Lecture 3: Analysing Complexity of Algorithms
Big Oh, Big Omega, and Big Theta Notation

Georgy Gimel’farb

COMPSCI 220 Algorithms and Data Structures
1. Complexity
2. Basic tools
3. Big-Oh
4. Big Omega
5. Big Theta
6. Examples
## Typical Complexity Curves

Running time $T(n)$ is proportional to:

<table>
<thead>
<tr>
<th>$T(n)$</th>
<th>Complexity:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) \propto \log n$</td>
<td>logarithmic</td>
</tr>
<tr>
<td>$T(n) \propto n$</td>
<td>linear</td>
</tr>
<tr>
<td>$T(n) \propto n \log n$</td>
<td>linearithmic</td>
</tr>
<tr>
<td>$T(n) \propto n^2$</td>
<td>quadratic</td>
</tr>
<tr>
<td>$T(n) \propto n^3$</td>
<td>cubic</td>
</tr>
<tr>
<td>$T(n) \propto n^k$</td>
<td>polynomial</td>
</tr>
<tr>
<td>$T(n) \propto 2^n$</td>
<td>exponential</td>
</tr>
<tr>
<td>$T(n) \propto k^n; k &gt; 1$</td>
<td>exponential</td>
</tr>
</tbody>
</table>
Separating an Algorithm Itself from Its Implementation

Two concepts to separate an algorithm from implementation:

- The input data size $n$, or the number of individual data items in a single data instance to be processed.
- The number of elementary operations $f(n)$ taken by an algorithm, or its running time.

The running time of a program implementation: $cf(n)$

- The constant factor $c$ can rarely be determined and depends on a computer, operating system, language, compiler, etc.

When the input size increases from $n = n_1$ to $n = n_2$, all other factors being equal, the relative running time of the program increases by a factor of $\frac{T(n_2)}{T(n_1)} = \frac{cf(n_1)}{cf(n_2)} = \frac{f(n_1)}{f(n_2)}$. 
Relative Growth $g(n) = \frac{f(n)}{f(5)}$ of Running Time

The approximate running time for large input sizes gives enough information to distinguish between a good and a bad algorithm.

<table>
<thead>
<tr>
<th>Function $f(n)$</th>
<th>Input size $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Constant</td>
<td>1</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>$\log_5 n$</td>
</tr>
<tr>
<td>Linear</td>
<td>$n$</td>
</tr>
<tr>
<td>Linearithmic</td>
<td>$n \log_5 n$</td>
</tr>
<tr>
<td>Quadratic</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Cubic</td>
<td>$n^3$</td>
</tr>
<tr>
<td>Exponential</td>
<td>$2^n$</td>
</tr>
</tbody>
</table>
Big-Oh, Big-Theta, and Big-Omega Tools

Let $f(n)$ and $g(n)$ be nonnegative-valued functions defined on nonnegative integers $n$.

Math notation for “of the order of . . .” or “roughly proportional to . . .”:

- Big-Oh (actually: Big-Omicron) $O(\ldots)$ $\Rightarrow$ $g(n) = O(f(n))$
- Big-Theta $\Theta(\ldots)$ $\Rightarrow$ $g(n) = \Theta(f(n))$
- Big-Omega $\Omega(\ldots)$ $\Rightarrow$ $g(n) = \Omega(f(n))$

**Big Oh $O(\ldots)$ – Formal Definition**

The function $g(n)$ is $O(f(n))$ (read: $g(n)$ is Big Oh of $f(n)$) **iff** there exists a positive real constant $c$ and a positive integer $n_0$ such that $g(n) \leq cf(n)$ for all $n > n_0$.

The notation **iff** abbreviates “if and only if”.
Example 1.8; p.13: \( g(n) = 100 \log_{10} n \) is \( O(n) \)

\[
g(n) < n \text{ if } n > 238 \text{ or } g(n) < 0.3n \text{ if } n > 1000.
\]

By definition, \( g(n) \) is \( O(f(n)) \), or \( g(n) = O(f(n)) \) if a constant \( c > 0 \) exists, such that \( cf(n) \) grows faster than \( g(n) \) for all \( n > n_0 \).

- To prove that some \( g(n) \) is \( O(f(n)) \) means to show that for \( g \) and \( f \) such constants \( c \) and \( n_0 \) exist.
- The constants \( c \) and \( n_0 \) are interdependent.
- \( g(n) \) is \( O(f(n)) \) iff the graph of \( g(n) \) is always below or at the graph of \( cf(n) \) after \( n_0 \).
**Big-Oh $O(\ldots)$: Informal Meaning**

$O(f(n))$ generalises an **asymptotic upper bound**.

- If $g(n)$ is $O(f(n))$, an algorithm with running time $g(n)$ runs **asymptotically**, i.e. for large $n$, **at most** as fast, to within a constant factor, as an algorithm with running time $f(n)$.
  - In other words, $g(n)$ for large $n$ may approach $cf(n)$ closer and closer while the relationship $g(n) \leq cf(n)$ holds for $n > n_0$.
  - The scaling factor $c$ and the threshold $n_0$ are interdependent and differ for different particular functions $g(n)$ in $O(f(n))$.
- Notations $g(n) = O(f(n))$ or $g(n)$ is $O(f(n))$ mean actually $g(n) \in O(f(n))$.
  - The notation $g(n) \in O(f(n))$ indicates that $g(n)$ is a member of the set $O(f(n))$ of functions.
  - All the functions in the set $O(f(n))$ are increasing with the same or the lesser rate as $f(n)$ when $n \to \infty$. 
**Big-Oh** $O(\ldots)$: Informal Meaning

$O(f(n))$ generalises an **asymptotic upper bound**.

- If $g(n)$ is $O(f(n))$, an algorithm with running time $g(n)$ runs **asymptotically**, i.e. for large $n$, **at most** as fast, to within a constant factor, as an algorithm with running time $f(n)$.
  - In other words, $g(n)$ for large $n$ may approach $cf(n)$ closer and closer while the relationship $g(n) \leq cf(n)$ holds for $n > n_0$.
  - The scaling factor $c$ and the threshold $n_0$ are interdependent and differ for different particular functions $g(n)$ in $O(f(n))$.
- Notations $g(n) = O(f(n))$ or $g(n)$ is $O(f(n))$ mean actually $g(n) \in O(f(n))$.
  - The notation $g(n) \in O(f(n))$ indicates that $g(n)$ is a member of the set $O(f(n))$ of functions.
  - All the functions in the set $O(f(n))$ are increasing with the same or the lesser rate as $f(n)$ when $n \rightarrow \infty$.  

Big Omega $\Omega(\ldots)$

The function $g(n)$ is $\Omega(f(n))$ 
iff there exists a positive real constant $c$ and a positive integer $n_0$ such that $g(n) \geq cf(n)$ for all $n > n_0$.

- $\Omega(\ldots)$ is complementary to $O(\ldots)$.
- It generalises the concept of “lower bound” ($\geq$) in the same way as $O(\ldots)$ generalises the concept of “upper bound” ($\leq$): if $g(n)$ is $\Omega(f(n))$ then $f(n)$ is $O(g(n))$.
- **Example 1**: $5n^2$ is $\Omega(n)$ because $5n^2 \geq 5n$ for $n \geq 1$.
- **Example 2**: $0.01n$ is $\Omega(\log n)$ because $0.01n \geq 0.5 \log_{10} n$ for $n \geq 100$. 
The function $g(n)$ is $\Omega(f(n))$ iff there exists a positive real constant $c$ and a positive integer $n_0$ such that $g(n) \geq cf(n)$ for all $n > n_0$.

- $\Omega(\ldots)$ is complementary to $O(\ldots)$.
- It generalises the concept of “lower bound” ($\geq$) in the same way as $O(\ldots)$ generalises the concept of “upper bound” ($\leq$): if $g(n)$ is $\Omega(f(n))$ then $f(n)$ is $O(g(n))$.

- **Example 1**: $5n^2$ is $\Omega(n)$ because $5n^2 \geq 5n$ for $n \geq 1$.
- **Example 2**: $0.01n$ is $\Omega(\log n)$ because $0.01n \geq 0.5\log_{10} n$ for $n \geq 100$. 
The function $g(n)$ is $\Omega(f(n))$ iff there exists a positive real constant $c$ and a positive integer $n_0$ such that $g(n) \geq cf(n)$ for all $n > n_0$.

- $\Omega(\ldots)$ is complementary to $O(\ldots)$.
- It generalises the concept of “lower bound” ($\geq$) in the same way as $O(\ldots)$ generalises the concept of “upper bound” ($\leq$): if $g(n)$ is $\Omega(f(n))$ then $f(n)$ is $O(g(n))$.
- **Example 1:** $5n^2$ is $\Omega(n)$ because $5n^2 \geq 5n$ for $n \geq 1$.
- **Example 2:** $0.01n$ is $\Omega(\log n)$ because $0.01n \geq 0.5 \log_{10} n$ for $n \geq 100$. 
Big Theta Θ(…)

The function $g(n)$ is $\Theta(f(n))$

iff there exists two positive real constants $c_1$ and $c_2$ and a positive integer $n_0$ such that $c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n > n_0$.

- If $g(n)$ is $\Theta(f(n))$ then $g(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$ or, what is the same, $g(n)$ is $O(f(n))$ and $g(n)$ is $\Omega(f(n))$:
  - $g(n)$ is $O(f(n)) \rightarrow g(n) \leq c' f(n)$ for $n > n'$.
  - $g(n)$ is $\Omega(f(n)) \rightarrow f(n) \leq c'' g(n)$ for $n > n''$.
  - $g(n)$ is $\Theta(f(n)) \leftrightarrow c'' = \frac{1}{c_1}; c' = c_2$, and $n_0 = \max\{n', n''\}$.
- Informally, if $g(n)$ is $\Theta(f(n))$ then both the functions have the same rate of increase.
- **Example:** the same rate of increase for $g(n) = n + 5n^{0.5}$ and $f(n) = n$ because $n \leq n + 5n^{0.5} \leq 6n$ for $n > 1$. 
**Big Theta \( \Theta(\ldots) \)**

The function \( g(n) \) is \( \Theta(f(n)) \) iff there exists two positive real constants \( c_1 \) and \( c_2 \) and a positive integer \( n_0 \) such that \( c_1 f(n) \leq g(n) \leq c_2 f(n) \) for all \( n > n_0 \).

- If \( g(n) \) is \( \Theta(f(n)) \) then \( g(n) \) is \( O(f(n)) \) and \( f(n) \) is \( O(g(n)) \) or, what is the same, \( g(n) \) is \( O(f(n)) \) and \( g(n) \) is \( \Omega(f(n)) \):
  - \( g(n) \) is \( O(f(n)) \) \( \rightarrow \) \( g(n) \leq c' f(n) \) for \( n > n' \).
  - \( g(n) \) is \( \Omega(f(n)) \) \( \rightarrow \) \( f(n) \leq c'' g(n) \) for \( n > n'' \).
  - \( g(n) \) is \( \Theta(f(n)) \) \( \leftarrow \) \( c'' = \frac{1}{c_1} \); \( c' = c_2 \), and \( n_0 = \max\{n', n''\} \).

- Informally, if \( g(n) \) is \( \Theta(f(n)) \) then both the functions have the same rate of increase.

- **Example:** the same rate of increase for \( g(n) = n + 5n^{0.5} \) and \( f(n) = n \) because \( n \leq n + 5n^{0.5} \leq 6n \) for \( n > 1 \).
The function $g(n)$ is $\Theta(f(n))$ iff there exists two positive real constants $c_1$ and $c_2$ and a positive integer $n_0$ such that $c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n > n_0$.

- If $g(n)$ is $\Theta(f(n))$ then $g(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$ or, what is the same, $g(n)$ is $O(f(n))$ and $g(n)$ is $\Omega(f(n))$:
  - $g(n)$ is $O(f(n)) \rightarrow g(n) \leq c' f(n)$ for $n > n'$.
  - $g(n)$ is $\Omega(f(n)) \rightarrow f(n) \leq c'' g(n)$ for $n > n''$.
  - $g(n)$ is $\Theta(f(n)) \leftarrow c'' = \frac{1}{c_1}; c' = c_2$, and $n_0 = \max\{n', n''\}$.

- Informally, if $g(n)$ is $\Theta(f(n))$ then both the functions have the same rate of increase.

- **Example:** the same rate of increase for $g(n) = n + 5n^{0.5}$ and $f(n) = n$ because $n \leq n + 5n^{0.5} \leq 6n$ for $n > 1$. 
Comparisons: Two Crucial Ideas

- The exact running time function $g(n)$ is not very important since it can be multiplied by an arbitrary positive constant, $c$.

- The relative behaviour of two functions is compared only asymptotically, for large $n$, but not near the origin where it may make no sense.
  - If the constants $c$ involved are very large, the asymptotical behaviour loses practical interest!
  - In most cases, however, the constants remain fairly small.
  - To prove that $g(n)$ is not $O(f(n))$, $\Omega(f(n))$, or $\Theta(f(n))$, one has to show that the desired constants do not exist, i.e. lead to a contradiction.
  - $g(n)$ and $f(n)$ in the Big-Oh, -Omega, and -Theta definitions mostly relate, respectively, to “exact” and rough approximate (like $\log n$, $n$, $n^2$, etc) running time on inputs of size $n$. 
Example 1.11, p.14

Prove that linear function \( g(n) = an + b; \ a > 0 \), is \( O(n) \).

The proof: By the following chain of inequalities:
\[
g(n) \leq an + |b| \leq (a + |b|)n \text{ for all } n \geq 1
\]

Do not write \( O(2n) \) or \( O(an + b) \) as this means still \( O(n) \)!

\( O(n) \)-time:
\[
\begin{align*}
T(n) &= 3n + 1 & T(n) &= 10^8 + n \\
T(n) &= 50 + 10^{-8}n & T(n) &= 10^6n + 1
\end{align*}
\]

- Remember that “Big-Oh”, as well as “Big-Omega” and “Big-Theta”, describes an asymptotic behaviour for large problem sizes.
- Only the dominant terms as \( n \rightarrow \infty \) need to be shown as the argument of “Big-Oh”, “Big-Omega”, and “Big-Theta”.

Example 1.12, p.15

The polynomial $P_k(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$; $a_k > 0$, is $O(n^k)$.

The proof: $P_k(n) \leq (a_k + |a_{k-1}| + \ldots + |a_0|) n^k$ for $n \geq 1$.

- Do not write $O(P_k(n))$ as this means still $O(n^k)$!
- $O(n^k)$-time:
  \[
  \begin{align*}
  T(n) &= 3n^2 + 5n + 1 \text{ is } O(n^2) & \text{Is it also } O(n^3) ? \\
  T(n) &= 10^{-8} n^3 + 10^8 n^2 + 30 \text{ is } O(n^3) & \text{Is it also } \Omega(n^2) ? \\
  T(n) &= 10^{-8} n^8 + 1000 n + 1 \text{ is } O(n^8) & \text{Is it also } \Theta(n^8) ?
  \end{align*}
  \]

$T(n) = P_k(n)$ is
\[
\begin{align*}
O(n^m); & \quad m \geq k \\
\Theta(n^k); & \quad \Theta(n^k); \\
\Omega(n^m); & \quad m \leq k
\end{align*}
\]
The polynomial \( P_k(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0; \)
\( a_k > 0, \) is \( O(n^k). \)

**The proof:** \( P_k(n) \leq (a_k + |a_{k-1}| + \ldots + |a_0|) n^k \) for \( n \geq 1. \)

- **Do not write** \( O(P_k(n)) \) **as this means still** \( O(n^k)! \)

- **\( O(n^k) \)-time:**

\[
T(n) = 3n^2 + 5n + 1 \quad \text{is} \quad O(n^2) \quad \text{Is it also} \quad O(n^3)?
\]

\[
T(n) = 10^{-8} n^3 + 10^8 n^2 + 30 \quad \text{is} \quad O(n^3) \quad \text{Is it also} \quad \Omega(n^2)?
\]

\[
T(n) = 10^{-8} n^8 + 1000n + 1 \quad \text{is} \quad O(n^8) \quad \text{Is it also} \quad \Theta(n^8)?
\]

\[
T(n) = P_k(n) \quad \text{is} \quad \begin{cases} 
O(n^m); & m \geq k \\
\Theta(n^k); & \\
\Omega(n^m); & m \leq k 
\end{cases}
\]
Example 1.13, p.15

- The exponential function \( g(n) = 2^{n+k} \), where \( k \) is a constant, is \( O(2^n) \) because \( 2^{n+k} = 2^k 2^n \) for all \( n \).

- Generally, \( g(n) = m^{n+k} \) is \( O(l^n) \); \( l \geq m > 1 \), because \( m^{n+k} \leq l^{n+k} = l^k l^n \) for any constant \( k \).

A “brute-force” search for the best combination of \( n \) binary decisions by exhausting all the \( 2^n \) possible combinations has exponential time complexity!

- \( 2^{30} \approx 10^9 = 1,000,000,000 \) and
  \( 2^{40} \approx 10^{12} = 1,000,000,000,000 \)

- Therefore, try to find a more efficient way of solving the decision problem if \( n \geq 30 \ldots 40 \).
For each $m > 1$, the logarithmic function $g(n) = \log_m(n)$ has the same rate of increase as $\lg(n)$, i.e. $\log_2 n$, because
\[ \log_m(n) = \log_m(2) \lg(n) \text{ for all } n > 0. \]

Omit the logarithm base when using “Big-Oh”, “Big-Omega”, and “Big-Theta” notation: $\log_m n$ is $O(\log n)$, $\Omega(\log n)$, and $\Theta(\log n)$.

You will find later that the most efficient search for data in an ordered array has logarithmic time complexity.