BIG O!
Big-O notation Part I

COMP 250: Winter 2018
Lecture 11
Carlos G. Oliver

Slides adapted from M. Langer and M. Blanchette
Running time of selection sort

- We showed that running selection sort on an array of \( n \) elements takes in the worst case
  \( T(n) = 1 + 15n + 5n^2 \) primitive operations.
- When \( n \) is large, \( T(n) \approx 5n^2 \).
- When \( n \) is large,
  \[
  \frac{T(2n)}{T(n)} \approx \frac{5(2n)^2}{5n^2} \approx 4
  \]
  Doubling \( n \) quadruples \( T(n) \).

N.B. That is true for any coefficient of \( n^2 \) (not just 5).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( T(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>661</td>
</tr>
<tr>
<td>20</td>
<td>2301</td>
</tr>
<tr>
<td>30</td>
<td>4951</td>
</tr>
<tr>
<td>40</td>
<td>8601</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1000</td>
<td>5015001</td>
</tr>
<tr>
<td>2000</td>
<td>20030001</td>
</tr>
</tbody>
</table>
Towards a formal definition of big O

Let $t(n)$ be a function that describes the time it takes for some algorithm on input size $n$.

We would like to express how $t(n)$ grows with $n$, as $n$ becomes large i.e. \textit{asymptotic} behavior.

\textit{Unlike with limits,} we want to say that $t(n)$ grows like certain \textit{simpler} functions such as $\log_2 n$, $n$, $n^2$, $\ldots$, $2^n$, etc.
Preliminary Formal Definition

Let \( t(n) \) and \( g(n) \) be two functions, where \( n \geq 0 \). We say \( t(n) \) is asymptotically bounded above by \( g(n) \) if there exists \( n_0 \) such that, for all \( n \geq n_0 \),

\[
t(n) \leq g(n).
\]

This is not yet a formal definition of big O.
for all \( n_0 \geq n \), \( t(n) \leq g(n) \)
Example

\[ 6n \]

\[ 5n + 70 \]
Claim: $5n + 70$ is asymptotically bounded above by $6n$.

Proof:

(State definition) We want to show there exists an $n_0$ such that, for all $n \geq n_0$, $5n + 70 \leq 6n$. 
Claim: \(5n + 70\) is asymptotically bounded above by \(6n\).

Proof:

(State definition) We want to show there exists an \(n_0\) such that, for all \(n \geq n_0\), \(5n + 70 \leq 6n\).

\[
5n + 70 \leq 6n \\
\iff \\
70 \leq n
\]

Symbol “\(\iff\)” means “if and only if” i.e. logical equivalence.
Claim: $5n + 70$ is asymptotically bounded above by $6n$.

Proof:

(State definition) We want to show there exists an $n_0$ such that, for all $n \geq n_0$, $5n + 70 \leq 6n$.

\[ 5n + 70 \leq 6n \iff 70 \leq n \]

Thus, we can use $n_0 = 70$.

Symbol "\(\iff\)" means "if and only if" i.e. logical equivalence.
\[ 5n + 70 = 12 \]

\[ 75n \]

\[ n_0 = 1 \]

\[ 11n \]

\[ n_0 = 12 \]

\[ 6n \]

\[ n_0 = 70 \]
We would like to express formally how some function \( t(n) \) grows with \( n \), as \( n \) becomes large.

We would like to compare the function \( t(n) \) with \textit{simpler} functions, \( g(n) \), such as \( \log_2 n, n, n^2, \ldots, 2^n \), etc.
Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

$g(n)$ will be a simple function, but this is not required in the definition.

We say $t(n)$ is $O(g(n))$ if there exist two positive constants $n_0$ and $c$ such that, for all $n \geq n_0$,

$$t(n) \leq c \cdot g(n).$$
Intuition and visualization

- “f(n) is O(g(n))” iff there exists a point $n_0$ beyond which $f(n)$ is less than some fixed constant times $g(n)$

For all $n \geq n_0$

$$f(n) \leq c \cdot g(n) \quad (\text{for } c = 1)$$
Claim: $5n + 70$ is $O(n)$. 

\[ 5n + 70 \]

\[ n \]
Claim: \( 5n + 70 \) is \( O(n) \).

Proof 1:

\[
5n + 70 \leq \, ?
\]

We say \( t(n) \) is \( O(g(n)) \) if there exist two positive constants \( n_0 \) and \( c \) such that, for all \( n \geq n_0 \),

\[
t(n) \leq c \, g(n).
\]
Claim: $5n + 70$ is $O(n)$.

Proof 1:

$$5n + 70 \leq 5n + 70n, \text{ if } n \geq 1$$

We say $t(n)$ is $O(g(n))$ if there exist two positive constants $n_0$ and $c$ such that, for all $n \geq n_0$,

$$t(n) \leq c \cdot g(n).$$
Claim: \( 5n + 70 \) is \( O(n) \).

Proof 1:

\[
5n + 70 \leq 5n + 70n, \quad \text{if } n \geq 1
\]

\[
= 75n
\]

So take \( c = 75, \quad n_0 = 1 \).
Claim: \( 5n + 70 \) is \( O(n) \).

Proof 2:

\[
5n + 70 \leq 5n + 6n, \quad \text{if } n \geq 12
\]
Claim: \( 5n + 70 \) is \( O(n) \).

Proof 2:

\[
5n + 70 \leq 5n + 6n, \quad \text{if } n \geq 12
\]

\[
= 11n
\]

So take \( c = 11, \quad n_0 = 12 \).
Claim: $5n + 70$ is $O(n)$.

Proof 3:

$$5n + 70 \leq 5n + n, \quad n \geq 70$$
Claim: \( 5n + 70 \) is \( O(n) \).

Proof 3:

\[
5n + 70 \leq 5n + n, \quad n \geq 70
\]

\[
= 6n
\]

So take \( c = 6, \quad n_0 = 70 \).
\[ 5n + 70 = 12 \]

\[ 75n \]

\[ n_0 = 1 \]

\[ 5n + 70 \]

\[ 11n \]

\[ n_0 = 12 \]

\[ 6n \]

\[ 5n + 70 \]

\[ n_0 = 70 \]
Claim: \( 5n + 70 \) is \( O(n) \).

Incorrect Proof:

\[
\begin{align*}
5n + 70 & \leq cn \\
5n + 70n & \leq cn, \quad n \geq 1 \\
75n & \leq cn \\
\text{Thus,} \quad c > 75, \quad n_0 = 1
\end{align*}
\]

Q: Why is this incorrect?
Claim: \( 5n + 70 \) is \( O(n) \).

Incorrect Proof:

\[
\begin{align*}
5n + 70 & \leq cn \\
5n + 70n & \leq cn, \quad n \geq 1 \\
75n & \leq cn \\
\text{Thus, } \quad c & > 75, \quad n_0 = 1
\end{align*}
\]

Q: Why is this incorrect? A: Because we don’t know which line follows logically from which.
Claim: $8n^2 - 17n + 46$ is $O(n^2)$.

Proof (1):

$8n^2 - 17n + 46$
Claim: $8 n^2 - 17n + 46$ is $O(n^2)$.

Proof (1):

$$8 n^2 - 17n + 46 \leq 8 n^2 + 46 n^2, \quad n \geq 1$$
Claim: $8n^2 - 17n + 46$ is $O(n^2)$.

Proof (1):

\[ 8n^2 - 17n + 46 \leq 8n^2 + 46n^2, \quad n \geq 1 \]

\[ \leq 54n^2 \]
Claim: \( 8 n^2 - 17n + 46 \) is \( O(n^2) \).

Proof (1):

\[
8 n^2 - 17n + 46 \\
\leq 8 n^2 + 46 n^2, \quad n \geq 1 \\
\leq 54 n^2
\]

So take \( c = 54, \ n_0 = 1 \).
Claim: \( 8n^2 - 17n + 46 \) is \( O(n^2) \).

Proof (2):
\[
8n^2 - 17n + 46
\]
Claim: $8 n^2 - 17n + 46$ is $O(n^2)$.

Proof (2):

$$8 n^2 - 17n + 46 \leq 8 n^2 , \quad \quad n \geq 3$$

So take $c = 8, \quad n_0 = 3.$
What does $O(1)$ mean?

We say $t(n)$ is $O(1)$, if there exist two positive constants $n_0$ and $c$ such that, for all $n \geq n_0$,

\[ t(n) \leq c. \]

So it just means that $t(n)$ is bounded.
Never write $O(3n), O(5 \log_2 n)$, etc.

Instead, write $O(n), O(\log_2 n)$, etc.

Why? The point of big O notation is to avoid dealing with constant factors.

It is still technically correct to write the above. We just don’t do it.
Brute-Force Solution: $O(n!)$

Dynamic Programming Algorithms: $O(n^22^n)$

Selling on eBay: $O(1)$

Still working on your route?

Shut the hell up.