Test on Oct 6
review on Thursday
not responsible for phases of software development

Musical chairs assignment
initialize list pointer to NULL
adding first node is a special case

\[ \text{tail} \rightarrow \text{head} \rightarrow \text{prev}(\) \\
not completely \]
create node \[ \text{connect its next} \rightarrow \text{prev} \]
then connect others to it

Algorithm Complexity
reading chapter 2

\[ \text{don't get turned off by 2.1 - skim it} \]
\[ \text{great discussion of why & how we do this} \]
\[ \text{I won't ask you about specific algorithms} \]
\[ \text{he mentions and I don't} \]
\[ \text{book complements lecture} \]

our goals in studying algorithm complexity

to gain a general yet usable understanding of
how to determine algorithm complexity
consider
the time required for algorithm to complete
the amount of space used
the complexity of writing algorithm correctly
Big-O notation
most common terminology for comparing algorithms
big-O stands for "order of"

approximate measure of the complexity of the
average case of an algorithm usually based
on the amount of time or memory used
often consider # of comparisons or exchanges
can also look at steps of algorithm
but constant time things are washed out
by loops on data
lesser terms and constants are ignored

$O(n) + O(n^2) \implies O(n^2)$
$35O(n) \implies O(n)$

lesser term that is ignored

Quicksort demo
# of comparisons
# of exchanges
pivot should divide list in half
divide in half $n/2 \implies O(\log n)$

Big-O describes how the size of the input data
will affect running time

sometimes combine strategies
use quicksort for big list
switch to insertion sort for smaller list
use binary search to narrow in on possible solution
switch to linear search for lower overhead
Refer to this table of values that is not in textbook
(text has running time on p. 48 and p. 63)

<table>
<thead>
<tr>
<th>n</th>
<th>( \log_2 n )</th>
<th>nlog_2 n</th>
<th>( n^2 )</th>
<th>( n^3 )</th>
<th>( 2^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>( 4 \times 2 )</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>16</td>
<td>64</td>
<td>( 16 \times 2^3 )</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>64</td>
<td>256</td>
<td>4096</td>
<td>( 65,536 \times 2^6 )</td>
</tr>
<tr>
<td>100</td>
<td>7</td>
<td>700</td>
<td>10,000</td>
<td>1,000,000</td>
<td>( 1,268 \times 10^9 )</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>10,000</td>
<td>1,000,000</td>
<td>1,000,000,000</td>
<td>really big</td>
</tr>
</tbody>
</table>

\( \log_2 N \) = exponent for which \( 2^{\log_2 N} = N \)

\[ \log_2 50 = 6 \text{ because } 2^6 = 64 \]

\( 32 < 50 < 64 \)

**Algorithm Examples for orders of complexity**

- \( O(1) \) constant time
  - find largest value in a sorted array
- \( O(\log n) \) logarithmic
  - binary search on a sorted list
- \( O(n) \) linear
  - find largest value in an unsorted array
- \( O(n \log n) \) dividing in half and combining
  - merge sort is always \( n \log n \)
  - quicksort is worst case \( O(n^2) \) bad pivots
- \( O(n^2) \) quadratic
  - insertion sort usually results from
  - selection sort nested for loops
- \( O(n^3) \) cubic
  - matrix multiplication \( \times \text{something involving} \times \text{something involving} \)
  - \( N \times N \) matrix with \( N \times N \) multiplications at each location
- \( O(2^n) \) exponential
  - adding one data item doubles running time
  - travelling salesman \( \Rightarrow \text{visit} \ N \text{ cities only once; shortest path} \)

Really hard NP-completeness

Distributing to \( m \) processors "only" helps constant time
by multiplying by \( \frac{1}{m} \); doesn't change big-O complexity

**Big-O is only one way of evaluating complexity**

Not the only way

**Heuristics \( \Rightarrow \) hints to help find a solution**