Last time
• Computation history method
• Review, test.

Today
• Recursion theorem
• Complexity theory

Sofya Raskhodnikova
Recursion Theorem

Making TMs that can obtain their own descriptions with applications to computer viruses
A TM $P_w$ that prints $w$

There is a computable function $q$ that on input $w$ outputs $\langle P_w \rangle$, where $P_w$ is a TM that prints $w$.

$P_w =$ "Erase input.
1. Print $w$ and halt."

TM computing $q$:
``On input $w$,
1. Print $\langle P_w \rangle$ and halt."

3/17/2016
Theorem. There is a TM S that erases input, prints $\langle S \rangle$ and halts.

Proof:

- $q(w) = \langle P_w \rangle$

$S$ = "Erase input."

1. Run TM A.
2. Run TM B.
3. Halt."

$A = P_{\langle B \rangle}$

$B$ = "On input $\langle M \rangle$, where $M$ is a TM,"

1. Compute $\langle P_{\langle M \rangle} \rangle = q(\langle M \rangle)$.
2. Construct TM $S'$:

$S'$ = "Erase input."

1. Run $P_{\langle M \rangle}$.
2. Run $M$.
3. Halt."

3. Output $S'$ and halt."
In English

• Write this sentence.

• Write two copies of the following, the second one in quotes:
  ``Write two copies of the following, the second one in quotes:’’
Recursion Theorem

If there is a TM T that computes a function \( t(\langle R \rangle, w) \) then there is a TM R that computes \( r(w) = t(\langle R \rangle, w) \).

**Punchline:** ``Obtain your own description’’ is a valid step in an algorithmic description of a TM.

\[ R = \text{``Erase input.} \]
\[ 1. \text{ Run TM A.} \]
\[ 2. \text{ Run TM B.} \]
\[ 3. \text{ Run TM T.} \]
\[ 4. \text{ Halt.’’} \]

\[ A = P_{\langle BT \rangle} \]

(The rest of the proof -- on the board)
Application of recursion theorem

• Give an alternative proof that $A_{TM}$ is undecidable. (on the board)
A TM $M$ is **minimal** if there is no TM equivalent to $M$ that has a shorter description than $\langle M \rangle$.

$MIN_{TM} = \{ \langle M \rangle \mid M \text{ is minimal TM} \}$.

Show that $MIN_{TM}$ is not Turing-recognizable.

(on the board)
Complexity Theory

Already learned
• Automata theory
• Computability theory

Last unit: complexity theory

First topic: time complexity
• Measuring complexity (as in Algorithms)
• Asymptotic notation (as in Algorithms)
• Relationships between models
How much time/memory needed to decide a language?

Example: Consider $A = \{0^m 1^m | m \geq 0\}$.

• Time needed for 1-tape TM?

• $M_1 =$ “1. Scan input and reject if it is not of the form $0^*1^*$.  
2. Repeat while both $0$s and $1$s remain on the tape:  
3. Cross off one $0$ and one $1$  
4. Accept if no $0$s and no $1$s left; otherwise reject.””
Running time analysis

If $M$ is a TM and $f : \mathbb{N} \rightarrow \mathbb{N}$ then

“$M$ runs in time $f(n)$” means

for every input $w \in \Sigma^*$ of length $n$,

$M$ on $w$ halts within $f(n)$ steps

• Focus on worst case:
  – upper bound on running time for all inputs of given length

• Exact time depends on computer
  – instead measure asymptotic growth
Asymptotic notation

**O-notation (upper bounds):**

\[ f(n) = O(g(n)) \] means

there exist constants \( c > 0, n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

**Example:** \( 2n^2 = O(n^3) \) \( (c = 2, n_0 = 1) \)

functions, not values
Asymptotic Notation

- **One-sided equality:** \( T(n) = O(f(n)) \).

  - Not transitive:
    - \( f(n) = 5n^3; \ g(n) = 3n^2 \)
    - \( f(n) = O(n^3) \) and \( g(n) = O(n^3) \)
    - but \( f(n) \neq g(n) \).
  
  - Alternative notation: \( f(n) \in O(g(n)) \).
Examples

• $10^6 n^3 + 2n^2 - n + 10 = O(n^3)$

• $\sqrt{n} + \log n = O(\sqrt{n})$

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

• $n = O(n), \text{ also } O(n^2)$
Ω-notation (lower bounds)

Ω-notation is an upper-bound notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$f(n) = \Omega(g(n))$ means there exist constants $c > 0, n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.

**Example:** \( \sqrt{n} = \Omega(\log n) \) \quad (c = 1, n_0 = 16)
• **Be careful:** “Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.”
  – Meaningless!
  – Use $\Omega$ for lower bounds.
$\Theta(g(n))$ means both $O(g(n))$ and $\Omega(g(n))$

**Example:** \[ \frac{1}{2} n^2 - 2n = \Theta(n^2) \]

Polynomials are simple:
\[ a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d) \]
**o-notation and \( \omega \)-notation**

\( \Theta \)-notation and \( \Omega \)-notation are like \( \leq \) and \( \geq \).

\( o \)-notation and \( \omega \)-notation are like \( < \) and \( > \).

**Example:**

\[ f(n) = o(g(n)) \text{ means for every constant } c > 0, \]

there exists a constants \( n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

**Example:**

\[ 2n^2 = o(n^3) \quad (n_0 = 2/c) \]
# Overview

<table>
<thead>
<tr>
<th>Notation</th>
<th>... means …</th>
<th>Think…</th>
<th>Example</th>
<th>(\lim_{n\to\infty} \frac{f(n)}{g(n)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f(n) = O(g(n)))</td>
<td>(\exists c &gt; 0, n_0 &gt; 0, \forall n &gt; n_0: 0 \leq f(n) &lt; cg(n))</td>
<td>Upper bound</td>
<td>(100n^2 = O(n^3))</td>
<td>If it exists, it is &lt; 1</td>
</tr>
<tr>
<td>(f(n) = \Omega(g(n)))</td>
<td>(\exists c &gt; 0, n_0 &gt; 0, \forall n &gt; n_0: 0 \leq cg(n) &lt; f(n))</td>
<td>Lower bound</td>
<td>(n^{100} = \Omega(2^n))</td>
<td>If it exists, it is &gt; 0</td>
</tr>
<tr>
<td>(f(n) = \Theta(g(n)))</td>
<td>both of the above: (f = \Omega(g)) and (f = O(g))</td>
<td>Tight bound</td>
<td>(\log(n!) = \Theta(n \log n))</td>
<td>If it exists, it is &gt; 0 and &lt; 1</td>
</tr>
<tr>
<td>(f(n) = o(g(n)))</td>
<td>(\forall c &gt; 0, \exists n_0 &gt; 0, \forall n &gt; n_0: 0 \leq f(n) &lt; cg(n))</td>
<td>Strict upper bound</td>
<td>(n^2 = o(2^n))</td>
<td>Limit exists, = 0</td>
</tr>
<tr>
<td>(f(n) = \omega(g(n)))</td>
<td>(\forall c &gt; 0, \exists n_0 &gt; 0, \forall n &gt; n_0: 0 \leq cg(n) &lt; f(n))</td>
<td>Strict lower bound</td>
<td>(n^2 = \omega(\log n))</td>
<td>Limit exists, = 1</td>
</tr>
</tbody>
</table>
Common Functions: Asymptotic Bounds

• **Polynomials.** $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.

• **Logarithms.** $\log_a n = \Theta(\log_b n)$ for all constants $a, b > 0$.
  
  For every $x > 0$, $\log n = o(n^x)$.

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

• **Factorial.** $n! = n(n - 1) \ldots 1$.

  By Sterling’s formula,
  
  $$n! = (\sqrt{2\pi n}) \left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}$$
Sort by asymptotic order of growth

a) $n \log n$
b) $\sqrt{n}$
c) $\log n$
d) $n^2$
e) $2^n$
f) $n$
g) $\log \log n$
h) $n!$
i) $n^{1,000,000}$
j) $n^{1/\log(n)}$
k) $\log(n!)$
l) $\binom{n}{2}$
m) $2^{n^2}$
n) $2^{2^n}$
TIME($f(n)$) is a class of languages.

$A \in \text{TIME}(f(n))$ means that some 1-tape TM $M$ that runs in time $O(f(n))$ decides $A$. 
The class $P$

$P$ is the class of languages decidable in polynomial time on a deterministic 1-tape TM:

$$P = \bigcup_{k} TIME(n^k).$$

- The same class even if we substitute another reasonable deterministic model.
- Roughly the class of problems realistically solvable on a computer.