The Time Complexity of an Algorithm

Specifies how the running time depends on the size of the input.
Purpose

- To estimate how long a program will run.
- To estimate the largest input that can reasonably be given to the program.
- To compare the efficiency of different algorithms.
- To help focus on the parts of code that are executed the largest number of times.
- To choose an algorithm for an application.
Time Complexity Is a Function

• Specifies how the running time depends on the size of the input.

• A function mapping “size” of input $\Rightarrow$ “time” $T(n)$ executed.
Example: Insertion Sort
Example: Insertion Sort

(a) \[\begin{array}{ccccccc}
5 & 2 & 4 & 6 & 1 & 3 \\
\end{array}\]

(b) \[\begin{array}{ccccccc}
2 & 5 & 4 & 6 & 1 & 3 \\
\end{array}\]

(c) \[\begin{array}{ccccccc}
2 & 4 & 5 & 6 & 1 & 3 \\
\end{array}\]

(d) \[\begin{array}{ccccccc}
2 & 4 & 5 & 6 & 1 & 3 \\
\end{array}\]

(e) \[\begin{array}{ccccccc}
1 & 2 & 4 & 5 & 6 & 3 \\
\end{array}\]

(f) \[\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}\]
Example: Insertion Sort

**INSERTION-SORT(A)**

1. for $j \leftarrow 2$ to $\text{length}[A]$
2. \hspace{1em} do $\text{key} \leftarrow A[j]$
3. \hspace{1em} $\triangleright$ Insert $A[j]$ into the sorted sequence $A[1 \ldots j-1]$.
4. \hspace{1em} $i \leftarrow j - 1$
5. \hspace{1em} while $i > 0$ and $A[i] > \text{key}$
6. \hspace{2em} do $A[i+1] \leftarrow A[i]$
7. \hspace{1em} $i \leftarrow i - 1$
8. $A[i+1] \leftarrow \text{key}$

<table>
<thead>
<tr>
<th>cost</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$n$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$\sum_{j=2}^{n} t_j$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>$\sum_{j=2}^{n} (t_j - 1)$</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$\sum_{j=2}^{n} (t_j - 1)$</td>
</tr>
<tr>
<td>$c_8$</td>
<td>$n - 1$</td>
</tr>
</tbody>
</table>

Worst case (reverse order): $t_j = j : \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \rightarrow T(n) \in \theta(n^2)$
Example: Merge Sort

sorted sequence

1 2 2 3 4 5 6 7

merge

2 4 5 7

merge

2 5

merge

5 2

merge

4 7

merge

1 3

merge

1 3

merge

2 6

merge

2 6

initial sequence
**Merge-Sort** \((A, p, r)\)

1. if \(p < r\)
2. \(\text{then } q \leftarrow \lfloor (p + r) / 2 \rfloor\)
3. \(\text{Merge-Sort}(A, p, q)\)
4. \(\text{Merge-Sort}(A, q + 1, r)\)
5. \(\text{Merge}(A, p, q, r)\)
Example: Merge Sort

\texttt{MERGE}(A, p, q, r)

1. \hspace{1em} n_1 \gets q - p + 1
2. \hspace{1em} n_2 \gets r - q
3. \hspace{1em} \text{create arrays } L[1\ldots n_1 + 1] \text{ and } R[1\ldots n_2 + 1]
4. \hspace{1em} \textbf{for } i \gets 1 \textbf{ to } n_1
5. \hspace{2em} \textbf{do } L[i] \gets A[p + i - 1]
6. \hspace{1em} \textbf{for } j \gets 1 \textbf{ to } n_2
7. \hspace{2em} \textbf{do } R[j] \gets A[q + j]
8. \hspace{1em} L[n_1 + 1] \gets \infty
9. \hspace{1em} R[n_2 + 1] \gets \infty
10. \hspace{1em} i \gets 1
11. \hspace{1em} j \gets 1
12. \hspace{1em} \textbf{for } k \gets p \textbf{ to } r
13. \hspace{2em} \textbf{do if } L[i] \leq R[j]
14. \hspace{3em} \textbf{then } A[k] \gets L[i]
15. \hspace{3em} \hspace{1em} i \gets i + 1
16. \hspace{3em} \textbf{else } A[k] \gets R[j]
17. \hspace{3em} \hspace{1em} j \gets j + 1
Example: Merge Sort

\[ T(n) \in \theta(n \log n) \]
Relevant Mathematics: Classifying Functions
Assumed Knowledge

See J. Edmonds, How to Think About Algorithms (HTA):
http://www.cse.yorku.ca/~jeff/courses/3101/

Chapter 22. Existential and Universal Quantifiers

\[ \exists g \forall b \text{ Loves}(b, g) \]

\[ \forall g \exists b \text{ Loves}(b, g) \]

Chapter 24. Logarithms and Exponentials
Classifying Functions

Describing how fast a function grows without going into too much detail.
Which are more alike?
Which are more alike?

Mammals
Which are more alike?
Which are more alike?

Dogs
Classifying Animals

Vertebrates

- Birds
- Mammals
  - Dogs
  - Giraffe
- Reptiles
- Fish
Classifying Functions

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

Note: The universe is estimated to contain $\sim 10^{80}$ particles.
Which are more alike?

\[ n^{1000}, \quad n^2, \quad 2^n \]
End of Lecture 1

Wed, March 4, 2009
Which are more alike?

Polynomials

\[ n^{1000} \quad n^2 \quad 2^n \]
Which are more alike?

1000n^2  3n^2  2n^3
Which are more alike?

1000n^2  3n^2  2n^3

Quadratic
Classifying Functions?
Classifying Functions

Functions

- Constant
- Logarithmic
- Poly Logarithmic
- Polynomial
- Exponential
- Double Exponential

Values:
- $5$
- $5 \log n$
- $(\log n)^5$
- $n^5$
- $2^{5n}$
- $2^{n^5}$
- $2^{2^{5n}}$
Classifying Functions?

Polynomial
Classifying Functions

Polynomial

Linear
  - $5n$
Quadratic
  - $5n^2$
Cubic
  - $5n^3$
4th Order
  - $5n^4$
Classifying Functions

Functions

- Constant
- Logarithmic
- Poly Logarithmic
- Polynomial
- Exponential
- Double Exponential

Examples:
- $5$
- $5 \log n$
- $(\log n)^5$
- $n^5$
- $2^{5n}$
- $2^{n^5}$
- $2^{2^{5n}}$
Properties of the Logarithm

Changing bases:  \[ \log_a n = \frac{1}{\log_b a} \times \log_b n \]

Swapping base and argument:  \[ \log_a b = \frac{1}{\log_b a} \]
Logarithmic

- $\log_{10} n = \# \text{ digits to write } n$
- $\log_{2} n = \# \text{ bits to write } n$
  - $= 3.32 \log_{10} n$
- $\log(n^{1000}) = 1000 \log(n)$

Differ only by a multiplicative constant.

Changing bases:  
$$\log_{a} n = \frac{1}{\log_{b} a} \times \log_{b} n$$
Classifying Functions

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

\((\log n)^5 = \log^5 n\)
(Poly)Logarithmic $\ll$ Polynomial

$log_{1000} n \ll n^{0.001}$

For sufficiently large $n$
Classifying Functions

Polynomial

- Linear: $5n$
- Quadratic: $5n^2$
- Cubic: $5n^3$
- 4th Order: $5n^4$
Linear $\ll$ Quadratic

$10000 \, n \ll 0.0001 \, n^2$

For sufficiently large $n$
Classifying Functions

Functions

- Constant
- Logarithmic
- Poly Logarithmic
- Polynomial
- Exponential
- Exp
- Double Exponential

Expressions:
- $5$
- $5 \log n$
- $(\log n)^5$
- $n^5$
- $2^{5n}$
- $2^{n^5}$
- $2^{2^{5n}}$
Polynomial $\ll$ Exponential

$n^{1000} \ll 2^{0.001n}$

For sufficiently large $n$
Ordering Functions

Functions

Constant << Logarithmic << Poly Logarithmic << Polynomial << Exponential << Exp << Double Exponential

5 << 5 log n << (log n)^5 << n^5 << 2^{5n} << 2^{n^5} << 2^{2^{5n}}
Which Functions are Constant?

Yes • 5
Yes • 1,000,000,000,000
Yes • 0.000000000000001
Yes • -5
Yes • 0
No • 8 + sin(n)
Which Functions are “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 • 0.0000000000000001
No • -5
No • 0
Yes • 8 + sin(n)

Lies in between
Which Functions are Quadratic?

- $n^2$
- $\ldots$ $?$
Which Functions are Quadratic?

- \( n^2 \)
- \( 0.001 \, n^2 \)
- \( 1000 \, n^2 \)

Some constant times \( n^2 \).
Which Functions are Quadratic?

- $n^2$
- $0.001 \, n^2$
- $1000 \, n^2$
- $5n^2 + 3n + 2\log n$
Which Functions are Quadratic?

- $n^2$
- $0.001 \cdot n^2$
- $1000 \cdot n^2$
- $5n^2 + 3n + 2\log n$

Lies in between
Which Functions are Quadratic?

- \( n^2 \)
- \( 0.001 \, n^2 \)
- \( 1000 \, n^2 \)
- \( 5n^2 + 3n + 2\log n \)

Ignore low-order terms
Ignore multiplicative constants.
Ignore "small" values of \( n \).
Write \( \Theta(n^2) \).
Which Functions are Polynomial?

- $n^5$
- $\ldots$ ?
Which Functions are Polynomial?

- $n^c$
- $n^{0.0001}$
- $n^{10000}$

$n$ to some constant power.
Which Functions are Polynomial?

- $n^c$
- $n^{0.0001}$
- $n^{10000}$
- $5n^2 + 8n + 2\log n$
- $5n^2 \log n$
- $5n^{2.5}$
Which Functions are Polynomial?

- $n^c$
- $n^{0.0001}$
- $n^{10000}$
- $5n^2 + 8n + 2\log n$
- $5n^2 \log n$
- $5n^{2.5}$

Lie in between
Which Functions are Polynomials?

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

Ignore low-order terms
Ignore power constant.
Ignore "small" values of n.
Write $n^{\theta(1)}$
Which Functions are Exponential?

- $2^n$
- ... ?
Which Functions are Exponential?

- $2^n$
- $2^{0.0001n}$
- $2^{10000n}$

2 raised to a linear function of $n$. 
Which Functions are Exponential?

- \(2^n\)
- \(2^{0.0001 \times n}\)
- \(2^{10000 \times n}\)
- \(8^n\)
- \(\frac{2^n}{n^{100}}\)
- \(2^n \times n^{100}\)

too small? too big?
Which Functions are Exponential?

- $2^n$
- $2^{0.0001n}$
- $2^{10000n}$
- $8^n$
- $2^n / n^{100}$
- $2^n \cdot n^{100}$

$\Rightarrow 2^{3n}$
$\Rightarrow 2^{0.5n}$
$\Rightarrow 2^{2n}$

Lie in between
Which Functions are Exponential?

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

- $2^{0.5n} > n^{100}$
- $2^n = 2^{0.5n} \cdot 2^{0.5n} > n^{100} \cdot 2^{0.5n}$
- $2^n / n^{100} > 2^{0.5n}$
Which Functions are Exponential?

- $2^n$
- $2^{0.0001 \cdot n}$
- $2^{10000 \cdot n}$
- $8^n$
- $2^n / n^{100}$
- $2^n \cdot n^{100}$

- Ignore low-order terms
- Ignore base.
- Ignore "small" values of n.
- Ignore polynomial factors.
- Write $2^{\theta(n)}$
Classifying Functions

\[ f(n) = 8 \cdot 2^{4n} / n^{100} + 5 \cdot n^3 \]

Tell me the most significant thing about your function

Rank constants in significance.
f(n) = 8 \cdot 2^{4n} / n^{100} + 5 \cdot n^3

Tell me the most significant thing about your function

2^{\Theta(n)}

because $2^{3n} < f(n) < 2^{4n}$
Classifying Functions

$f(n) = 8 \cdot 2^{4n} / n^{100} + 5 \cdot n^3$

Tell me the most significant thing about your function

$2^{\Theta(n)}$

$2^{4n} / n^{\Theta(1)}$

$\Theta(2^{4n} / n^{100})$

$8 \cdot 2^{4n} / n^{100} + n^{\Theta(1)}$

$8 \cdot 2^{4n} / n^{100} + \Theta(n^3)$

$8 \cdot 2^{4n} / n^{100} + 5 \cdot n^3$
# Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theta</strong></td>
<td>$f(n) = \theta(g(n))$</td>
<td>$f(n) \approx c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Big Oh</strong></td>
<td>$f(n) = O(g(n))$</td>
<td>$f(n) \leq c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Big Omega</strong></td>
<td>$f(n) = \Omega(g(n))$</td>
<td>$f(n) \geq c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Little Oh</strong></td>
<td>$f(n) = o(g(n))$</td>
<td>$f(n) &lt;&lt; c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Little Omega</strong></td>
<td>$f(n) = \omega(g(n))$</td>
<td>$f(n) &gt;&gt; c \cdot g(n)$</td>
</tr>
</tbody>
</table>
Definition of Theta

\[ f(n) = \theta(g(n)) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]
Definition of Theta

\[ f(n) = \theta(g(n)) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

f(n) is sandwiched between \( c_1 g(n) \) and \( c_2 g(n) \)
Definition of Theta

\[ f(n) = \theta(g(n)) \]

There exists \( c_1, c_2, n_0 > 0 \) such that for all \( n \geq n_0 \),
\[ c_1 g(n) \leq f(n) \leq c_2 g(n) \]

f(n) is sandwiched between \( c_1 g(n) \) and \( c_2 g(n) \)

for some sufficiently small \( c_1 = 0.0001 \)

for some sufficiently large \( c_2 = 1000 \)
Definition of Theta

\[ f(n) = \theta(g(n)) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

For all sufficiently large \( n \)
Definition of Theta

\[ f(n) = \theta(g(n)) \]

For all sufficiently large \( n \)

For some definition of “sufficiently large”
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ c_1 \cdot n^2 \leq 3n^2 + 7n + 8 \leq c_2 \cdot n^2 \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2 \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot 1^2 \leq 3 \cdot 1^2 + 7 \cdot 1 + 8 \leq 4 \cdot 1^2 \]

False
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 7 \quad 3 \cdot 7^2 \leq 3 \cdot 7^2 + 7 \cdot 7 + 8 \leq 4 \cdot 7^2 \]

False
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot 8^2 \leq 3 \cdot 8^2 + 7 \cdot 8 + 8 \leq 4 \cdot 8^2 \]

True
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot 9^2 \leq 3 \cdot 9^2 + 7 \cdot 9 + 8 \leq 4 \cdot 9^2 \]

True
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot 10^2 \leq 3 \cdot 10^2 + 7 \cdot 10 + 8 \leq 4 \cdot 10^2 \]

True
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2 \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \Theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2 \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ n \geq 8 \]

\[ 3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2 \]

True

COSC 3101, PROF. J. ELDER
Definition of \( \Theta \)

\[
3n^2 + 7n + 8 = \Theta(n^2)
\]

\[
\exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)
\]

\[
3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2
\]

\[
n \geq 8
\]

\[
7n + 8 \leq 1 \cdot n^2
\]

\[
7 + \frac{8}{n} \leq 1 \cdot n
\]

True
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n^2) \]

True

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ n \geq 8 \quad 3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2 \]
### Asymptotic Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Formula</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theta</strong></td>
<td>$f(n) = \theta(g(n))$</td>
<td>$f(n) \approx c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>BigOh</strong></td>
<td>$f(n) = O(g(n))$</td>
<td>$f(n) \leq c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Omega</strong></td>
<td>$f(n) = \Omega(g(n))$</td>
<td>$f(n) \geq c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Little Oh</strong></td>
<td>$f(n) = o(g(n))$</td>
<td>$f(n) &lt;&lt; c \cdot g(n)$</td>
</tr>
<tr>
<td><strong>Little Omega</strong></td>
<td>$f(n) = \omega(g(n))$</td>
<td>$f(n) &gt;&gt; c \cdot g(n)$</td>
</tr>
</tbody>
</table>
Definition of \( \Theta \)

\[ 3n^2 + 7n + 8 = \Theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ c_1 \cdot n \leq 3n^2 + 7n + 8 \leq c_2 \cdot n \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot n \leq 3n^2 + 7n + 8 \leq 100 \cdot n \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \cdot n \leq 3n^2 + 7n + 8 \leq 100 \cdot n \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 100 \quad 3 \cdot 100 \leq 3 \cdot 100^2 + 7 \cdot 100 + 8 \leq 100 \cdot 100 \]

False
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 3 \leq 10,000 \]

\[ 3 \cdot n \leq 3n^2 + 7n + 8 \leq 10,000 \cdot n \]
Definition of Theta

\[ 3n^2 + 7n + 8 = \theta(n) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ 10,000 \quad 3 \cdot 10,000 \leq 3 \cdot 10,000^2 + 7 \cdot 10,000 + 8 \leq 10,000 \cdot 10,000 \]

False
Definition of Theta

\[ 3n^2 + 7n + 8 \not= \Theta(n) \]

What is the reverse statement?

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]
Understand Quantifiers!!!

∀ b, Loves(b, MM)  →  [∀ b, Loves(b, MM)]
¬[∃ b, ¬Loves(b, MM)]  →  [∃ b, ¬Loves(b, MM)]
∃ b, ¬Loves(b, MM)
Definition of Theta

\[ 3n^2 + 7n + 8 \neq \theta(n) \]

The reverse statement

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) < f(n) \leq c_2 g(n) \]

\[ \forall c_1, c_2, n_0 > 0 : \exists n \geq n_0 : f(n) < c_1 g(n) \text{ or } f(n) > c_2 g(n) \]
Definition of Theta

\[ 3n^2 + 7n + 8 \neq \Theta(n) \]

\[ \forall c_1, c_2, n_0 > 0 : \exists n \geq n_0 : f(n) < c_1 g(n) \text{ or } f(n) > c_2 g(n) \]

\[ 3n^2 + 7n + 8 > c_2 n \]

\[ \leftrightarrow 3 + \frac{7}{n} + \frac{8}{n^2} > \frac{c_2}{n} \]

Satisfied for \( n = \max(n_0, c_2) \)
Order of Quantifiers

\[ f(n) = \Theta(g(n)) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ \exists n_0 > 0 : \forall n \geq n_0, \exists c_1, c_2 : c_1 g(n) \leq f(n) \leq c_2 g(n) \]
Understand Quantifiers!!

\[ \exists g, \forall b, \text{loves}(b, g) \]

One girl

- Sam
- Bob
- John
- Fred

Mike

\[ \forall b, \exists g, \text{loves}(b, g) \]

Could be a separate girl for each boy.

- Sam
- Bob
- John
- Fred

Mary

Beth

Marilin

Monro

Ann
Order of Quantifiers

\[ f(n) = \Theta(g(n)) \]

\[ \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \]

\[ \exists n_0 > 0 : \forall n \geq n_0, \exists c_1, c_2 : c_1 g(n) \leq f(n) \leq c_2 g(n) \]

No! It cannot be a different \( c_1 \) and \( c_2 \) for each \( n \).
### Other Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition 1</th>
<th>Definition 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theta</td>
<td>$f(n) = \theta(g(n))$</td>
<td>$f(n) \approx c \cdot g(n)$</td>
</tr>
<tr>
<td>BigOh</td>
<td>$f(n) = O(g(n))$</td>
<td>$f(n) \leq c \cdot g(n)$</td>
</tr>
<tr>
<td>Omega</td>
<td>$f(n) = \Omega(g(n))$</td>
<td>$f(n) \geq c \cdot g(n)$</td>
</tr>
<tr>
<td>Little Oh</td>
<td>$f(n) = o(g(n))$</td>
<td>$f(n) \ll c \cdot g(n)$</td>
</tr>
<tr>
<td>Little Omega</td>
<td>$f(n) = \omega(g(n))$</td>
<td>$f(n) \gg c \cdot g(n)$</td>
</tr>
</tbody>
</table>
Definition of “Big Omega”

\[ f(n) \in \Omega(g(n)) \]

\[ \exists c, n_0 > 0 : \forall n \geq n_0, f(n) \geq cg(n) \]
Definition of “Big Oh”

\[ f(n) \in O(g(n)) \]

\[ \exists c, n_0 > 0 : \forall n \geq n_0, f(n) \leq c g(n) \]
Definition of “Little Omega”

\[ f(n) \in \omega(g(n)) \]

\[ \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, f(n) > cg(n) \]
Definition of “Little Oh”

\[ f(n) \in o(g(n)) \]

\[ \forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, f(n) < cg(n) \]
Time Complexity Is a Function

• Specifies how the running time depends on the size of the input.

• A function mapping “size” of input $\rightarrow$ “time” $T(n)$ executed.
Definition of Time?
Definition of Time

- # of seconds (machine dependent).
- # lines of code executed.
- # of times a specific operation is performed (e.g., addition).

- These are all reasonable definitions of time, because they are within a constant factor of each other.
Definition of Input Size

- A good definition: #bits
- Examples:

  1. Sorting an array $A$ of $k$-bit integers.
     e.g., $k=16$.
     Input size $n = k \times \text{length}(A)$
     Note that, since $n \propto \text{length}(A)$, sufficient to define $n = \text{length}(A)$.

  2. Multiplying $A, B \in \mathbb{N}$.
     Let $n_A = \#\text{bits used to represent } A$
     Let $n_B = \#\text{bits used to represent } B$
     Then input size $n = n_A + n_B$
     Note that $n_A \approx \log_2 A$ and $n_B \approx \log_2 B$
     Thus input size $n \approx \log_2 A + \log_2 B$
Time Complexity Is a Function

• Specifies how the running time depends on the size of the input.

• A function mapping:
  – “size” of input $\rightarrow$ “time” $T(n)$ executed.

• Or more precisely:
  – # of bits $n$ needed to represent the input $\rightarrow$ # of operations $T(n)$ executed.
Time Complexity of an Algorithm

The time complexity of an algorithm is the largest time required on any input of size n. (Worst case analysis.)

- $O(n^2)$: For any input size $n>n_0$, the algorithm takes no more than $cn^2$ time on every input.
- $\Omega(n^2)$: For any input size $n>n_0$, the algorithm takes at least $cn^2$ time on at least one input.
- $\Theta(n^2)$: Do both.
End of Lecture 2

Monday, Mar 9, 2009
What is the height of tallest person in the class?

Bigger than this?

Smaller than this?

Need to find only one person who is taller

Need to look at every person
Time Complexity of a Problem

The time complexity of a problem is the time complexity of the *fastest* algorithm that solves the problem.

- $\mathcal{O}(n^2)$: Provide an algorithm that solves the problem in no more than this time.
  - Remember: for every input, i.e. worst case analysis!
- $\Omega(n^2)$: Prove that no algorithm can solve it faster.
  - Remember: only need one input that takes at least this long!
- $\Theta(n^2)$: Do both.