Agenda

• Today
  - Return test
  - Continue Section 7.1
Announcement

• No quiz next week
Machine-dependent complexity

• Example, let L={w | w is a palindrome}
• How long will it take us to decide L on a standard TM?
  - Go back and forth crossing off matching symbols at beginning and end
  - O(n²)
• How long will it take us to decide L on a 2-tape TM?
  - Copy string
  - Compare symbols reading forward on tape 1 and backward on tape 2
  - O(n)
Complexity relationships

Theorem: Let \( t(n) \) be a function, where \( t(n) \geq n \). Then every \( t(n) \) time multitape TM has an equivalent \( O(t^2(n)) \) time single-tape TM

Proof idea: Consider structure of equivalent single-tape TM. Analyzing behavior shows each step on multi-tape machine takes \( O(t(n)) \) on single tape machine
Equivalent machines

\[ M \]

\[
\begin{array}{cccccccc}
0 & 1 & ~ & ~ & ~ & ~ & ~ & ~ \\
\end{array}
\]

\[
\begin{array}{cccccccc}
a & a & a & ~ & ~ & ~ & ~ & ~ \\
\end{array}
\]

\[
\begin{array}{cccccccc}
a & b & ~ & ~ & ~ & ~ & ~ & ~ \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\# & 0 & 1 & \# & a & a & a & \# & a & b & \# & ~ & ~ \\
\end{array}
\]
Simulating k-tape behavior

• Single tape start string is
  \[#w\#\_\#\ldots\#\_\#\#

• Each move proceeds as follows:
  - Start at leftmost slot
  - Scan right to \((k+1)^{st}\) \# to find symbol at each virtual tape head
  - Make second pass making updates indicated by k-tape transition function
  - When a virtual head moves onto a \#, shift string to right
Proof of theorem

• Analyzing simulation of k-tape machine

• Each step on single-tape machine has two phases
  - Scan tape
  - Perform operations

• How long does first phase take?
  - Length of string on tape
  - Each portion has $O(t(n))$ length (this occurs if tape heads only move right)
Proof of theorem (cont.)

• How long does second phase take?
  - Perform k steps
    • Each step may require a right shift
  - Each step takes $O(t(n))$ time
  - Total of k steps is $O(t(n))$ because k is a constant

• What’s the total time?
  - $O(t(n))$ steps each take $O(t(n))$ time
  - Total time is $O(t^2(n))$
Determinism vs. non-determinism

**Definition:** Let $P$ be a non-deterministic Turing machine. The running time of $P$ is the function $f: \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $P$ uses on any branch of its computation in any input of length $n$. 
Non-deterministic tree

Deterministic

Non-deterministic

f(n)
Complexity relationship

**Theorem:** Let $t(n)$ be a function where $t(n) \geq n$. Then every $t(n)$ time non-deterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.
Complexity relationship proof

Proof: Given a non-deterministic TM, \( P \), running in \( t(n) \) time, construct a 3-tape deterministic TM that simulates \( P \). The height of the tree is at most \( t(n) \). Assume the maximum number of branches in the tree is \( b \). Therefore, the number of leaves in the tree is \( O(bt(n)) \).

Total number of nodes is less than twice the number of leaves – i.e. \( O(b^{t(n)}) \).
Complexity relationship proof (cont.)

Deterministic TM does a breadth-first search of the non-deterministic TM’s tree.

Total time to search tree is $O(t(n))$ to travel from the root to a leaf $\times O(b^{t(n)})$, the number of leaves.

$O(t(n)b^{t(n)}) = O(2^{\log_2 t(n)} 2^{(\log_2 b)t(n)}) = O(2^{O(t(n))})$
Complexity relationship proof (cont.)

Are we done?
No! We constructed a 3-tape TM with running time $O(2^{O(t(n))})$

Single-tape TM will take
$O((2^{O(t(n))})^2) = O(2^{2O(t(n))}) = O(2^{O(t(n))})$

Are we done?
Yes!
Polynomial vs. exponential time

• We distinguish between algorithms that have polynomial running time and those that have exponential running time

• Polynomial functions – even ones with large exponents – grow less quickly than exponential functions

• We can only process large data sets with polynomial running time algorithms
Polynomial equivalence

• Two algorithms $A_1$ and $A_2$ are polynomially equivalent if we can simulate $A_2$ using $A_1$ with only a polynomial increase in running time.
The class $P$

- $P$ is the class of languages that are decidable in polynomial time on a single-tape Turing machine
  
  \[ P = \bigcup_k \text{TIME}(n^k) \]

- $P$ “roughly corresponds” to the problems that are realistically solvable on a computer
Size of input: Important consideration

• The running time is measured in terms of the size of the input
  - If we increase the input size can that make the problem seem more efficient
  - E.g., if we represent integers in unary instead of binary

• We consider only reasonable encodings
A problem in class P

- Binary tree query
- Given a binary search tree $T$ and a key $k$, find the node in $T$ with $\text{key}(\text{node}) = k$
- How do we show this problem is in class $P$?
  - Write an algorithm and show that the algorithm has running time $O(n^k)$ for some $k$
**Binary search**

\[ M = \text{"On input } \langle G,k \rangle \]

1. Let node = root(G)
2. Do while key(node) \( \neq \) k
3. If key(node) < k
4. If right(node) == NIL
5. \hspace{1em} return NIL
6. Else
7. \hspace{1em} node = right(node)
8. Else
9. \hspace{1em} If left(node) == NIL
10. \hspace{2em} return NIL
11. \hspace{1em} Else
12. \hspace{2em} node = left(node)
13. Return node
Execution time

• Worst case running time?
  - $O(|\text{nodes}|)$
  - Occurs if tree is unbalanced
  - Is this $O(n)$?
    • Yes ... any reasonable encoding will have an entry for each node
Have a wonderful weekend