Measuring Complexity
Announcements

Problem Set 0 is due now!

TA Hours are in SSL 030A
Measuring Complexity

• What constitutes an efficient algorithm?
  • Runs quickly on large, ‘real’ instances of problems
  • Qualitatively better than brute force
  • Scales well to large instances
Measuring Complexity : Brute Force

• Efficient: Qualitatively better than brute force

• Brute force: often exponentially large because
  • Might examine all subsets of a set: $2^n$
  • Might examine all orderings of a list: $n!$

• But $2^n$ is still not efficient even though it’s qualitatively better than $n!$
Measuring Complexity : Scalability

• Defining Scalability
  • Doubling problem size increases resource use by a constant factor

• Examples
  • $n^k : (2n)^k = 2^k n^k = c n^k$ for any fixed $k$
  • $\log n : \log 2n = \log 2 + \log n \leq c \log n$ ($n \geq 2$)

• But not
  • $2^n : 2^{2n} = 2^n \cdot 2^n$
  • $n! : (2n)! \geq n^n \cdot n!$
Polynomial Time

Functions that exhibit scalability can be bounded above by some polynomial function.

This leads us to define an efficient algorithm as one having running time bounded above by a polynomial function.

But how do we measure running time?
Worst Case Runtime

- Worst-case running time: the maximum number of steps needed to solve a problem instance of size $n$
  - Overestimates the typical run-time
  - Can frequently be determined by algorithm analysis
  - Frequently captures the runtime in practice
  - Often there’s no “worst” case we can identify
Average Case Runtime

- Average Case Runtime: The mean number of operations needed across all instances of size $n$
  - A much more nuanced, realistic measure
  - Very difficult to determine in practice
    - Are all instances equally likely?
    - If not, can we specify the probability distribution?
Growth of Functions

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds $10^{25}$ years, we simply record the algorithm as taking a very long time.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n$</th>
<th>$n \log_2 n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$1.5^n$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>4 sec</td>
</tr>
<tr>
<td>30</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>18 min</td>
<td>$10^{25}$ years</td>
</tr>
<tr>
<td>50</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>11 min</td>
<td>36 years</td>
<td>very long</td>
</tr>
<tr>
<td>100</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>12,892 years</td>
<td>$10^{17}$ years</td>
<td>very long</td>
</tr>
<tr>
<td>1,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>18 min</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>10,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>2 min</td>
<td>12 days</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>100,000</td>
<td>&lt; 1 sec</td>
<td>2 sec</td>
<td>3 hours</td>
<td>32 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1 sec</td>
<td>20 sec</td>
<td>12 days</td>
<td>31,710 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
</tbody>
</table>
Asymptotic Growth

What matters: How functions behave “as n gets large”
Asymptotic Upper Bounds

**Def'n:** $g(n)$ is *eventually bounded above by* $f(n)$ if for all sufficiently large $n$, $g(n) \leq f(n)$

That is, for some $N$, $g(n) \leq f(n)$ for all $n \geq N$

**Def'n:** $O(f(n)) = \{ g(n) : \text{for some constant } c > 0, g(n) \text{ is eventually bounded above by } cf(n) \}$

**Examples**

- $100n^2 \in O(n^3) : 100n^2 \leq c n^3$ for $c = 100$ and $n \geq 1$

- $3n \in O(n \log_2 n) : 3n \leq c n \log_2 n$ for $c = 1$ and $n \geq 8$
More Challenging Examples

- \( \log_2(\log_2(n)) \in O(\log_2(n)) \)

- Fact: For \( 1 < a < b \), \( o < \log(a) < \log(b) \)

- For \( n > 2 \), \( 1 < \log_2(n) < n \), so \( o < \log_2(\log_2(n)) < \log_2(n) \)

- \( 3n^4 - n^3 + 10n^2 - 4n + 5 \in O(n^5) \)

- Yikes! We need some tools....
**A Tool**

**Def’n:** A function $f: \mathbb{N} \rightarrow \mathbb{R}$ is *eventually positive* if $f(n) > 0$ for all sufficiently large $n \in \mathbb{N}$

**Thm:** If $f(n)$ and $g(n)$ are eventually positive functions such that the limit, as $n \rightarrow \infty$, of $f(n)/g(n)$ equals $0$, then $f(n) \in O(g(n))$

- Let $c > 0$. For all sufficiently large $n$, $f(n)/g(n) < c$

- So $f(n) < c \cdot g(n)$ (since $g(n)$ is positive for sufficiently large $n$)

**Example:** $3n^4 - n^3 + 10n^2 - 4n + 5 \in O(n^5)$

- $(3n^4 - n^3 + 10n^2 - 4n + 5)/n^5 = 3/n - 1/n^2 + 10/n^3 - 4/n^4 + 5/n^5$

- $3/n - 1/n^2 + 10/n^3 - 4/n^4 + 5/n^5 \rightarrow 0$ as $n \rightarrow \infty$
A (Much) Better Tool

Thm: L’Hôpital’s Rule
If

\[ \lim_{n \to \infty} f(n) = \lim_{n \to \infty} g(n) = \infty \]

\[ \lim_{n \to \infty} f'(n)/g'(n) \text{ exists} \]

then

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)} \]

Example: \( \log n \in O(n) \): Here \( f(n) = \log n \) and \( g(n) = n \)

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{(\log n)}{n} = \lim_{n \to \infty} \frac{(\ln n/ \ln 2)}{n} \]

\[ \lim_{n \to \infty} \frac{f'(n)}{g'(n)} = \lim_{n \to \infty} \frac{(1/(n \ln 2))}{1} = 0 \]
Speaking of logs…

Def’n: For any $b > 1$, $\log_b n$ is the number $x$ such that $b^x = n$.

Facts

• For every $b > 1$ and $\varepsilon > 0$, $\log_b n \in O(n^\varepsilon)$ (logs grow slowly!)

  • Because as $n \to \infty$, $(\log_b n)/n^\varepsilon \to 0$ (by L’Hôpital’s Rule)

• $\log_a n = \log_b n / \log_b a$

  • Example: $\log_2 n = \log_{10} n / \log_2 10 \approx 0.3 \log_{10} n$
...and Exponentials

Fact

• For every $r > 1$ and $d > 0$, $n^d \in O(r^n)$ (exponentials grow fast)
  
  • Because as $n \to \infty$, $n^d/r^n \to 0$ (by L’Hôpital’s Rule)

• Restated: Every exponential function grows faster than every power of $n$ (in fact, faster than every polynomial)
Asymptotic Lower Bounds

Def’n: $g(n)$ is **eventually bounded below by** $f(n)$ if for all sufficiently large $n$, $f(n) \leq g(n)$

Def’n: $\Omega(f(n)) = \{ g(n) : \text{for some constant } c > 0, g(n) \text{ is eventually bounded below by } c f(n) \}$

Examples

- $100 n^3 \in \Omega(n^2) : c n^2 \leq 100 n^3$ for $c = 1$ and $n \geq 1$
- $n \log_2 n \in \Omega(n) : c n \leq n \log_2 n$ for $c = 1$ and $n \geq 2$
Why Lower Bounds?

Show that an algorithm performs \textit{at least} so many steps

- Searching an unordered list of \( n \) items takes \( \Omega(n) \) steps in some cases

- Quicksort (and selection/insertion/bubble sorts) take \( \Omega(n^2) \) steps in some cases

- Mergesort takes \( \Omega(n \log(n)) \) steps in all cases
Another Tool

**Thm:** Let \( f(n) \) and \( g(n) \) be eventually positive functions such that the limit, as \( n \to \infty \), of \( f(n)/g(n) \) also \( \to \infty \), then \( f(n) \in \Omega(g(n)) \)

Example: \( 5n^3 + n^2 - 10 \in \Omega(n \log_2 n) \)

- \( (5n^3 + n^2 - 10)/n \log_2 n = 5n^2/\log_2 n - n/\log_2 n - 10/n \log_2 n \)
- \( (5n - 1) n/\log_2 n - 10/n \log_2 n \to \infty \) as \( n \to \infty \)
Asymptotically Tight Bounds

Def’n: \( \Theta(f(n)) = O(f(n)) \cap \Omega(f(n)) \)

That is: \( \Theta(f(n)) = \{ g(n) : g(n) \in O(f(n)) \text{ and } g(n) \in \Omega(f(n)) \} \)

If \( g(n) \in \Theta(f(n)) \), then \( f \) is an asymptotically tight bound on \( g \)

Example

• \( n^3 - 4n^2 + 2 \in \Theta(n^3) \)
  
  • \( n^3 - 4n^2 + 2 \in O(n^3) : n^3 - 4n^2 + 2 \leq cn^3 \text{ for } n \geq 1 \text{ & } c=1 \)
  
  • \( n^3 - 4n^2 + 2 \in \Omega(n^3) : c'n^3 \leq n^3 - 4n^2 + 2 \text{ for } n \geq 8 \text{ & } c' = 1/2 \)
**Yet Another Tool**

**Thm:** Let $f(n)$ and $g(n)$ be eventually positive functions such that the limit as $n \to \infty$ of $f(n)/g(n) \to c > 0$, then $f(n) \in \Theta(g(n))$

**Example:** $5n^3 + n^2 - 10 \in \Theta(n^3)$

- $(5n^3 + n^2 - 10)/n^3 = 5 + 1/n - 10/n^3$
- $5 + 1/n - 10/n^3 \to 5$ as $n \to \infty$

**Cor:** $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in \Theta(n^k)$ if $a_k > 0$
**And a Few More**

**Thm:** Let \( f, g, \) and \( h \) be eventually positive functions

Transitivity of \( O, \Omega, \) and \( \Theta \)

If \( f \in O(g) \) and \( g \in O(h) \), then \( f \in O(h) \) (same for \( \Omega, \Theta \))

**Sum Rule**

If \( f \in O(h) \) and \( g \in O(h) \), then \( f \pm g \in O(h) \) (same for \( \Omega, \Theta \))

Careful: This assumes that \( f \) and \( g \) are eventually positive!
In the Wild

Sample Usage: Quicksorting an $n$-element array

• Thm: The number of operations performed is $O(n^2)$

• Thm: The average number of operations performed is $O(n \log(n))$

• Thm: Quicksort will always execute $\Omega(n \log(n))$ steps

Sample Usage: Basic Bubblesort runs in time $\Theta(n^2)$
Θ(f) and Operation Count

Which operations should count when analyzing run time?

• All of them?
  • Can be difficult and/or tedious

• Better: A representative sample
  • If the algorithm performs $f(n)$ operations, it suffices to count $g(n)$ of them, where $g(n) \in \Theta(f(n))$