Lecture 3: Program Analysis

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/~skiena
Big Oh Addition/Subtraction

Suppose $f(n) = O(n^2)$ and $g(n) = O(n^2)$.

- What do we know about $g'(n) = f(n) + g(n)$? Adding the bounding constants shows $g'(n) = O(n^2)$.

- What do we know about $g''(n) = f(n) - |g(n)|$? Since the bounding constants don’t necessarily cancel, $g''(n) = O(n^2)$.

We know nothing about the lower bounds on $g'$ and $g''$ because we know nothing about lower bounds on $f$ and $g$. 
Big Oh Multiplication by Constant

Multiplication by a constant does not change the asymptotics:

\[ O(c \cdot f(n)) \rightarrow O(f(n)) \]
\[ \Omega(c \cdot f(n)) \rightarrow \Omega(f(n)) \]
\[ \Theta(c \cdot f(n)) \rightarrow \Theta(f(n)) \]
Big Oh Multiplication by Function

But when both functions in a product are increasing, both are important:

\[ O(f(n)) \times O(g(n)) \rightarrow O(f(n) \times g(n)) \]

\[ \Omega(f(n)) \times \Omega(g(n)) \rightarrow \Omega(f(n) \times g(n)) \]

\[ \Theta(f(n)) \times \Theta(g(n)) \rightarrow \Theta(f(n) \times g(n)) \]
## Asymptotic Dominance in Action

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
<th>$\lg n$</th>
<th>$n$</th>
<th>$n \lg n$</th>
<th>$n^2$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td>0.003 μs</td>
<td>0.01 μs</td>
<td>0.033 μs</td>
<td>0.1 μs</td>
<td>1 μs</td>
<td>3.63 ms</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>0.004 μs</td>
<td>0.02 μs</td>
<td>0.086 μs</td>
<td>0.4 μs</td>
<td>1 ms</td>
<td>77.1 years</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>0.005 μs</td>
<td>0.03 μs</td>
<td>0.147 μs</td>
<td>0.9 μs</td>
<td>1 sec</td>
<td>8.4 × 10^{15} yrs</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>0.005 μs</td>
<td>0.04 μs</td>
<td>0.213 μs</td>
<td>1.6 μs</td>
<td>18.3 min</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>0.006 μs</td>
<td>0.05 μs</td>
<td>0.282 μs</td>
<td>2.5 μs</td>
<td>13 days</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>0.007 μs</td>
<td>0.1 μs</td>
<td>0.644 μs</td>
<td>10 μs</td>
<td>4 × 10^{13} yrs</td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td>0.010 μs</td>
<td>1.00 μs</td>
<td>9.966 μs</td>
<td>1 ms</td>
<td>100 ms</td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td></td>
<td>0.013 μs</td>
<td>10 μs</td>
<td>130 μs</td>
<td>10 sec</td>
<td>16.7 min</td>
<td></td>
</tr>
<tr>
<td>100,000</td>
<td></td>
<td>0.017 μs</td>
<td>0.10 ms</td>
<td>1.67 ms</td>
<td>1.16 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td></td>
<td>0.020 μs</td>
<td>1 ms</td>
<td>19.93 ms</td>
<td>115.7 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000,000</td>
<td></td>
<td>0.023 μs</td>
<td>0.01 sec</td>
<td>0.23 sec</td>
<td>31.7 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100,000,000</td>
<td></td>
<td>0.027 μs</td>
<td>0.10 sec</td>
<td>2.66 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000,000,000</td>
<td></td>
<td>0.030 μs</td>
<td>1 sec</td>
<td>29.90 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implications of Dominance

- Exponential algorithms get hopeless fast.
- Quadratic algorithms get hopeless at or before 1,000,000.
- $O(n \log n)$ is possible to about one billion.
- $O(\log n)$ never sweats.
Testing Dominance

\[ f(n) \text{ dominates } g(n) \text{ if } \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0, \text{ which is the same as saying } g(n) = o(f(n)). \]

Note the little-o – it means “grows strictly slower than”.
Implications of Dominance

- $n^a$ dominates $n^b$ if $a > b$ since
  \[
  \lim_{n \to \infty} \frac{n^b}{n^a} = n^{b-a} \to 0
  \]

- $n^a + o(n^a)$ doesn’t dominate $n^a$ since
  \[
  \lim_{n \to \infty} \frac{n^a}{n^a + o(n^a)} \to 1
  \]
Dominance Rankings

You must come to accept the dominance ranking of the basic functions:

\[ n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1 \]
Advanced Dominance Rankings

Additional functions arise in more sophisticated analysis than we will do in this course:

\[ n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg \log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1 \]
Problem of the Day

Find two functions $f(n)$ and $g(n)$ that satisfy the following relationship. If no such $f$ and $g$ exist, write ”None”.

1. $f(n) = o(g(n))$ and $f(n) \neq \Theta(g(n))$
2. $f(n) = \Theta(g(n))$ and $f(n) = o(g(n))$
3. $f(n) = \Theta(g(n))$ and $f(n) \neq O(g(n))$
4. $f(n) = \Omega(g(n))$ and $f(n) \neq O(g(n))$
Reasoning About Efficiency

Grossly reasoning about the running time of an algorithm is usually easy given a precise-enough written description of the algorithm.

When you *really* understand an algorithm, this analysis can be done in your head. However, recognize there is always implicitly a written algorithm/program we are reasoning about.
Selection Sort

selection_sort(int s[], int n)
{
    int i, j;
    int min;

    for (i=0; i<n; i++) {
        min=i;
        for (j=i+1; j<n; j++)
            if (s[j] < s[min]) min=j;
        swap(&s[i], &s[min]);
    }
}
Worst Case Analysis

The outer loop goes around $n$ times. The inner loop goes around at most $n$ times for each iteration of the outer loop. Thus selection sort takes at most $n \times n \rightarrow O(n^2)$ time in the worst case.
More Careful Analysis

An exact count of the number of times the *if* statement is executed is given by:

\[ S(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} n - i - 1 \]

\[ S(n) = (n - 2) + (n - 3) + \ldots + 2 + 1 + 0 = (n - 1)(n - 2)/2 \]

Thus the worst case running time is \( \Theta(n^2) \).
Logarithms

It is important to understand deep in your bones what logarithms are and where they come from.
A logarithm is simply an inverse exponential function. Saying $b^x = y$ is equivalent to saying that $x = \log_b y$.
Logarithms reflect how many times we can double something until we get to $n$, or halve something until we get to 1.
Binary Search

In binary search we throw away half the possible number of keys after each comparison. Thus twenty comparisons suffice to find any name in the million-name Manhattan phone book!

How many time can we halve $n$ before getting to 1?

Answer: $\lceil \lg n \rceil$. 
Logarithms and Trees

How tall a binary tree do we need until we have \( n \) leaves? The number of potential leaves doubles with each level. How many times can we double 1 until we get to \( n \)?

Answer: \( \lceil \lg n \rceil \).
Logarithms and Bits

How many bits do you need to represent the numbers from 0 to $2^i - 1$?
Each bit you add doubles the possible number of bit patterns, so the number of bits equals $\log_2(2^i) = i$. 
Logarithms and Multiplication

Recall that

$$\log_a(xy) = \log_a(x) + \log_a(y)$$

This is how people used to multiply before calculators, and remains useful for analysis.
What if $x = a$?
The Base is not Asymptotically Important

Recall the definition, $c^{\log_c x} = x$ and that

$$\log_b a = \frac{\log_c a}{\log_c b}$$

Thus $\log_2 n = \left(\frac{1}{\log_{100} 2}\right) \times \log_{100} n$. Since $1/\log_{100} 2 = 6.643$ is just a constant, it does not matter in the Big Oh.