- a way to measure speed of an algorithm (independent of implementation)
- different from program speed.
- vaguely it is the number of "steps" taken to complete an algorithm.
- quantified using different types of "asymptotic notations".

1. **Big-O notation**:
   \[ f(n) = O(g(n)) \iff \exists C, n_0 \text{ (both positive)} \text{ s.t. } f(n) \leq C g(n) \text{ for all } n \geq n_0. \]

Alternatively:
\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = \epsilon \text{ where } \epsilon \text{ is some finite constant} \]

**Graphical representation**:

\[ C = 1, f(n) \text{ is } O(g(n)) \]
\[ f(n) = 3n + 400, \quad g(n) = n^2 + 10n + 5 \]

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{3n + 400}{n^2 + 10n + 5} = \lim_{n \to \infty} \frac{3 + 400/n}{n + 10 + 5/n} = 0.
\]

\[ f(n) \text{ is } O(g(n)) \quad \text{(in fact } f(n) \text{ is also } o(g(n)) \text{ — little o, which we'll see later)} \]

**Method 2**

Also we can clearly say that

\[ 3n + 400 \leq 3(n^2 + 10n + 5) \quad \forall n \geq n_0. \]

\[ n_0 = ? \]

<table>
<thead>
<tr>
<th>(3n + 400)</th>
<th>(3n^2 + 10n + 5)</th>
<th>(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>403</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>406</td>
<td>37</td>
<td>2</td>
</tr>
<tr>
<td>409</td>
<td>62</td>
<td>3</td>
</tr>
<tr>
<td>4012</td>
<td>93</td>
<td>4</td>
</tr>
<tr>
<td>430</td>
<td>335</td>
<td>5</td>
</tr>
<tr>
<td>445</td>
<td>&gt;600</td>
<td></td>
</tr>
<tr>
<td>460</td>
<td>&gt;120</td>
<td></td>
</tr>
</tbody>
</table>

Another example: \( f(n) = 3n + 20 \)

and \( g(n) = n + 1 \).

Here \( f(n) \) is \( O(g(n)) \). In fact here \( g(n) \) is also \( O(f(n)) \).
We can say \( f(n) \) is \( O(g(n)) \) because
\[
\lim_{n \to \infty} \frac{3n+20}{n+1} = 3 = \text{constant}.
\]
Likewise \( \lim_{n \to \infty} \frac{n+1}{3n+20} = \frac{1}{3} = \text{constant}.

Method 2
Also \( f(n) \leq 20g(n) \) i.e. \( 3n+1 \leq 20(n+1) \)
\[\forall n \geq 1, \text{ i.e. } C = 30, n_0 = 1. \text{ So } f(n) \text{ is } O(g(n)).\]

Similarly, \( g(n) \leq 1f(n) \) \( \forall n \geq 1 \) i.e. \( C = 1, n_0 = 1 \.
So \( g(n) \) is \( O(f(n)) \).

Note: If I say \( f(n) \) is \( O(g(n)) \), it means that \( g(n) \) grows faster than or as fast as \( f(n) \) in the limit when \( n \) becomes larger and larger.

Tip: If \( f(n) \) is a polynomial of the form \( f(n) = \sum_{i=0}^{k} a_i n^i \), then \( f(n) = O(n^k) \), i.e. only the fastest growing term matters in a compound mathematical expression.

On the following page, we make a list of functions in increasing order of their growth rates:
<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>Linear</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Quadratic</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>Cubic</td>
</tr>
<tr>
<td>$O(n^x)$</td>
<td>polynomial</td>
</tr>
<tr>
<td>($x$ is a constant $&gt; 1$)</td>
<td></td>
</tr>
<tr>
<td>$\alpha^n$ ($\alpha &gt; 1$)</td>
<td>exponential</td>
</tr>
<tr>
<td>$n^n$</td>
<td>exponential with non-constant base</td>
</tr>
</tbody>
</table>

There are functions that grow slower than linear. They are called sublinear functions.

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant</td>
</tr>
<tr>
<td>$O(\log \log n)$</td>
<td></td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>logarithmic complexity</td>
</tr>
<tr>
<td>$O(n^x)$, $\alpha &lt; 1$</td>
<td>sublinear</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>linear</td>
</tr>
</tbody>
</table>
In all asymptotic analyses, constant multiples may be ignored. So the complexity of \( f(n) = 3n^2 + 10n + 50 \) is the same as \( g(n) = 0.0001n^2 + 0.001n \) (both are quadratic). But you should never ignore constants in exponential terms.

E.g.: \( 4^n \) does not have the same complexity as \( 2^n \). In fact, \( 2^n \) is \( O(4^n) \).

Note: When I say \( f(n) \) is \( O(g(n)) \), it means \( f(n) \) grows faster than (or only as fast as) \( g(n) \), \( g(n) \) may grow faster than \( f(n) \) and so it may be a very loose relationship.

For e.g.: \( f(n) = 3n + 10 \) is \( O(n) \). It is also \( O(n^2) \) or \( O(n^3) \) or \( O(n^n) \). Saying that \( f(n) \) is \( O(n^n) \) when \( f(n) = 3n + 10 \) is like saying that all M.Sc. IT students at DAIICT are less than 100 years of age (:-)) — a statement that is true, but not very informative.
Big $\Omega$ notation.

$f(n)$ is said to be $\Omega(g(n))$ iff $\exists C, n_0 > 0$ such that $\forall n \geq n_0$, $f(n) \geq C \cdot g(n)$ (method 2).

Or

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = \text{constant } C.$$

If $f(n)$ is $\Omega(g(n))$, then $g(n)$ is a lower bound on $f(n)$.

In fact, if $f(n)$ is $O(g(n))$, then $g(n)$ is $\Omega(f(n))$ [why?].

Examples:

- $f(n) = 10n^2 + 4n + 2$, $g(n) = 100n + 40$.

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0 = \text{constant}. \therefore f(n) \text{ is } \Omega(g(n)).$$

Also, $f(n) \geq 1 \cdot g(n) \forall n \geq 10$ ($C=1, n_0=10$).

Examples:

- $f(n) = 3n + 10$, $g(n) = n + 1000$.

In this case $f(n)$ is $O(g(n))$ and $f(n)$ is also $\Omega(g(n))$.

Note: $g(n)$ can be a loose lower bound on $f(n)$ if $f(n)$ is $\Omega(g(n))$. It's like saying that all M.Sc. IT students are at least 1 year old :-)
\( \Theta \) notation:

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

The \( \Theta \)-notation is asymptotically "tight", since it provides both an upper bound as well as a lower bound.

Example of a real program

Selection sort: It is a strategy to select the largest element of an array and move it to the last location of the array, i.e., to \( \text{arr}[n] \). Then the largest number from \( \text{arr}[0 \text{ to } n-2] \) is moved to \( \text{arr}[n-2] \). Then the largest number from \( \text{arr}[0 \text{ to } n-3] \) is moved to \( \text{arr}[n-3] \). And so on.

```c
#include <stdio.h>

void SelectSort ( int * arr, int n )
{
    for ( int j = n; j > 1; j -- )
    {
        int k = max ( arr, j );
        swap ( & arr[j], & arr[j-1] );
    }
}
```

The routine \( \text{max}(\text{arr}, j) \) finds the maximum
element in the array from location 0 to j. Executing Max will require j-1 comparisons.
The routine Max will be invoked n-1 times.
The total number of comparisons
\[= (n-1) + (n-2) + \cdots + 3 + 2 + 1\]
\[= \frac{n(n-1)}{2} = O(n^2) \text{ or } \Omega(n^2)\]
\[\rightarrow \text{ in fact } \Theta(n^2)\]