Most of us would rather not deal with "minor" details such as

- whether it takes 2, 3, 7, or 53 instructions/operations
to compare & swap two numbers & re-iterate. e.g. $\frac{1}{2}n^2 + \frac{1}{2}n$

- the size of non-leading terms e.g. $\frac{1}{2}n^2 + \frac{1}{2}n$

For theory, the fun (and easier) part is to deal with large $n$
so what we care about is $\frac{1}{2}n^2 + \frac{1}{2}n \approx n^2$

This leads to $\Theta$-notation
$F(n) = 53n^2 + 107n - 6 = \Theta(n^2)$

just look at the dominating term.

Whatever $b$ and $c$ are, eventually $n$ will grow large enough that $bn + c \ll an^2$

The same goes for $a$: it is a constant and thus overshadowed by any function of $n$ (that grows $\rightarrow \infty$)

\[ f(n) = n + 100 \]
\[ g(n) = \frac{1}{8}n^2 \]
$\Theta$-notation can also be called "big-O" notation.

Formally, $f(n) = O(g(n))$ \textit{or} $f(n) \in O(g(n))$

\[ \text{there is a constant } c \text{ such that for all } n > n_0 \rightarrow 0 \leq f(n) \leq c \cdot g(n) \]

\[ \text{upper bound.} \]

$g(n)$ is an asymptotic upper bound for $f(n)$.

\[ \text{Usage: given a complicated } f(n), \text{ find a simple } g(n) \text{ s.t. } f(n) = O(g(n)) \]
\[ f(n) = O(g(n)) \quad \rightarrow \quad f(n) \leq c \cdot g(n). \]

\[ \Omega \text{ gives a lower bound: } \quad f(n) = \Omega(g(n)) \quad \rightarrow \quad f(n) \geq c \cdot g(n) \]

Again, exists some \( c \) for all \( n > n_0 \) ...

\[ \Theta \text{ is the combination of } O \text{ & } \Omega \]

\[ f(n) = \Theta(g(n)) \quad \text{if} \quad f(n) = O(g(n)) \text{ AND } f(n) = \Omega(g(n)) \]

i.e. \[ 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \]

for all \( n > n_0 \) ...
\[ f(n) = O(g(n)) \]
\( f(n) = \Omega(h(n)) \)
If you want a better upper bound, it will only be valid for larger $n$. 

$c \cdot g(n)$

$c_2 \cdot g(n)$

$f(n)$

$c_2 < c$

$f(n) = O(g(n))$
$f(n) = \Theta(g(n))$
Prove $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find $c_1, c_2, n_0$ s.t. $c_1 n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 n^2$ for all $n > n_0$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

For $n \gg 1 : c_2 \gg \frac{1}{2}$ works.

For $n \gg 17 : c_1 \leq \frac{1}{14}$ (or, for $n \gg 30 : c_1 \leq 0.4$)

(as $n_0 \to \infty$, $c_1 \to \frac{1}{2}$)

Prove $6n^3 = \Theta(n^2)$

Find $c_1, c_2, n_0$ s.t. $c_1 n^2 \leq 6n^3 \leq c_2 n^2$

Trivially true for $n \gg 1$ & $c_1 \leq 6$

R.h.s. : $6n \leq c_2 \Rightarrow n \leq \frac{c_2}{6}$ : not true for any constant $c_2$
Is \( 50 \cdot n = O(n^2) \) ?

\[
\begin{align*}
50 \cdot n &\leq c \cdot n^2 \quad \text{if } n > 1, c = 50 \\
50 &\leq c \cdot n \quad \text{if } n > 50, c = 1
\end{align*}
\]

Yes, but also \( 50 \cdot n = O(n) \) \( 50 \cdot n \leq c \cdot n \) if \( n \) is sufficiently large

\( 50 \cdot n = O(n^2) \) is not wrong; it's just not very accurate.
Some basic functions: \( \log n, \sqrt{n}, n, n \log n, n^2, n^3, 2^n, n! \)

\[ \log n = O(\sqrt{n}) \quad n^2 = \Omega(n \log n) \]

Try \( n^{1-\varepsilon}, 3^{n-\varepsilon}, \log(n^2), \log^2 n, 2^{\log_2 n}, 4^{\log_2 n}, n, n^2, n^3 \)

(for \( \varepsilon > 0 \))

compare \( 2^n \) v. \( 3^{n-\varepsilon} \)

\( \log n \) v. \( \log(n^2) \)

\( \log(n^2) \) v. \( \log^2 n \)

\( 2^{\log_2 n} \) v. \( n^k \) (k=?)

etc.
Recall that we determined that Insertion sort costs $\sum_{j=1}^{n} j$

This is $\Theta(n^2)$

Does it always cost this? No. This is the worst case.

We could also discuss the "best case": that is less interesting.

Any algo can be expanded to handle some input $I$ in $O(|I|)$ time.

Anyway, Insertion sort would take $O(n)$ time if the input was already sorted.

Insertion sort takes $\Omega(n)$ time but the $O(n^2)$-time bound is "tight". // The worst-case time complexity of insertion sort is $\Theta(n^2)$
Sometimes big-\(O\) is used to describe lower order terms.

\[ 2n^2 + 3n + 1 = 2n^2 + o(n). \]
More asymptotic notation although less common.

Recall that

$\Theta : = \\begin{array}{c}
O : = \\
\Omega : > 
\end{array}$

little-$o$ $\rightarrow <$

$f(n) = o(g(n))$ if for any constant $c$
there exists large enough $n_0$
s.t. $0 \leq f(n) < c \cdot g(n)$ for all $n \geq n_0$

little-$\omega$ $\rightarrow >$

In other words, once again in the world of very large $n$,
$f(n)$ is not asymptotically equivalent to $g(n)$.
No matter what constant we multiply $f(n)$ with
it won't exceed $g(n)$
Example of little-o

\[ 5n = o(n^2) \]  \quad \text{For any constant } c \quad 5 \cdot c \cdot n < n^2 \text{ when } n > 5 \cdot c

In some sense \( f(n) = o(g(n)) \) means \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)

\( \omega \) is similar \( \frac{1}{10}n^3 = \omega(n^2) \)  \quad \text{For any } c, \text{ there is some } n_0 \text{ s.t. } \frac{1}{10}n^3 > c \cdot n^2 \text{ for all } n > n_0

\[ \text{given } c, \text{ choose } n \gg 10 \cdot c \]

If \( f(n) = \omega(g(n)) \), \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \)

Notice \( f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n)) \)
see CLRS 51 for intuitive transitivity rules.

So, what do we accomplish with $\Theta$-notation?

- We can understand behavior of algorithms for large input.
  (Same for complexity of problems)
- We can analyze algorithms in a hardware/implementation-independent way.