Comments on Lower-Bound Proofs

• They are usually difficult.
  – Worthy of a Ph.D. degree.

• A lower bound that matches a known upper bound (given by an efficient algorithm) shows that the algorithm is optimal.
  – The simple $O(n^2)$ algorithm for PALINDROME is optimal.

• This happens rarely and is model dependent.
  – Searching, sorting, PALINDROME, matrix-vector multiplication, etc.
Decidability and Recursive Languages

- Let $L \subseteq (\Sigma - \{\square\})^*$ be a **language**, i.e., a set of strings of symbols with a finite length.
  - For example, $\{0, 01, 10, 210, 1010, \ldots\}$.

- Let $M$ be a TM such that for any string $x$:
  - If $x \in L$, then $M(x) = \text{“yes.”}$
  - If $x \notin L$, then $M(x) = \text{“no.”}$

- We say $M$ **decides** $L$.

- If $L$ is decided by some TM, then $L$ is **recursive**.
  - Palindromes over $\{0, 1\}^*$ are recursive.
Acceptability and Recursively Enumerable Languages

• Let \( L \subseteq (\Sigma - \{|\}|)^* \) be a language.

• Let \( M \) be a TM such that for any string \( x \):
  – If \( x \in L \), then \( M(x) = "yes." \)
  – If \( x \notin L \), then \( M(x) = \uparrow \).

• We say \( M \) accepts \( L \).
Acceptability and Recursively Enumerable Languages (concluded)

- If $L$ is accepted by some TM, then $L$ is a **recursively enumerable language**.
  
  - A recursively enumerable language can be generated by a TM, thus the name.
  
  - That is, there is an algorithm such that for every $x \in L$, it will be printed out eventually.
Recursive and Recursively Enumerable Languages

Proposition 2 If \( L \) is recursive, then it is recursively enumerable.

- We need to design a TM that accepts \( L \).
- Let TM \( M \) decide \( L \).
- We next modify \( M \)'s program to obtain \( M' \) that accepts \( L \).
- \( M' \) is identical to \( M \) except that when \( M \) is about to halt with a “no” state, \( M' \) goes into an infinite loop.
- \( M' \) accepts \( L \).
Turing-Computable Functions

• Let \( f : (\Sigma - \{\square\})^* \rightarrow \Sigma^* \).
  
  – Optimization problems, root finding problems, etc.

• Let \( M \) be a TM with alphabet \( \Sigma \).

• \( M \) computes \( f \) if for any string \( x \in (\Sigma - \{\square\})^* \),
  \[
  M(x) = f(x) .
  \]

• We call \( f \) a recursive function\(^a\) if such an \( M \) exists.

\(^a\)Gödel (1931).
Church’s Thesis or the Church-Turing Thesis

• What is computable is Turing-computable; TMs are algorithms (Kleene 1953).

• Many other computation models have been proposed.
  – Recursive function (Gödel), λ calculus (Church), formal language (Post), assembly language-like RAM (Shepherdson & Sturgis), boolean circuits (Shannon), extensions of the Turing machine (more strings, two-dimensional strings, and so on), etc.

• All have been proved to be equivalent.

• No “intuitively computable” problems have been shown not to be Turing-computable (yet).
Extended Church’s Thesis

- All “reasonably succinct encodings” of problems are *polynomially related*.
  - Representations of a graph as an adjacency matrix and as a linked list are both succinct.
  - The *unary* representation of numbers is not succinct.
  - The *binary* representation of numbers is succinct.
    * 1001 vs. 11111111.

- All numbers for TMs will be binary from now on.
Turing Machines with Multiple Strings

- A $k$-string Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- $K, \Sigma, s$ are as before.
- $\delta : K \times \Sigma^k \rightarrow (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times (\Sigma \times \{-, \rightarrow, -, \}^k$.
- All strings start with a $\rhd$.
- The first string contains the input.
- Decidability and acceptability are the same as before.
- When TMs compute functions, the output is on the last ($k$th) string.
A 2-String TM

\[ \delta \]

\[ \langle 10011000011100111001110 \rangle \]

\[ \langle 11110000 \rangle \]
PALINDROME Revisited

- A 2-string TM can decide PALINDROME in \( O(n) \) steps.
  - It copies the input to the second string.
  - The cursor of the first string is positioned at the first symbol of the input.
  - The cursor of the second string is positioned at the last symbol of the input.
  - The two cursors are then moved in opposite directions until the ends are reached.
  - The machine accepts if and only if the symbols under the two cursors are identical at all steps.
Configurations and Yielding

• The concept of configuration and yielding is the same as before except that a configuration is a \((2k + 1)\)-triple

\[(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k).\]

  – \(w_iu_i\) is the \(i\)th string.
  – The \(i\)th cursor is reading the last symbol of \(w_i\).
  – Recall that \(\rhd\) is each \(w_i\)’s first symbol.

• The \(k\)-string TM’s initial configuration is

\[(s, \rhd, x, \rhd, \epsilon, \rhd, \epsilon, \ldots, \rhd, \epsilon).\]
Time Complexity

• The multistring TM is the basis of our notion of the time expended by TM computations.

• If for a $k$-string TM $M$ and input $x$, the TM halts after $t$ steps, then the time required by $M$ on input $x$ is $t$.

• If $M(x) = \uparrow$, then the time required by $M$ on $x$ is $\infty$.

• Machine $M$ operates within time $f(n)$ for $f : \mathbb{N} \rightarrow \mathbb{N}$ if for any input string $x$, the time required by $M$ on $x$ is at most $f(|x|)$.
  
  – $|x|$ is the length of string $x$.
  – Function $f(n)$ is a time bound for $M$. 
Time Complexity Classes

- Suppose language \( L \subseteq (\Sigma - \{\square\})^* \) is decided by a multistring TM operating in time \( f(n) \).
- We say \( L \in \text{TIME}(f(n)) \).
- \( \text{TIME}(f(n)) \) is the set of languages decided by TMs with multiple strings operating within time bound \( f(n) \).
- \( \text{TIME}(f(n)) \) is a complexity class.
  - PALINDROME is in \( \text{TIME}(f(n)) \), where \( f(n) = O(n) \).

\(^a\)Hartmanis and Stearns (1965), Hartmanis, Lewis, and Stearns (1965).
The Simulation Technique

**Theorem 3** Given any $k$-string $M$ operating within time $f(n)$, there exists a (single-string) $M'$ operating within time $O(f(n)^2)$ such that $M(x) = M'(x)$ for any input $x$.

- The single string of $M'$ implements the $k$ strings of $M$.
- Represent configuration $(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k)$ of $M$ by configuration

  $$(q, w'_1u_1 \triangleleft w'_2u_2 \triangleleft \cdots \triangleleft w'_ku_k \triangleleft \triangleleft)$$

  of $M'$.
  - $\triangleleft$ is a special delimiter.
  - $w'_i$ is $w_i$ with the first and last symbols “primed.”
The Proof (continued)

• The initial configuration of \( M' \) is

\[
(s, \overrightarrow{\prime} \overrightarrow{\prime} x \overleftarrow{\prime} \overleftarrow{\prime} \overleftarrow{\prime} \overrightarrow{\prime} \overleftarrow{\prime} \overrightarrow{\prime} \overleftarrow{\prime} \overrightarrow{\prime})
\]

\( k - 1 \) pairs

• To simulate each move of \( M \):
  - \( M' \) scans the string to pick up the \( k \) symbols under the cursors.
    - The states of \( M' \) must include \( K \times \Sigma^k \) to remember them.
    - The transition functions of \( M' \) must also reflect it.
  - \( M' \) then changes the string to reflect the overwriting of symbols and cursor movements of \( M \).
The Proof (continued)

- It is possible that some strings of $M$ need to be lengthened.
  - The linear-time algorithm on p. 36 can be used for each such string.

- The simulation continues until $M$ halts.

- $M'$ erases all strings of $M$ except the last one.

- Since $M$ halts within time $f(|x|)$, none of its strings ever becomes longer than $f(|x|)$.

- The length of the string of $M'$ at any time is $O(kf(|x|))$.

\textsuperscript{a}We tacitly assume $f(n) \geq n$. 
<table>
<thead>
<tr>
<th>string 1</th>
<th>string 2</th>
<th>string 3</th>
<th>string 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>string 1</td>
<td>string 2</td>
<td>string 3</td>
<td>string 4</td>
</tr>
</tbody>
</table>
The Proof (concluded)

- Simulating each step of $M$ takes, per string of $M$, $O(kf(|x|))$ steps.
  - $O(f(|x|))$ steps to collect information.
  - $O(kf(|x|))$ steps to write and, if needed, to lengthen the string.

- $M'$ takes $O(k^2f(|x|))$ steps to simulate each step of $M$.

- As there are $f(|x|)$ steps of $M$ to simulate, $M'$ operates within time $O(k^2f(|x|)^2)$. 
Linear Speedup\textsuperscript{a}

**Theorem 4** Let $L \in \text{TIME}(f(n))$. Then for any $\epsilon > 0$, $L \in \text{TIME}(f'(n))$, where $f'(n) = \epsilon f(n) + n + 2$.

\textsuperscript{a}Hartmanis and Stearns (1965).
Implications of the Speedup Theorem

• State size can be traded for speed.
  – $m^k \cdot |\Sigma|^{3mk}$-fold increase to gain a speedup of $O(m)$.

• If $f(n) = cn$ with $c > 1$, then $c$ can be made arbitrarily close to 1.

• If $f(n)$ is superlinear, say $f(n) = 14n^2 + 31n$, then the constant in the leading term (14 in this example) can be made arbitrarily small.
  – *Arbitrary* linear speedup can be achieved.
  – This justifies the asymptotic big-O notation.
P

- By the linear speedup theorem, any polynomial time bound can be represented by its leading term \( n^k \) for some \( k \geq 1 \).

- If \( L \) is a polynomially decidable language, it is in \( \text{TIME}(n^k) \) for some \( k \in \mathbb{N} \).
  - Clearly, \( \text{TIME}(n^k) \subseteq \text{TIME}(n^{k+1}) \).

- The union of all polynomially decidable languages is denoted by \( P \):
  \[
P = \bigcup_{k>0} \text{TIME}(n^k).
  \]

- Problems in \( P \) can be efficiently solved.
Charging for Space

• We do not charge the space used only for input and output.

• Let $k > 2$ be an integer.

• A $k$-string Turing machine with input and output is a $k$-string TM that satisfies the following conditions.
  – The input string is read-only.
  – The last string, the output string, is write-only.
  – So its cursor never moves to the left.
  – The cursor of the input string does not wander off into the $\sqsubset$'s.
Space Complexity

- Consider a \( k \)-string TM \( M \) with input \( x \).
- Assume \( \sqcup \) is never written over by a non-\( \sqcup \) symbol.
- If \( M \) halts in configuration 
  \( (H, w_1, u_1, w_2, u_2, \ldots, w_k, u_k) \), then the space required by \( M \) on input \( x \) is 
  \( \sum_{i=1}^{k} |w_i u_i| \).
- If \( M \) is a TM with input and output, then the space required by \( M \) on input \( x \) is 
  \( \sum_{i=2}^{k-1} |w_i u_i| \).
- Machine \( M \) operates within space bound \( f(n) \) for 
  \( f : \mathbb{N} \to \mathbb{N} \) if for any input \( x \), the space required by \( M \) on \( x \) is at most \( f(|x|) \).
Space Complexity Classes

• Let $L$ be a language.

• Then

$$L \in \text{SPACE}(f(n))$$

if there is a TM with input and output that decides $L$ and operates within space bound $f(n)$.

• SPACE($f(n)$) is a set of languages.
  – PALINDROME $\in$ SPACE($\log n$): Keep 3 pointers.

• As in the linear speedup theorem (Theorem 4), constant coefficients do not matter.
Nondeterminism\textsuperscript{a}

- A nondeterministic Turing machine (NTM) is a quadruple $N = (K, \Sigma, \Delta, s)$.

- $K, \Sigma, s$ are as before.

- $\Delta \subseteq K \times \Sigma \rightarrow (K \cup \{h, "yes", "no"\}) \times \Sigma \times \{←, →, −\}$ is a relation, not a function.
  - For each state-symbol combination, there may be more than one next steps—or none at all.

- A configuration yields another configuration in one step if there exists a rule in $\Delta$ that makes this happen.

\textsuperscript{a}Rabin and Scott (1959).
Computation Tree and Computation Path

\[ S \]

\[ h \]

\[ \text{“no”} \]

\[ h \]

\[ \text{“yes”} \]

\[ \text{“yes”} \]
Decidability under Nondeterminism

- Let $L$ be a language and $N$ be an NTM.

- $N$ decides $L$ if for any $x \in \Sigma^*$, $x \in L$ if and only if there is a sequence of valid configurations that ends in “yes.”
  - It is not required that the NTM halts in all computation paths.
  - If $x \notin L$, no nondeterministic choices should lead to a “yes” state.

- What is key is the algorithm’s overall behavior not whether it gives a correct answer for each particular run.

- Determinism is a special case of nondeterminism.
An Example

- Let $L$ be the set of logical conclusions of a set of axioms.
  - Predicates not in $L$ may be false under the axioms.
  - They may also be independent of the axioms, meaning they can be assumed true or false without contradicting the axioms.
An Example (concluded)

• Let $\phi$ be a predicate whose validity we would like to prove.

• Consider the nondeterministic algorithm:
  1: $b := \text{true}$;
  2: while the input predicate $\phi \neq b$ do
  3: Generate a logical conclusion of $b$ by applying some of the axioms; {Nondeterministic choice.}
  4: Assign this conclusion to $b$;
  5: end while
  6: “yes”;

• This algorithm decides $L$. 
Complementing a TM’s Halting States

• Let $M$ decide $L$, and $M'$ be $M$ after “yes” $\leftrightarrow$ “no”.

• If $M$ is a (deterministic) TM, then $M'$ decides $\overline{L}$.

• But if $M$ is an NTM, then $M'$ may not decide $\overline{L}$.
  
  – It is possible that both $M$ and $M'$ accept $x$ (see next page).

  – When this happens, $M$ and $M'$ accept languages that are not complements of each other.
\[
\begin{array}{c}
\text{\(x\)} \\
\begin{array}{c}
\begin{array}{c}
\text{\(h\)} \\
\text{\"no\"}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\text{\(h\)} \\
\text{\"yes\"}
\end{array}
\end{array}
\end{array}
\end{array}
\rightarrow
\begin{array}{c}
\text{\(x\)} \\
\begin{array}{c}
\begin{array}{c}
\text{\(h\)} \\
\text{\"yes\"
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\text{\(h\)} \\
\text{\"no\"}
\end{array}
\end{array}
\end{array}
\end{array}
A Nondeterministic Algorithm for Satisfiability

\( \phi \) is a boolean formula with \( n \) variables.

1: \textbf{for} \( i = 1, 2, \ldots, n \) \textbf{do}
2: \hspace{1em} Guess \( x_i \in \{0, 1\} \); \{Nondeterministic choice.\}
3: \textbf{end for}
4: \{Verification:\}
5: \textbf{if} \( \phi(x_1, x_2, \ldots, x_n) = 1 \) \textbf{then}
6: \hspace{1em} “yes”;
7: \textbf{else}
8: \hspace{1em} “no”;
9: \textbf{end if}
The Computation Tree for Satisfiability

\[ x_1 = 0 \]
\[ x_2 = 1 \]
\[ x_3 = 1 \]
\[ x_4 = 0 \]
\[ x_5 = 0 \]
\[ x_6 = 1 \]
\[ x_7 = 1 \]
\[ x_8 = 0 \]
Analysis

• The algorithm decides language \( \{ \phi : \phi \text{ is satisfiable} \} \).
  – The computation tree is a complete binary tree of depth \( n \).
  – Every computation path corresponds to a particular truth assignment out of \( 2^n \).
  – \( \phi \) is satisfiable if and only if there is a computation path (truth assignment) that results in “yes.”

• General paradigm: Guess a “proof” and then verify it.