|
                check that b = 1 if and only if
                    φ ∈ CNFSAT and k = s^H(s)
                else
                    failed := TRUE
        endfor
    endwhile
    return i
```

Recursive call

Checking if M_i decides SAT_H correctly on all strings of at most logarithmic size

CLAIMS ABOUT H

(1) H is well-defined (i.e., the algorithm computes a specific function)

(2) H(n) is computed in time polynomial in n

(3) $SAT_H \in P$ if and only if $H(n) = O(1)$
[i.e., there exists a K such that $\forall n\ H(n) \leq K$]

(4) If $SAT_H \notin P$ then $H(n) \to \infty$ as $n \to \infty$

Proofs: Problem set (plus read Arora-Barak)

Now assume $P \neq NP$

(i) Suppose $SAT_H \in P$.
Then we can show that CNFSAT $\in P$
But CNFSAT NP-complete. Contradiction.

(ii) Suppose SAT_H NP-complete
Then we can reduce CNFSAT to SAT_H efficiently
But if so can compose reductions and compress CNFSAT instance so much that they are solvable in polynomial time.
Contradiction.

Detailed proof: Problem set

Are there more interesting and natural non-NP-complete languages in NP\P?

Obviously, we don't know

But FACTORING and GRAPH ISOMORPHISM are candidates (though graph isomorphism not so much any longer)