Agenda

• Today
  - Return test
  - Continue Section 7.1
Announcement
n, log n, log(log n), and n*log n
log n and log log n

- f(n) vs. n
- Logarithmic scale for n
- Logarithmic scale for f(n)

Log n: Blue line
Log log n: Pink line
n, n^2, and n^2 log n

Graph showing the functions n, n^2, and n^2 log n against their respective values.
Small-o notation

Definition: Let $f$ and $g$ be two functions from $\mathbb{N}$ to $\mathbb{R}^+$ (the set of positive real numbers). Then $f(n) = o(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

i.e., for any positive real number $c$, a number $n_0$ exists such that for every $n \geq n_0$, $f(n) < c \times g(n)$.
Small-o vs. big-O

• Small-o is strictly less than
• Big-O is less than or equal to
• For any function f, is f(n) = o(f(n))?  
  - No ... never!
• For any function f, is f(n) = O(f(n))?  
  - Yes ... always!
Some identities

- $n^i = o(n^k)$ for every $i < k$
- $\log n = o(n)$
- $\log \log n = o(\log n)$
Analyzing algorithms

• We can examine an algorithm to determine how long it will take to halt on an input of length $n$

• The amount of time to complete is called the algorithms complexity class

**Definition:** Let $t: \mathbb{N} \rightarrow \mathbb{N}$ be a function. The time complexity class, \( \text{TIME}(t(n)) \), is defined as follows. \[ \text{TIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time TM} \} \]
Example

- Last class, we saw that insertion sort takes $n^2+2n-3$ time
- Insertion sort is in the time complexity class $O(n^2)$
Another example

- Finding minimum element in a set
- Amount of time depends on the structure of the input
- If set is a sorted array?
  - $O(1)$
- If set is an unsorted array?
  - $O(n)$
- If set is a balanced sorted tree?
Sorted tree
Sorted tree examined

• Finding minimum involves selecting left child until you reach a leaf
  - Number of steps = depth of tree
• Since the tree is balanced, the depth of the tree is $O(\log n)$
• What if the tree was not balanced?
Importance of model

• The complexity of our algorithms depends on assumptions about our data & other model assumptions
• The complexity of an algorithm can vary depending on the machine we use
**Machine-dependent complexity**

- Example, let $L = \{ w \mid w \text{ is a palindrome} \}$
- 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
  - $O(n^2)$
- 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

\[
\begin{array}{c}
M
\end{array}
\]

\[
\begin{array}{c}
0 & 1 & \sim & \sim & \sim & \sim & \sim & \sim \\
\end{array}
\]

\[
\begin{array}{c}
a & a & a & \sim & \sim & \sim & \sim \\
\end{array}
\]

\[
\begin{array}{c}
a & b & \sim & \sim & \sim & \sim & \sim \\
\end{array}
\]

\[
\begin{array}{c}
S
\end{array}
\]

\[
\begin{array}{c}
\# & 0 & 1 & \# & a & a & a & \# & a & b & \# & \sim & \sim \\
\end{array}
\]
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