CMPSC250
Lecture 1: Algorithms

Prof. John Wenskovitch
01/22/2016
What is an Algorithm?

- **Algorithm** – A step-by-step procedure for solving a problem or accomplishing some end.

- **Program** – An algorithm expressed in a language the computer can understand.

- An algorithm **solves** a problem if it produces an acceptable output on EVERY input.
How do we Compare Algorithms?

- Implementation complexity (likelihood of bugs)
- Resource usage
  - Run time
  - Space used (used to be more important)

Consider two choices for a programmer:
1. Implement an algorithm, then run it to find out how long it takes.
2. Figure out how long it will take an algorithm to run, then decide whether or not it is worthwhile to implement.
Two Sorting Methods

• StupidSort.java
  – For a list of $n$ numbers:
    • $n$ possibilities in the first position, $(n - 1)$ in the second, $(n - 2)$ in the third, ...
    • There are $n!$ possible orderings, each with equal probability of occurring.
    • We expect the runtime to be $n!/2$.

• QuickSort.java
  – For a list of $n$ numbers:
    • We are sorting the whole list at every level.
    • Because we cut the list in half at every level, we expect to have $\log(n)$ levels.
    • We expect the runtime to be $n \times \log(n)$. 
## Function Growth Rates

<table>
<thead>
<tr>
<th>Function</th>
<th>$f(1)$</th>
<th>$f(2)$</th>
<th>$f(3)$</th>
<th>$f(4)$</th>
<th>$f(5)$</th>
<th>$f(10)$</th>
<th>$f(100)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>0</td>
<td>1</td>
<td>~1.585</td>
<td>2</td>
<td>~2.322</td>
<td>~3.322</td>
<td>~6.644</td>
</tr>
<tr>
<td>Linear</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>“Linearithmic”</td>
<td>0</td>
<td>2</td>
<td>~4.755</td>
<td>8</td>
<td>~11.61</td>
<td>~33.22</td>
<td>~664.4</td>
</tr>
<tr>
<td>Quadratic</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>Cubic</td>
<td>1</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>125</td>
<td>1000</td>
<td>1000000</td>
</tr>
<tr>
<td>Exponential</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>1024</td>
<td>1.268E30</td>
</tr>
<tr>
<td>Factorial</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>120</td>
<td>3628800</td>
<td>9.33E157</td>
</tr>
</tbody>
</table>

\[
\log(2) + \log(5) \quad 2(\log(2) + \log(5))
\]
How Precise do we Need to be?

<table>
<thead>
<tr>
<th>$n^2$</th>
<th>f(1)</th>
<th>f(2)</th>
<th>f(3)</th>
<th>f(4)</th>
<th>f(5)</th>
<th>f(10)</th>
<th>f(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>100</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>$3n^2$</td>
<td>3</td>
<td>12</td>
<td>27</td>
<td>48</td>
<td>75</td>
<td>300</td>
<td>30000</td>
</tr>
<tr>
<td>$4n^2$</td>
<td>4</td>
<td>16</td>
<td>36</td>
<td>64</td>
<td>100</td>
<td>400</td>
<td>40000</td>
</tr>
<tr>
<td>$3n^3$</td>
<td>3</td>
<td>24</td>
<td>81</td>
<td>192</td>
<td>375</td>
<td>3000</td>
<td>300000</td>
</tr>
</tbody>
</table>

- Measure performance as input size increases without bound (towards infinity)
- Ignore multiplicative constants
- Ignore lower order terms (why?)
Big O

- \( f(x) \in O(g(x)) \) iff there exists some constant \( c > 0 \) and some input size \( x_0 \) such that \( f(x) < c \times g(x) \) for all \( x > x_0 \).
Asymptotic Performance

- **Big O** – Upper bound on the asymptotic performance
- **Big Omega** – Lower bound on the asymptotic performance
- **Theta** – Exact bound (both upper and lower)

- We use Big O almost exclusively. *(why?)*
  - Theta is harder to show.
  - Lower bounds are typically harder to show than upper bounds
Tilde Approximation

• Practically speaking, Tilde is like Theta but without dropping the multiplicative constants
  – Ex: \( f(n) = 6n^2 + 18n - 50 \)
    • Theta\( (n^2) \) (order of growth)
    • Also O\( (n^2) \)
    • \( \sim 6n^2 \)

• Tilde approximation can be useful, but the coefficients are implementation-dependent.
Average Case vs. Worst Case

• Consider QuickSort again:
  – In the average case, the key we select will split the list in half for the next level. $n \times \log(n)$.
  – In the worst case, the key we select will be the largest or smallest number, which won’t reduce the number of levels. $n \times n = n^2$

• Which is more important?
Any Questions?