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,
  
  $T(2n) / T(n) \approx 5(2n)^2 / 5n^2$

  $\approx 4$

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. asymptotic behavior.

Unlike with limits, we want to say that $t(n)$ grows like certain simpler functions such as $\log_2 n, n, n^2, \ldots, 2^n$, etc.
Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$. 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 $n_0 \geq n$, \hspace{1em} 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 $n \geq n_0$, $5n + 70 \leq 6n$.

$$5n + 70 \leq 6n$$

$\Leftrightarrow$

$$70 \leq n$$

Symbol “$\Leftrightarrow$” 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.
\[5n + 70\]

\[n_0 = 1\]

\[n_0 = 12\]

\[n_0 = 70\]
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 simpler functions, $g(n)$, such as $\log_2 n$, $n$, $n^2$, ..., $2^n$, etc.
Formal Definition of Big O

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 $n \geq n_0$

$$f(n) \leq c \cdot g(n) \text{ (for } c = 1)$$
Claim: $5n + 70$ is $O(n)$.
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 \(n \geq n_0\),

\[t(n) \leq c \cdot g(n).\]
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 $c = 75$, $n_0 = 1$. 
Claim: \( 5n + 70 \) is \( O(n) \).

Proof 2:

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

Proof 2:

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

\[
= 11n
\]

So take \( c = 11, \ n_0 = 12 \).
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 \( c = 6, \quad n_0 = 70 \).
\[ 75n + 5n + 70 = 1211n + 5n + 70 = 70 \]
Claim: \( 5n + 70 \) is \( O(n) \).

Incorrect Proof:

\[
\begin{align*}
5n + 70 & \leq cn \\
5n + 70n & \leq cn, \quad n \geq 1 \\
75n & \leq cn \\
\end{align*}
\]

Thus, \( c > 75, \quad n_0 = 1 \)

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

Incorrect Proof:

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

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: \[ 8 n^2 - 17n + 46 \text{ is } O(n^2). \]

Proof (2):
\[
8 n^2 - 17n + 46 \\
\leq 8 n^2 , \quad n \geq 3
\]

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.
Considerations

- $n_0$ and $c$ are not \textit{uniquely} defined. For a given $c$ and $n_0$ that satisfy $O()$ we can increase one or both to again satisfy the definition. There is no “better” choice of constants.

- However, we generally want a “tight” upper bound, so smaller $O()$ relations give us more information. (This is not the same as smaller $c$ or $n_0$).

- e.g. any $f(n)$ that is $O(n)$ is also $O(n^2)$ and $O(2^n)$. But $O(n)$ is more informative.
Big O as a set

- When we show that a $t(n)$ is $O(g(n))$ you will sometimes see this written as $g(n) = O(g(n))$
- This is not strictly true given the standard definition of $=$ so instead we think of $O(g(n))$ as a set of functions bounded by $g(n)$.
- We can then say that $t(n)$ is a member of this set as such: $t(n) \in O(g(n))$
Example

Show that $n!$ is $\mathcal{O}((n + 2)!)$

$$n! \leq c(n + 2)!$$
$$n! \leq c(n + 2)(n + 1) \text{ divide by } n!$$
$$1 \leq c(n + 2)(n + 1)$$

We choose $n_0 = 1$ and $c = 1$
Show that \((n + 2)!\) is \(\mathcal{O}(n!\))
If this is true, I can write;

\[
(n + 2)! \leq n! \quad \text{for all } n \geq n_0
\]
\[
(n + 2)(n + 1)n! \leq cn!
\]
\[
(n + 2)(n + 1) \leq c
\] (2)

However, this is clearly not the case for all \(n \geq n_0\) since \(c\) is constant (and \(c < \infty\)) and so it cannot be larger than an infinite number of increasing \(n\)
Complexity Classes

Comparing Big O Functions

Number of Operations

- $O(2^n)$
- $O(n^2)$
- $O(n \log n)$
- $O(n)$
- $O(\log n)$
- $O(1)$

$n$
(amount of data)

(C) 2010 Thomas J Cortina, Carnegie Mellon University
Brute-force solution: $O(n!)$

Dynamic programming algorithms: $O(n^2 2^n)$

Selling on eBay: $O(1)$

Still working on your route?

Shut the hell up.