Time Complexity
Lecture 13
Section 10.4

Robb T. Koether

Hampden-Sydney College

Fri, Feb 13, 2009
Outline

1. “Big-O” Notation: $O(f)$
2. Estimating Run Times
3. Comparison of Growth Rates
4. Assignment
“Big-O” Notation: $O(f)$

Definition (Big-O)

Let $f : \mathbb{R} \to \mathbb{R}$ be a function.

A function $g : \mathbb{R} \to \mathbb{R}$ is “big-O” of $f$ if there exist constants $c \in \mathbb{R}, \ n_0 \in \mathbb{N}$ such that

$$g(n) \leq cf(n)$$

for all $n \geq n_0$.

In other words, $g(n)$ is bounded above by a constant multiple of $f(n)$ from some point on.
Examples (Proving and Disproving Big-O)

- Show that $5n + 8$ is $O(n)$.
- Show that $an + b$ is $O(n)$ for all $a, b \in \mathbb{R}$.
- Show that $n^2 + 10n + 5$ is $O(n^2)$.
- Show that $n^2 + 10n + 5$ is not $O(n)$.
Growth Rates

- If $g$ is $O(f)$, then the growth rate of $g(n)$ is no greater than the growth rate of $f(n)$.
- If $g$ is $O(f)$ and $f$ is $O(g)$, then

$$O(f) = O(g)$$

and $f(n)$ and $g(n)$ have the same growth rate.
Example

Example (Equal Growth Rates)
- Show that $n$ and $5n + 8$ have the same growth rate.
Common Growth Rates

<table>
<thead>
<tr>
<th>Growth Rate</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Access an array element</td>
</tr>
<tr>
<td>$O(\log_2 n)$</td>
<td>Binary search</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Sequential search</td>
</tr>
<tr>
<td>$O(n \log_2 n)$</td>
<td>Merge sort</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Bubble sort</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>Factor an integer</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>Traveling salesman problem</td>
</tr>
</tbody>
</table>
Estimating Run Times

- Suppose the run time of a program is $O(f)$, for a specific function $f(x)$.
- Then the run time $t(n)$ is of the form
  \[ t(n) = cf(n) \]
  for some real number $c$.
- Suppose program runs in $t_0$ seconds when the input size is $n_0$. Then
  \[ c = \frac{t_0}{f(n_0)}. \]
Thus,

\[ t(n) = t_0 \left( \frac{f(n)}{f(n_0)} \right). \]
Example (Estimating Run Times)

- Suppose the run time of a program is $O(n^2)$.
- Suppose the program runs in $t_0 = 5 \, \mu\text{sec}$ when the input size is $n_0 = 100$.
- Then
  \[
  t(n) = 5 \left( \frac{n^2}{100^2} \right).
  \]
- Thus, if the input size is 1000, then the run time is
  \[
  t(1000) = 5 \left( \frac{1000^2}{100^2} \right) = 500 \, \mu\text{sec}.
  \]
Example (Estimating Run Times)

- Suppose the run time of a program is
  \[ O(n \log n). \]
- Suppose the program runs in \( t_0 = 5 \, \mu\text{sec} \) when the input size is \( n_0 = 100 \).
- Then
  \[
  t(n) = 5 \left( \frac{n \log n}{100 \log 100} \right).
  \]
- Thus, if the input size is 1000, then the run time is
  \[
  t(n) = 5 \left( \frac{1000 \log 1000}{100 \log 100} \right) = 75 \, \mu\text{sec}.
  \]
Comparison of Growth Rates

- Consider programs with run times that are $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, and $O(n^2)$.

- Assume that each program runs in 1 $\mu$sec when the input size is 100.

- Calculate the run times for input sizes $10^2$, $10^3$, $10^4$, $10^5$, $10^6$, $10^7$, and $10^8$. 
Comparison of Growth Rates

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O(1)$</th>
<th>$O(\log_2 n)$</th>
<th>$O(n)$</th>
<th>$O(n \log_2 n)$</th>
<th>$O(n^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>1 µsec</td>
<td>1 µsec</td>
<td>1 µsec</td>
<td>1 µsec</td>
<td>1 µsec</td>
</tr>
<tr>
<td>$10^3$</td>
<td>1 µsec</td>
<td>1.5 µsec</td>
<td>10 µsec</td>
<td>15 µsec</td>
<td>100 µsec</td>
</tr>
<tr>
<td>$10^4$</td>
<td>1 µsec</td>
<td>2 µsec</td>
<td>100 µsec</td>
<td>200 µsec</td>
<td>10 msec</td>
</tr>
<tr>
<td>$10^5$</td>
<td>1 µsec</td>
<td>2.5 µsec</td>
<td>1 msec</td>
<td>2.5 msec</td>
<td>1 sec</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1 µsec</td>
<td>3 µsec</td>
<td>10 msec</td>
<td>30 msec</td>
<td>1.7 min</td>
</tr>
<tr>
<td>$10^7$</td>
<td>1 µsec</td>
<td>3.5 µsec</td>
<td>100 msec</td>
<td>350 msec</td>
<td>2.8 hr</td>
</tr>
<tr>
<td>$10^8$</td>
<td>1 µsec</td>
<td>4 µsec</td>
<td>1 sec</td>
<td>4 sec</td>
<td>11.7 d</td>
</tr>
</tbody>
</table>
Comparison of $O(n^2)$ and $O(2^n)$

Now consider a program with growth rate $O(2^n)$.

Assume that the program runs in $1 \mu$sec when the input size is 100.

Calculate the run times for input sizes 100, 110, 120, 130, 140, and 150.
Comparison of $O(n^2)$ and $O(2^n)$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O(n^2)$</th>
<th>$O(2^n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 $\mu$sec</td>
<td>1 $\mu$sec</td>
</tr>
<tr>
<td>110</td>
<td>1.2 $\mu$sec</td>
<td>1 msec</td>
</tr>
<tr>
<td>120</td>
<td>1.4 $\mu$sec</td>
<td>1 sec</td>
</tr>
<tr>
<td>130</td>
<td>1.7 $\mu$sec</td>
<td>18 min</td>
</tr>
<tr>
<td>140</td>
<td>2.0 $\mu$sec</td>
<td>13 d</td>
</tr>
<tr>
<td>150</td>
<td>2.3 $\mu$sec</td>
<td>37 yr</td>
</tr>
<tr>
<td>160</td>
<td>2.6 $\mu$sec</td>
<td>37,000 yr</td>
</tr>
</tbody>
</table>
Comparison of $O(2^n)$ and $O(n!)$

- Now consider a program with growth rate $O(n!)$.
- Assume that the program runs in $1 \mu$sec when the input size is 100.
- Calculate the run times for input sizes 100, 101, 102, 103, 104, and 105.
Comparison of $O(2^n)$ and $O(n!)$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O(2^n)$</th>
<th>$O(n!)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 $\mu$sec</td>
<td>1 $\mu$sec</td>
</tr>
<tr>
<td>101</td>
<td>2 $\mu$sec</td>
<td>100 $\mu$sec</td>
</tr>
<tr>
<td>102</td>
<td>4 $\mu$sec</td>
<td>10 msec</td>
</tr>
<tr>
<td>103</td>
<td>8 $\mu$sec</td>
<td>1.1 sec</td>
</tr>
<tr>
<td>104</td>
<td>16 $\mu$sec</td>
<td>1.8 min</td>
</tr>
<tr>
<td>105</td>
<td>32 $\mu$sec</td>
<td>3.2 hr</td>
</tr>
<tr>
<td>106</td>
<td>64 $\mu$sec</td>
<td>14 d</td>
</tr>
<tr>
<td>107</td>
<td>128 $\mu$sec</td>
<td>4.2 yr</td>
</tr>
<tr>
<td>108</td>
<td>256 $\mu$sec</td>
<td>450 yr</td>
</tr>
</tbody>
</table>
Example

Example (TravelingSalesman.cpp)
- Download and run FactoringTimes.cpp.
- Download and run TravelingSalesman.cpp.
Feasible vs. Infeasible

Definition (Feasible)

An algorithm is considered to be feasible if its growth rate is $O(n^k)$ for some integer $k$. Otherwise, it is considered to be infeasible.

- Of course, nearly every algorithm is feasible for small input sizes.
- It is feasible to factor small integers ($< 50$ digits).
- It is infeasible to factor large integers ($> 200$ digits).
Assignment

Homework

- Read Section 10.4, pages 551 - 562.