### Linear Search vs Binary Search

<table>
<thead>
<tr>
<th></th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best Case</strong></td>
<td>4 at [0]</td>
<td>4 at [middle]</td>
</tr>
<tr>
<td><strong>Worst Case</strong></td>
<td>3n+2</td>
<td>4 log n + 4</td>
</tr>
</tbody>
</table>

So ... which algorithm is better?  
*What tradeoffs can you make?*
Fast Computer vs. Slow Computer

![Graph showing the comparison between linear search on a Pentium-IV and a 486 processor. The graph plots time in ms against the number of elements to be searched. The Pentium-IV shows a much steeper slope, indicating faster processing time.]
Fast Computer vs. Smart Programmer
(round 1)
Fast Computer vs. Smart Programmer
(round 2)

- Linear search on Pentium-IV
- Binary search on 486

Graph showing time in ms vs. number of elements to be searched.
Asymptotic Analysis

• Asymptotic analysis looks at the order of the running time of the algorithm
  – A valuable tool when the input gets “large”
  – Ignores the effects of different machines or different implementations of an algorithm

• Intuitively, to find the asymptotic runtime, throw away the constants and low-order terms
  – Linear search is $T(n) = 3n + 2 \in O(n)$
  – Binary search is $T(n) = 4 \log_2 n + 4 \in O(\log n)$

Remember: the fastest algorithm has the slowest growing function for its runtime
Asymptotic Analysis

• Eliminate low order terms
  – 4n + 5 ⇒
  – 0.5 n log n + 2n + 7 ⇒
  – n³ + 2ⁿ + 3n ⇒

• Eliminate coefficients
  – 4n ⇒
  – 0.5 n log n ⇒
  – n log n² =>
Properties of Logs

• log \( AB = \log A + \log B \)

• Proof: \[ A = 2^{\log_2 A}, \quad B = 2^{\log_2 B} \]

\[ AB = 2^{\log_2 A} \cdot 2^{\log_2 B} = 2^{(\log_2 A + \log_2 B)} \]

\[ \therefore \log AB = \log A + \log B \]

• Similarly:
  - log(A/B) = log A − log B
  - log(A^B) = B log A

• Any log is equivalent to log-base-2
Order Notation: Intuition

\[ f(n) = n^3 + 2n^2 \]
\[ g(n) = 100n^2 + 1000 \]

Although not yet apparent, as \( n \) gets “sufficiently large”, \( f(n) \) will be “greater than or equal to” \( g(n) \)

9/30/2007
Definition of Order Notation

- **Upper bound:** \( T(n) = \mathcal{O}(f(n)) \) \quad \text{Big-O}
  
  Exist positive constants \( c \) and \( n' \) such that
  \[ T(n) \leq c f(n) \quad \text{for all } n \geq n' \]

- **Lower bound:** \( T(n) = \Omega(g(n)) \) \quad \text{Omega}
  
  Exist positive constants \( c \) and \( n' \) such that
  \[ T(n) \geq c g(n) \quad \text{for all } n \geq n' \]

- **Tight bound:** \( T(n) = \Theta(f(n)) \) \quad \text{Theta}
  
  When both hold:
  \[ T(n) = \mathcal{O}(f(n)) \]
  \[ T(n) = \Omega(f(n)) \]
Definition of Order Notation

\( O( f(n) ) \) : a set or class of functions

\[ g(n) \in O( f(n) ) \iff \text{there exist positive consts } c \text{ and } n_0 \text{ such that:} \]

\[ g(n) \leq c \cdot f(n) \text{ for all } n \geq n_0 \]

Example:

\[ 100n^2 + 1000 \leq 5(n^3 + 2n^2) \text{ for all } n \geq 19 \]

So \( g(n) \in O( f(n) ) \)
Order Notation: Example

\[ 100n^2 + 1000 \leq 5(n^3 + 2n^2) \text{ for all } n \geq 19 \]

So \( f(n) \in O( g(n) ) \)

9/30/2007
Some Notes on Notation

• Sometimes you’ll see
  \[ g(n) = O(f(n)) \]

• This is equivalent to
  \[ g(n) \in O(f(n)) \]

• What about the reverse?
  \[ O(f(n)) = g(n) \]
Big-O: Common Names

- constant: \( O(1) \)
- logarithmic: \( O(\log n) \) \( (\log_k n, \log n^2 \in O(\log n)) \)
- linear: \( O(n) \)
- log-linear: \( O(n \log n) \)
- quadratic: \( O(n^2) \)
- cubic: \( O(n^3) \)
- polynomial: \( O(n^k) \) \( (k \text{ is a constant}) \)
- exponential: \( O(c^n) \) \( (c \text{ is a constant } > 1) \)
Meet the Family

- $O( f(n) )$ is the set of all functions asymptotically less than or equal to $f(n)$
  - $o( f(n) )$ is the set of all functions asymptotically strictly less than $f(n)$

- $\Omega( f(n) )$ is the set of all functions asymptotically greater than or equal to $f(n)$
  - $\omega( f(n) )$ is the set of all functions asymptotically strictly greater than $f(n)$

- $\Theta( f(n) )$ is the set of all functions asymptotically equal to $f(n)$
Meet the Family, Formally

• $g(n) \in O(f(n))$ iff
  There exist $c$ and $n_0$ such that $g(n) \leq c f(n)$ for all $n \geq n_0$
  - $g(n) \in o(f(n))$ iff
    There exists a $n_0$ such that $g(n) < c f(n)$ for all $c$ and $n \geq n_0$
    Equivalent to: $\lim_{n \to \infty} g(n)/f(n) = 0$

• $g(n) \in \Omega(f(n))$ iff
  There exist $c$ and $n_0$ such that $g(n) \geq c f(n)$ for all $n \geq n_0$
  - $g(n) \in \omega(f(n))$ iff
    There exists a $n_0$ such that $g(n) > c f(n)$ for all $c$ and $n \geq n_0$
    Equivalent to: $\lim_{n \to \infty} g(n)/f(n) = \infty$

• $g(n) \in \Theta(f(n))$ iff
  $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$
## Big-Omega et al. Intuitively

<table>
<thead>
<tr>
<th>Asymptotic Notation</th>
<th>Mathematics Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Θ</td>
<td>=</td>
</tr>
<tr>
<td>Ω</td>
<td>≥</td>
</tr>
<tr>
<td>O</td>
<td>≤</td>
</tr>
<tr>
<td>ω</td>
<td>&gt;</td>
</tr>
<tr>
<td>o</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
Pros and Cons
of Asymptotic Analysis
Perspective: Kinds of Analysis

• Running time may depend on actual data input, not just length of input
• Distinguish
  – Worst Case
    • Your worst enemy is choosing input
  – Best Case
  – Average Case
    • Assumes some probabilistic distribution of inputs
  – Amortized
    • Average time over many operations
Types of Analysis

Two orthogonal axes:

- **Bound Flavor**
  - Upper bound ($O, o$)
  - Lower bound ($\Omega, \omega$)
  - Asymptotically tight ($\theta$)

- **Analysis Case**
  - Worst Case (Adversary)
  - Average Case
  - Best Case
  - Amortized
$16n^3\log_8(10n^2) + 100n^2 = O(n^3\log n)$

- Eliminate low-order terms

$16n^3\log_8(10n^2) + 100n^2$
$\rightarrow 16n^3\log_8(10n^2)$
$\rightarrow n^3\log_8(10n^2)$
$\rightarrow n^3(\log_8(10) + \log_8(n^2))$
$\rightarrow n^3\log_8(10) + n^3\log_8(n^2)$
$\rightarrow n^3\log_8(n^2)$
$\rightarrow 2n^3\log_8(n)$
$\rightarrow n^3\log_8(n)$
$\rightarrow n^3\log_8(2)\log(n)$
$\rightarrow n^3\log(n)/3$
$\rightarrow n^3\log(n)$

- Eliminate constant coefficients
• Should be started on Homework 1

• Priority Queues and Heaps up Next (relevant to Project 2)