Back to asymptotics......

We will now look more formally at the process of simplifying running times and other measures of complexity.
Asymptotic analysis

• Goal: to simplify analysis of running time by getting rid of ”details”, which may be affected by specific implementation and hardware
  – like “rounding”: $1,000,001 \approx 1,000,000$
  – $3n^2 \approx n^2$
• Capturing the essence: how the running time of an algorithm increases with the size of the input *in the limit*.
  – Asymptotically more efficient algorithms are best for all but small inputs
Asymptotic notation

- The “big-Oh” O-Notation
  - asymptotic upper bound
  - \( f(n) \in O(g(n)) \), if there exists constants \( c \) and \( n_0 \), s.t. \( f(n) \leq c g(n) \) for \( n \geq n_0 \)
  - \( f(n) \) and \( g(n) \) are functions over non-negative integers
- Used for *worst-case* analysis
Asymptotic notation – contd.

• The “big-Omega” $\Omega$–Notation
  – asymptotic lower bound
  – $f(n) \in \Omega(g(n))$ if there exists constants $c$ and $n_0$, s.t. $c \cdot g(n) \leq f(n)$ for $n \geq n_0$

• Used to describe best-case running times or lower bounds of algorithmic problems
  – E.g., lower-bound of searching in an unsorted array is $\Omega(n)$.
Asymptotic notation – contd.

• Simple Rule: Drop lower order terms and constant factors.
  – $50 \ n \ \log \ n \in O(n \ \log \ n)$
  – $7n - 3 \in O(n)$
  – $8n^2 \ \log \ n + 5n^2 + n \in O(n^2 \ \log \ n)$

• Note: Even though $50 \ n \ \log \ n \in O(n^5)$, we usually try to express a $O()$ expression using as small an order as possible
• The “big-Theta” \( \Theta \)–Notation
  – asymptotically tight bound
  \(- f(n) \in \Theta(g(n)) \) if there exists constants \( c_1, c_2, \) and \( n_0 \), s.t. \( c_1 g(n) \leq f(n) \leq c_2 g(n) \) for \( n \geq n_0 \)

• \( f(n) \in \Theta(g(n)) \) if and only if \( f(n) \in O(g(n)) \) and \( f(n) \in \Omega(g(n)) \)

• \( O(f(n)) \) is often misused instead of \( \Theta(f(n)) \)
Asymptotic notation – contd.

• Two more asymptotic notations
  – "Little-Oh" notation $f(n) = o(g(n))$
    non-tight analogue of Big-Oh
    • For every $c$, there should exist $n_0$, s.t. $f(n) \leq c \cdot g(n)$
      for $n \geq n_0$
  • Used for comparisons of running times.
    If $f(n) \in o(g(n))$, it is said that $g(n)$ dominates $f(n)$.
  • More useful defn:
    \[
    \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0
    \]

• "Little-omega" notation $f(n) \in \omega(g(n))$
  non-tight analogue of Big-Omega
Asymptotic notation – contd.

• (VERY CRUDE) Analogy with real numbers
  \[ f(n) = O(g(n)) \implies f \leq g \]
  \[ f(n) = \Omega(g(n)) \implies f \geq g \]
  \[ f(n) = \Theta(g(n)) \implies f = g \]
  \[ f(n) = o(g(n)) \implies f < g \]
  \[ f(n) = \omega(g(n)) \implies f > g \]

• Abuse of notation: \( f(n) = O(g(n)) \) actually means \( f(n) \in O(g(n)) \).
Points to ponder and lessons

Common “colloquial” uses:

\[ \Theta(1) \] – constant.

\[ n^{\Theta(1)} \] – polynomial

\[ 2^{\Theta(n)} \] – exponential

- When is asymptotic analysis useful?
- When is it NOT useful?

Many, many abuses of asymptotic notation in Computer Science literature.

Lesson: Always remember the implicit assumptions…

Be careful!

\[ n^{\Theta(1)} \neq \Theta(n^1) \]

\[ 2^{\Theta(n)} \neq \Theta(2^n) \]
## Comparison of Running Times

<table>
<thead>
<tr>
<th>Running Time</th>
<th>Maximum problem size (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 second</td>
</tr>
<tr>
<td>$400n$</td>
<td>2500</td>
</tr>
<tr>
<td>$20n \log n$</td>
<td>4096</td>
</tr>
<tr>
<td>$2n^2$</td>
<td>707</td>
</tr>
<tr>
<td>$n^4$</td>
<td>31</td>
</tr>
<tr>
<td>$2^n$</td>
<td>19</td>
</tr>
</tbody>
</table>
### Classifying functions

<table>
<thead>
<tr>
<th></th>
<th>$T(n)$</th>
<th>10</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log n$</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>$n^{1/2}$</td>
<td>3</td>
<td>10</td>
<td>31</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>$n \log n$</td>
<td>30</td>
<td>600</td>
<td>9,000</td>
<td>130,000</td>
<td></td>
</tr>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>10,000</td>
<td>$10^6$</td>
<td>$10^8$</td>
<td></td>
</tr>
<tr>
<td>$n^3$</td>
<td>1,000</td>
<td>$10^6$</td>
<td>$10^9$</td>
<td>$10^{12}$</td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td>1,024</td>
<td>$10^{30}$</td>
<td>$10^{300}$</td>
<td>$10^{3000}$</td>
<td></td>
</tr>
</tbody>
</table>
Hierarchy of functions

- Constant
- Poly Logarithmic: \( (\log n)^5 \)
- Polynomial: \( n^5 \)
- Exponential: \( 2^n, 2^{n^5}, 2^{2^n} \)
- Double Exp

- Others: \( 2^{n \log(n)} \)

\[ 09/09/17 \]
Classifying Polynomials

Dominant term is of the form $n^c$

Polynomial
- Linear: $5n$
- Quadratic: $5n^2$
- Cubic: $5n^3$
- Others: $5n^3 \log^7(n)$, $5n^4$

EECS 3101
Logarithmic functions

- \( \log_{10} n \) = \# digits to write \( n \)
- \( \log_2 n \) = \# bits to write \( n \) = 3.32 \( \log_{10} n \)
- \( \log(n^{1000}) = 1000 \log(n) \)

Differ only by a multiplicative constant.

Poly Logarithmic (a.k.a. polylog)

\((\log n)^5 = \log^5 n\)
Crucial asymptotic facts

Logarithmic $\ll$ Polynomial

$\log_{1000} n \ll n^{0.001}$ For sufficiently large $n$

Linear $\ll$ Quadratic

$10000 n \ll 0.0001 n^2$ For sufficiently large $n$

Polynomial $\ll$ Exponential

$n^{1000} \ll 2^{0.001 n}$ For sufficiently large $n$
Are constant functions constant?

The running time of the algorithm is a “constant”
It does not depend significantly on the size of the input.

Yes • 5
Yes • 1,000,000,000,000
Yes • 0.0000000000001
No • -5
No • 0
Yes • $8 + \sin(n)$

Write $\Theta(1)$.
Polynomial Functions

Quadratic
• $n^2$
• $0.001 \ n^2$
• $1000 \ n^2$
• $5n^2 + 3000n + 2\log n$

Polynomial
• $n^c$
• $n^{0.0001}$
• $n^{10000}$
• $5n^2 + 8n + 2\log n$
• $5n^2 \log n$
• $5n^{2.5}$
Exponential functions

- $2^n$
- $2^{0.0001 \ n}$
- $2^{10000 \ n}$
- $8^n = 2^{3n}$
- $2^n / n^{100} > 2^{0.5n}$
- $2^n \cdot n^{100} < 2^{2n}$

$$2^n = 2^{0.5n} \cdot 2^{0.5n} > n^{100} \cdot 2^{0.5n}$$

$$2^n / n^{100} > 2^{0.5n}$$
Proving asymptotic expressions

Use definitions!

\[ f(n) = 3n^2 + 7n + 8 = \theta(n^2) \]
\[ f(n) \in \Theta(g(n)) \text{ if there exists constants } c_1, \, c_2, \text{ and } n_0, \text{ s.t.} \]
\[ c_1 \, g(n) \leq f(n) \leq c_2 \, g(n) \text{ for } n \geq n_0 \]

Here \( g(n) = n^2 \)

One direction (\( f(n) = \Omega(g(n)) \)) is easy

\[ c_1 \, g(n) \leq f(n) \text{ holds for } c_1 = 3 \text{ and } n \geq 0 \]

The other direction (\( f(n) = O(g(n)) \)) needs more care

\[ f(n) \leq c_2 \, g(n) \text{ holds for } c_2 = 18 \text{ and } n \geq 1 (\text{CHECK}) \]

So \( n_0 = 1 \)
Proving asymptotic expressions – contd.

Caveats!
1. constants $c_1, c_2$ MUST BE POSITIVE.
2. Could have chosen $c_2 = 3 + \varepsilon$ for any $\varepsilon > 0$. WHY?
   -- because $7n + 8 \leq \varepsilon n^2$ for $n \geq n_0$ for some sufficiently large $n_0$. Usually, the smaller the $\varepsilon$ you choose, the harder it is to find $n_0$. So choosing a large $\varepsilon$ is easier.

3. Order of quantifiers
   $\exists c_1, c_2 \exists n_0 \forall n \geq n_0, c_1g(n) \leq f(n) \leq c_2g(n)$
   vs
   $\exists n_0 \forall n \geq n_0 \exists c_1, c_2, c_1g(n) \leq f(n) \leq c_2g(n)$
   -- allows a different $c_1$ and $c_2$ for each $n$. Can choose $c_2 = 1/n!!$ So we can “prove” $n^3 = \Theta(n^2)$. 

EECS 3101
Why polynomial vs exponential?

Philosophical/Mathematical reason – polynomials have different properties, grow much slower; mathematically natural distinction.

Practical reasons

1. almost every algorithm ever designed and every algorithm considered practical are very low degree polynomials with reasonable constants.
2. a large class of natural, practical problems seem to allow only exponential time algorithms. Most experts believe that there do not exist any polynomial time algorithms for any of these; i.e. $P \neq NP$. 