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 $\frac{1}{2}n^2 + \frac{1}{2}n \sim n^2$

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

Jean look at the dominating term.

\( an^2 + bn + c \)

Whatever \( b \) an \( c \) 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 \( \to \infty \))

\( f(n) = n + 100 \)

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

Formally, \( f(n) = O(g(n)) \) \(-or-\) \( f(n) \in O(g(n)) \)

if for all \( n > n_0 \) \( \exists \) s.t. \( 0 \leq f(n) \leq c \cdot g(n) \)

there is a constant \( c \)

\( g(n) \) is always a simplification of \( f(n) \),

is an asymptotic upper bound for \( f(n) \).

i.e. to within a const. factor,

for large values.
\( f(n) = O(g(n)) \quad \Rightarrow \quad f(n) \leq c \cdot g(n) \).

\( \Omega \) gives a lower bound:

\[ f(n) = \Omega(g(n)) \quad \Rightarrow \quad f(n) \geq c \cdot g(n) \]

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

\( \Theta \) is the combination of \( O \) & \( \Omega \)

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

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 $n$.

$c_3 < c_2$

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

Find \( c_1, c_2, n_0 \) s.t. 
\[
\frac{1}{2}n^2 - 3n \leq c_1 n^2 \leq c_2 n^2 
\]

for all \( n > n_0 \)

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

For \( n > 1 \): \( c_2 \geq \frac{1}{2} \) works.

For \( n > 7 \): \( c_1 \leq \frac{1}{14} \)

(or, for \( n > 30 \): \( c_1 \leq 0.4 \))

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

---

Prove \( 6n^3 \neq \Theta(n^2) \)

Find \( c_1, c_2, n_0 \) s.t. 
\[
\frac{1}{2}n^2 \leq 6n^3 \leq c_2 n^2
\]

trivially true for \( n > 1 \) & \( c_1 \leq 6 \)

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

\[
\begin{align*}
50 \cdot n & \leq c \cdot n^2 \\
50 & \leq c \cdot n \\
\text{(n=10, c=50)} \\
\text{So } & \leq c \\
\text{for all n.}
\end{align*}
\]

Yes, but also \( 50 \cdot n = O(n) \)

\[
50 \cdot n = O(n^2) \text{ is not wrong; it's just not very accurate.}
\]
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 $n^{1-\varepsilon}, 3^{n-\varepsilon}, \log(n^2), \log^2 n, 2^{\log_4 n}, 4^{\log_2 n}, n, n^2, n^3$

(for $\varepsilon > 0$)

compare $2^n \overset{?}{=} 3^{n-\varepsilon}$

$\log n \overset{?}{=} \log(n^2)$

$\log(n^2) \overset{?}{=} \log^2 n$

$2^{\log_4 n} \overset{?}{=} n^k \quad (k=?)$

et cetera
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".
"This algorithm ... ... takes $O(n^2)$ time"
... runs in $O(n^2)$ time"
... has a time complexity of $O(n^2)$"
... runs in quadratic time,
and this is known to be tight".

"This problem ... ... has a quadratic-time solution
and there is an $\Omega(n \log n)$ lower bound
... is known to require $\Omega(n^2)$ time."

Problems have upper bounds and lower bounds
Algorithms: • have equal or weaker bounds
• can have upper bounds that are "tight"
  but that implies nothing for the lower bound of the problem
Sometimes big-O is used to describe lower order terms.

\[ 2n^2 + 3n + 1 = 2n^2 + O(n) \,.
\]

We can continue to say this is \( \Theta(n^2) \).
More asymptotic notation although less common.

Recall that

\[ \Theta : = \]
\[ O : \leq \]
\[ \Omega : \geq \]

\text{little-}o \to < \quad \text{e.g. } f(n) = o(g(n)) \quad \text{if for any constant c there exists large enough n}_0 \quad \text{s.t. } 0 \leq f(n) < c \cdot g(n) \quad \text{for all } n \geq n_0

\text{little-}o \to >

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

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

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

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

Indeed, given \( c \), choose \( n > 10c \)

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

Notice \( f(n) = o(g(n)) \iff g(n) = \omega(f(n)) \)
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.

*Compare insertion sort on the fastest machine, coded by the best vs. some \( O(n\log n) \)-time algorithm on a 20-yr-old computer, coded by me.*

For large \( n \), I will win.