Big O makes a difference

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O(1)$</th>
<th>$O(\log n)$</th>
<th>$O(n)$</th>
<th>$O(n \log n)$</th>
<th>$O(n^2)$</th>
<th>$O(n^3)$</th>
<th>$O(2^n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>4,096</td>
<td>65,536</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1,024</td>
<td>32,768</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>6</td>
<td>64</td>
<td>384</td>
<td>4,069</td>
<td>262,144</td>
<td>$1.84 \times 10^{19}$</td>
</tr>
</tbody>
</table>

~$10^{11}$ years!
Recall Formal Definition of Big O

Let \( t(n) \) and \( g(n) \) be two functions, where \( n \geq 0 \).

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

\[
t(n) \leq c \cdot g(n).
\]
“constant factor” rule

Suppose \( f(n) \) is \( O(g(n)) \) and \( a \) is a positive constant.

Then \( a \, f(n) \) is also \( O(g(n)) \).

(This rule is obvious if and only if you understand the definition of big O.)
Proof of “constant factor” rule

By definition, if \( f(n) \) is \( O(g(n)) \) then there exist two positive constants \( n_0 \) and \( c \) such that, for all \( n \geq n_0 \),

\[
f(n) \leq c \cdot g(n).
\]

Thus,

?
Proof of “constant factor” rule

By definition, if \( f(n) \) is \( O(g(n)) \) then there exist two positive constants \( n_0 \) and \( c \) such that, for all \( n \geq n_0 \),

\[
f(n) \leq c \cdot g(n).
\]

Thus,

\[
af(n) \leq ac \cdot g(n).
\]

This constant satisfies the definition that \( af(n) \) is \( O(g(n)) \).
Sum Rule

Suppose $f_1(n)$ is $O(g(n))$ and $f_2(n)$ is $O(g(n))$.

Then $f_1(n) + f_2(n)$ is $O(g(n))$.

Proof:

Let $n_1$, $c_1$ and $n_2$, $c_2$ be constants such that $f_1(n) \leq c_1 g(n)$ for all $n \geq n_1$, $f_2(n) \leq c_2 g(n)$ for all $n \geq n_2$. So, $f_1(n) + f_2(n) \leq (c_1 + c_2) g(n)$ for all $n \geq \max(n_1, n_2)$. 
**Sum Rule**

Suppose $f_1 (n)$ is $O(g(n))$ and $f_2 (n)$ is $O(g(n))$.

Then $f_1 (n) + f_2 (n)$ is $O(g(n))$.

Proof: Let $n_1, c_1$ and $n_2, c_2$ be constants such that

$$f_1 (n) \leq c_1 g(n) \text{ for all } n \geq n_1$$

$$f_2 (n) \leq c_2 g(n) \text{ for all } n \geq n_2.$$
Sum Rule

Suppose $f_1(n)$ is $O(g(n))$ and $f_2(n)$ is $O(g(n))$.

Then $f_1(n) + f_2(n)$ is $O(g(n))$.

Proof: Let $n_1, c_1$ and $n_2, c_2$ be constants such that

$$f_1(n) \leq c_1 g(n) \text{ for all } n \geq n_1$$

$$f_2(n) \leq c_2 g(n) \text{ for all } n \geq n_2.$$

So, $f_1(n) + f_2(n) \leq (c_1 + c_2) g(n)$ for all $n \geq \max(n_1, n_2)$

These constants satisfy the big O definition
Sum Rule  (more general)

Suppose \( f_1(n) \) is \( O(g_1(n)) \) and \( f_2(n) \) is \( O(g_2(n)) \).

Then \( f_1(n) + f_2(n) \) is \( O(g_1(n) + g_2(n)) \).

Proof:  (Exercise – see the following)
Product Rule

Suppose \( f_1(n) \) is \( \theta(g_1(n)) \) and \( f_2(n) \) is \( \theta(g_2(n)) \).

Then \( f_1(n) \cdot f_2(n) \) is \( \theta(g_1(n) \cdot g_2(n)) \).

Proof:
Product Rule

Suppose \( f_1(n) \) is \( O(g_1(n)) \) and \( f_2(n) \) is \( O(g_2(n)) \).

Then \( f_1(n) \cdot f_2(n) \) is \( O(g_1(n) \cdot g_2(n)) \).

Proof: Let \( n_1, c_1 \) and \( n_2, c_2 \) be constants such that

\[
\begin{align*}
f_1(n) &\leq c_1 g_1(n) \text{ for all } n \geq n_1 \\
f_2(n) &\leq c_2 g_2(n) \text{ for all } n \geq n_2.
\end{align*}
\]
Product Rule

Suppose $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$.

Then $f_1(n) \cdot f_2(n)$ is $O(g_1(n) \cdot g_2(n))$.

Proof: Let $n_1, c_1$ and $n_2, c_2$ be constants such that

$$f_1(n) \leq c_1 g_1(n) \text{ for all } n \geq n_1$$

$$f_2(n) \leq c_2 g_2(n) \text{ for all } n \geq n_2.$$  

So, $f_1(n) \cdot f_2(n) \leq c_1 c_2 g_1(n) g_2(n)$ for all $n \geq \max(n_1, n_2)$.

These constants satisfy the big O definition.
Transitivity Rule

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then ?
Transitivity Rule

If \( f(n) \) is \( O(g(n)) \) and \( g(n) \) is \( O(h(n)) \)

then \( f(n) \) is \( O(h(n)) \).

Proof:
Transitivity Rule

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $f(n)$ is $O(h(n))$.

Proof: Let $n_1, c_1$ and $n_2, c_2$ be constants such that

$$f(n) \leq c_1 g(n) \text{ for all } n \geq n_1$$

$$g(n) \leq c_2 h(n) \text{ for all } n \geq n_2.$$ 

So, ...
Transitivity Rule

If \( f(n) \) is \( O(g(n)) \) and \( g(n) \) is \( O(h(n)) \), then \( f(n) \) is \( O(h(n)) \).

Proof: Let \( n_1, c_1 \) and \( n_2, c_2 \) be constants such that

\[
f(n) \leq c_1 g(n) \quad \text{for all } n \geq n_1
\]

\[
g(n) \leq c_2 h(n) \quad \text{for all } n \geq n_2.
\]

So, \( f(n) \leq c_1 c_2 h(n) \) for all \( n \geq \max(n_1, n_2) \).

These constants satisfy the big O definition.
Claim: Each of the following holds for $n$ sufficiently large:

$1 < \log_2 n < n < n \log_2 n < n^2 < n^3 < \ldots < 2^n < n!$

$n \geq 1 \quad n \geq 1 \quad n \geq 4$

$n^3 < 2^n$ for $n \geq 10$

Thus, we can write big O relationships between them.
Using rules to simplify long expressions

\[ f(n) = 3 + 17 \log_2 n + 4n + \frac{n(n + 1)}{6} \in \mathcal{O}(n^2) \]

e.g. Let’s simplify the first two terms

Let

\[ f_1(n) = 3 \]

\[ f_2(n) = 17 \log_2 n \]
Continued…

\[ f_1(n) \in \mathcal{O}(1) \]
\[ 1 \in \mathcal{O}(\log_2 n) \]
\[ f_1(n) \in \mathcal{O}(\log_2 n) \]
\[ f_2(n) \in \mathcal{O}(\log_2 n) \]
\[ f_1(n) + f_2(n) \in \mathcal{O}(\log_2 n) \]

by transitivity

trivially, pick \( c = 17 \)

by sum rule
Sets of $O(\cdot)$ functions

If $t(n)$ is $O( g(n) )$, we often write

$$t(n) \in O( g(n) ).$$

That is, $t(n)$ is a member of the set of functions that are $O( g(n) )$. 
Each of the following holds for \( n \) sufficiently large:

\[
1 < \log_2 n < n < n \log_2 n < n^2 < n^3 < \ldots < 2^n < n!
\]

Thus we have the following strict subset relationships:

\[
O(1) \subset O(\log_2 n) \subset O(n) \subset O(n \log_2 n) \subset O(n^2) \ldots \subset O(n^3) \subset \ldots \subset O(2^n) \subset O(n!)
\]
Asymptotic lower bounds

Sometimes we want to say that algorithms take at least a certain time to run as a function of the input size $n$.

e.g. One can show (COMP 251) that it is impossible to sort $n$ items "faster than $n \log_2 n"."
Asymptotic lower bound ("big Omega")

\[ t(n) \]
\[ g(n) \]
\[ n_0 \]
“small omega” $\omega$

“big omega” $\Omega$
Preliminary Definition (lower bound)

\( t(n) \) is \textit{asymptotically bounded below by} \( g(n) \) if there exists an \( n_0 \) such that, for all \( n \geq n_0 \),

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

Note: As with big O, the constant \( n_0 \) is not unique. If the definition works for some \( n_0 \), then it will work for larger \( n_0 \) too.
Example: \( t(n) = \frac{n(n-1)}{2} \) is asymptotically bounded below by \( g(n) = \frac{n^2}{4} \).

Proof:

\[ \frac{n(n-1)}{2} \geq ? \]
Example: \( t(n) = \frac{n(n-1)}{2} \) is asymptotically bounded below by \( g(n) = \frac{n^2}{4} \).

Proof:

\[
\frac{n(n-1)}{2} \geq \frac{n^2}{4}
\]

For each \( n \), this is either true or false.
Let’s rearrange this and find values of \( n \) for which this is true.
Example: \( t(n) = \frac{n(n-1)}{2} \) is asymptotically bounded below by \( g(n) = \frac{n^2}{4} \).

Proof:

\[
\frac{n(n-1)}{2} \geq \frac{n^2}{4}
\]

\[\iff 2n(n - 1) \geq n^2\]

"\(\iff\) " Means "if and only if"."
Example: \[ t(n) = \frac{n(n-1)}{2} \] is asymptotically bounded below by \( g(n) = \frac{n^2}{4} \).

Proof:

\[ \frac{n(n-1)}{2} \geq \frac{n^2}{4} \]

\( \iff \quad 2n(n - 1) \geq n^2 \)

\( \iff \quad n^2 \geq 2n \)
Example: \( t(n) = \frac{n(n-1)}{2} \) is asymptotically bounded below by \( g(n) = \frac{n^2}{4} \).

Proof:

\[
\frac{n(n-1)}{2} \geq \frac{n^2}{4}
\]

\[
\iff 2n(n - 1) \geq n^2
\]

\[
\iff n^2 \geq 2n
\]

\[
\iff n \geq 2 \quad \text{So take } n_0 = 2.
\]
Formal Definition of Big Omega ($\Omega$)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

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

$$t(n) \geq c \cdot g(n).$$
Claim: \( \frac{n(n-1)}{2} \) is \( \Omega(n^2) \).

Proof (1): Use \( c = \frac{1}{4} \) and derivation we just did.

\[
\frac{n(n-1)}{2} \geq \frac{n^2}{4}
\]

\[\iff\]

\[n \geq 2 \quad \text{So take } n_0 = 2, \quad c = \frac{1}{4}.\]
Claim: \( \frac{n(n-1)}{2} \) is \( \Omega(n^2) \).

Proof (2): Try \( c = \frac{1}{3} \)  

\[
\frac{n(n-1)}{2} \geq \frac{n^2}{3}
\]

\( \iff \) you can fill this in

\( \iff n \geq 3 \quad \text{So take } n_0 = 3, \quad c = \frac{1}{3}. \)
Exercise

Show that the constant, sum, product, transitivity rules all hold for big Omega also.
Sets of $\Omega()$ functions

If $t(n)$ is $\Omega(g(n))$, we often write

$$t(n) \in \Omega(g(n)).$$

That is, $t(n)$ is a member of the set of functions that are $\Omega(g(n))$. 
Each of the following holds for $n$ sufficiently large:

$$1 < \log_2 n < n < n \log_2 n < n^2 < n^3 < \ldots < 2^n < n!$$

We have the following strict subset relationships:

$$\Omega(1) \subset \Omega(\log_2 n) \subset \Omega(n) \subset \Omega(n \log_2 n) \subset \Omega(n^2) \ldots \subset \Omega(n^3) \subset \ldots \Omega(2^n) \subset \Omega(n!)$$
Definition of Big Theta (Θ)

Let \( t(n) \) and \( g(n) \) be two functions of \( n \geq 0 \).

We say \( t(n) \) is \( \Theta(g(n)) \), if there exist three positive constants \( n_0, c_1, c_2 \) such that for all \( n \geq n_0 \),

\[
c_1 \, g(n) \leq t(n) \leq c_2 \, g(n)
\]
Example

\[ t(n) \text{ is } \Theta(g(n)). \]
Definition of Big Theta ($\Theta$)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

We say $t(n)$ is $\Theta(g(n))$, if there exist three positive constants $n_0$, $c_1$, $c_2$ such that for all $n \geq n_0$,

$$c_1 \cdot g(n) \leq t(n) \leq c_2 \cdot g(n)$$

$t(n)$ is $O(g(n))$
Definition of Big Theta ($\Theta$)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

We say $t(n)$ is $\Theta(g(n))$, if there exist three positive constants $n_0$, $c_1$, $c_2$ such that for all $n \geq n_0$,

$$c_1 \ g(n) \leq t(n) \leq c_2 \ g(n)$$

$t(n)$ is $\Omega(g(n))$
Example

Let \( t(n) = 4 + 17 \log_2 n + 3n + 9n \log_2 n + \frac{n(n-1)}{2} \).

Claim: \( t(n) \) is \( \Theta(n^2) \).

Proof:
Example

Let \( t(n) = 4 + 17 \log_2 n + 3n + 9n \log_2 n + \frac{n(n-1)}{2} \).

Claim: \( t(n) \) is \( \Theta(n^2) \).

Proof:

\[
\frac{n^2}{4} \leq t(n) \leq (4 + 17 + 3 + 9 + \frac{1}{2})n^2
\]
For every $t(n)$, does there exist a “simple” $g(n)$ such that $t(n)$ is $\Theta()$?
For every $t(n)$, does there exist a “simple” $g(n)$ such that $t(n)$ is $\Theta()$?

No, as this contrived example shows:

Let $t(n) = \begin{cases} 
  n, & \text{n is odd} \\
  n^2, & \text{n is even}. 
\end{cases}$

$t(n)$ is $O(n^2)$, but $t(n)$ is not $O(n)$.

$t(n)$ is $\Omega(n)$, but $t(n)$ is not $\Omega(n^2)$.
Sets of $\Theta()$ functions

If $t(n)$ is $\Theta(g(n))$, we often write $t(n) \in \Theta(g(n))$.

That is, $t(n)$ is a member of the set of functions that are $\Theta(g(n))$. 
The time it takes for an algorithm to run depends on:

- constant factors (often implementation dependent)
- the size $n$ of the input
- ?
The time it takes for an algorithm to run depends on:

- constant factors (often implementation dependent)
- the size $n$ of the input
- the values of the input, including arguments if applicable

What are the best and worst cases?
For any algorithm,

let $t_{\text{best}}(n)$ be the runtime on the best case input(s).

let $t_{\text{worst}}(n)$ be the runtime for the worse case input(s).
e.g. Consider finding an element in a doubly linked list:

```java
linkedlist.get(e)
```

In the best case, ...

In the worst case, ...
A Linked List
Consider finding an element in a doubly linked list:

\[
\text{linkedlist.get}(e)
\]

In the best case, the element is at the front of the list and so it can be found in constant time. So,

\[
t_{\text{best}}(n) \text{ is } \Theta(1).
\]

In the worst case, ...
Consider finding an element in a doubly linked list:

\[ \text{linkedlist.get}(e) \]

In the best case, the element is at the front of the list and so it can be found in constant time. So,

\[ t_{\text{best}}(n) \text{ is } \Theta(1). \]

In the worst case, the element is at the opposite end of the list from where we start. So,

\[ t_{\text{worst}}(n) \text{ is } \Theta(n). \]
Binary search

Worst case (reach subarray of size 1)
Binary search

Worst case (reach subarray of size 1)

$\Theta(\log n)$

Best case (target is in middle)
Binary search

Worst case (reach subarray of size 1)

$\Theta(\log n)$

Best case (target is in middle)

$\Theta(1)$

$A = [20, 30, 40, 50, 80, 90, 100]$

binarySearch(40)
Algorithm SelectionSort(A,n)

Input: an array A of n elements

Output: the array is sorted

\[ i \leftarrow 0 \]

while \((i<n)\) do {
    \[ \text{minindex} \leftarrow \text{findMin}(A,i,n-1) \]
    \[ t \leftarrow A[\text{minindex}] \]
    \[ A[\text{minindex}] \leftarrow A[i] \]
    \[ A[i] \leftarrow t \]
    \[ i \leftarrow i + 1 \]
}

Primitive operations (worst case):

1
2
3
3 + \( T_{\text{findMin}(n-1-i+1)} = 3 + (10(n-i) - 2) \)
2
3
2
2
2 (last check of loop condition)

Best?
Worst?
Heads up!

When people want to express that an algorithm has different asymptotic behavior in best versus worst cases, they sometimes say that the algorithm is \( \Omega( g_{\text{best}}(n) ) \) and \( O( g_{\text{worst}}(n) ) \).

e.g. “Quicksort is \( \Omega( n \log_2 n ) \) and \( O( n^2 ) \).”

The statement is technically correct. However, there is a natural tendency to equate \( \Omega \) with best case and \( O \) with worst case. Technically speaking, this is incorrect.
Rather, for any algorithm,

\( t_{\text{best}}(n) \) is the runtime on the best case input(s).

\( t_{\text{worst}}(n) \) is the runtime for the worse case input(s).

\( t_{\text{best}}(n) \) and \( t_{\text{worst}}(n) \) each \textit{typically} belong to some \( \Theta() \) set, possibly the same.

\[ \Theta(1) \quad \Theta(\log_2 n) \quad \Theta(n) \quad \ldots \quad \Theta(2^n) \quad \Theta(n!) \]
If we have time...

Q: Can we use limits to prove asymptotic bounds?

A: Yes, if we apply certain rules.
Recall: Definition of Big O

Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

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

$$t(n) \leq c \cdot g(n)$$

or equivalently

$$\frac{t(n)}{g(n)} \leq c.$$
A sequence \( t(n) \) is \( O(g(n)) \), if there exists positive constants \( n_0 \) and \( c \) such that, for all \( n \geq n_0 \),

\[
\frac{t(n)}{g(n)} \leq c.
\]

A sequence \( t(n) \) has a limit \( t_\infty \) if, for any \( \varepsilon > 0 \), there exists an \( n_0 \) such that, for any \( n \geq n_0 \),

\[
|t(n) - t_\infty| < \varepsilon.
\]
Rule 1

If

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0$$

then

?
If
\[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \]
then, by definition,

for any \( \varepsilon > 0 \), there exists an \( n_0 \)

such that for any \( n \geq n_0 \),

\[ \left| \frac{t(n)}{g(n)} - 0 \right| < \varepsilon. \]
If
\[
\lim_{{n \to \infty}} \frac{t(n)}{g(n)} = 0
\]
then, by definition,

in particular, there exists \( \varepsilon > 0 \) and

for any \( \varepsilon > 0 \), there exists an \( n_0 \)

such that for any \( n \geq n_0 \),

\[
\left| \frac{t(n)}{g(n)} - 0 \right| < \varepsilon
\]

and so \( \frac{t(n)}{g(n)} < \varepsilon \) and so \( t(n) \) is \( O(g(n)) \).
What about the opposite direction?

If \( t(n) \) is \( O(g(n)) \).

Then: \( \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \).
The opposite direction does not always hold.

e.g. Take $t(n) = g(n)$. Then $t(n)$ is $O(g(n))$.  

but \[ \frac{t(n)}{g(n)} = 1 \] for all $n$, and so the limit isn’t 0.
Rule 1 (more general)

If

\[
\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0
\]

then:

\( t(n) \) is \( O( g(n) ) \).

\( t(n) \) is not \( \Omega( g(n) ) \)  \( Why \ not? \)

Thus,

\( t(n) \) is not \( \Theta( g(n) ) \).
Recall Definition of Big Omega

Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

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

$$t(n) \geq c \cdot g(n)$$

or equivalently

$$\frac{t(n)}{g(n)} \geq c.$$
\[ \lim_{{n \to \infty}} \frac{t(n)}{g(n)} = 0 \]

\[ t(n) \text{ is } \Omega(g(n)), \text{ if there exist two constants } n_0 \text{ and } \\
c > 0 \text{ such that, for all } n \geq n_0, \quad \frac{t(n)}{g(n)} \geq c. \]

It is impossible that both of the above are true!
If \[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \]

then:

\[ t(n) \text{ is } O( g(n) ). \]

\[ t(n) \text{ is not } \Omega( g(n) ) \quad \text{Proved!} \]

Thus,

\[ t(n) \text{ is not } \Theta( g(n) ). \]