CSc 345 — Analysis of Discrete Structures
(McCann)

# Asymptotic Notation: $O()$, $o()$, $\Omega()$, $\omega()$, and $\Theta()$

## The Idea

"Big-O" notation was introduced in P. Bachmann's 1892 book *Analytische Zahlentheorie*. He used it to say things like "$x$ is $O(\frac{n}{2})$" instead of "$x \approx \frac{n}{2}$." The notation works well to compare algorithm efficiencies because we want to say that the growth of effort of a given algorithm approximates the shape of a standard function.

## The Definitions

Big-O ($O()$) is one of five standard asymptotic notations. In practice, Big-O is used as a tight upper-bound on the growth of an algorithm's effort (this effort is described by the function $f(n)$), even though, as written, it can also be a loose upper-bound. To make its role as a tight upper-bound more clear, "Little-o" ($o()$) notation is used to describe an upper-bound that cannot be tight.

> **Definition** (Big–O, $O()$): Let $f(n)$ and $g(n)$ be functions that map positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ (or $f(n) \in O(g(n))$) if there exists a real constant $c > 0$ and there exists an integer constant $n_0 \geq 1$ such that $f(n) \leq c * g(n)$ for every integer $n \geq n_0$.

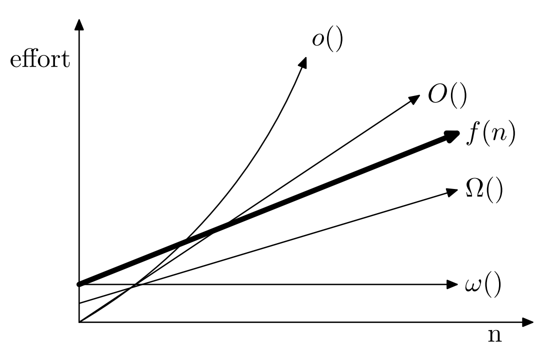
> **Definition** (Little–o, $o()$): Let $f(n)$ and $g(n)$ be functions that map positive integers to positive real numbers. We say that $f(n)$ is $o(g(n))$ (or $f(n) \in o(g(n))$) if for any real constant $c > 0$, there exists an integer constant $n_0 \geq 1$ such that $f(n) < c * g(n)$ for every integer $n \geq n_0$.

On the other side of $f(n)$, it is convenient to define parallels to $O()$ and $o()$ that provide tight and loose lower bounds on the growth of $f(n)$. "Big-Omega" ($\Omega()$) is the tight lower bound notation, and "little-omega" ($\omega()$) describes the loose lower bound.

> **Definition** (Big–Omega, $\Omega()$): Let $f(n)$ and $g(n)$ be functions that map positive integers to positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ (or $f(n) \in \Omega(g(n))$) if there exists a real constant $c > 0$ and there exists an integer constant $n_0 \geq 1$ such that $f(n) \geq c \cdot g(n)$ for every integer $n \geq n_0$.

> **Definition** (Little–Omega, $\omega()$): Let $f(n)$ and $g(n)$ be functions that map positive integers to positive real numbers. We say that $f(n)$ is $\omega(g(n))$ (or $f(n) \in \omega(g(n))$) if for any real constant $c > 0$, there exists an integer constant $n_0 \geq 1$ such that $f(n) > c \cdot g(n)$ for every integer $n \geq n_0$.

This graph should help you visualize the relationships between this notations:




(Continued...)

These definitions have far more similarities than differences. Here's a table that summarizes the key restrictions in these four definitions:

| Definition | $\boxed{?}\ c > 0$ | $\boxed{?}\ n_0 \geq 1$ | $f(n)\ \boxed{?}\ c \cdot g(n)$ |
|---|---|---|---|
| $O()$ | $\exists$ | $\exists$ | $\leq$ |
| $o()$ | $\forall$ | $\exists$ | $<$ |
| $\Omega()$ | $\exists$ | $\exists$ | $\geq$ |
| $\omega()$ | $\forall$ | $\exists$ | $>$ |

While $\Omega()$ and $\omega()$ aren't often used to describe algorithms, we can build on them (on $\Omega()$ in particular) to define a notation that describes a combination of $O()$ and $\Omega()$: "Big-Theta" ($\Theta()$). When we say that an algorithm is $\Theta(g(n))$, we are saying that $g(n)$ is both a tight upper-bound and a tight lower-bound on the growth of the algorithm's effort.

**Definition** (Big–Theta, $\Theta()$): Let $f(n)$ and $g(n)$ be functions that map positive integers to positive real numbers. We say that $f(n)$ is $\Theta(g(n))$ (or $f(n) \in \Theta(g(n))$) if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

## Application Examples

Here are a few examples that show how the definitions should be applied.

1. Let $f(n) = 7n + 8$ and $g(n) = n$. Is $f(n) \in O(g(n))$?

   For $7n + 8 \in O(n)$, we have to find $c$ and $n_0$ such that $7n + 8 \leq c \cdot n$, $\forall n \geq n_0$. By inspection, it's clear that $c$ must be larger than 7. Let $c = 8$.

   Now we need a suitable $n_0$. In this case, $f(8) = 8 \cdot g(8)$. Because the definition of $O()$ requires that $f(n) \leq c \cdot g(n)$, we can select $n_0 = 8$, or any integer above 8 – they will all work.

   We have identified values for the constants $c$ and $n_0$ such that $7n + 8$ is $\leq c \cdot n$ for every $n \geq n_0$, so we can say that $7n + 8$ is $O(n)$.

   (But how do we know that this will work for every $n$ above 7? We can prove by induction that $7n+8 \leq 8n$, $\forall n \geq 8$. Be sure that you can write such proofs if asked!)

2. Let $f(n) = 7n + 8$ and $g(n) = n$. Is $f(n) \in o(g(n))$?

   In order for that to be true, for any $c$, we have to be able to find an $n_0$ that makes $f(n) < c \cdot g(n)$ asymptotically true.

   However, this doesn't seem likely to be true. Both $7n + 8$ and $n$ are linear, and $o()$ defines loose upper-bounds. To show that it's not true, all we need is a counter–example.

   Because any $c > 0$ must work for the claim to be true, let's try to find a $c$ that won't work. Let $c = 100$. Can we find an $n_0$ such that $7n + 8 < 100n$? Sure; let $n_0 = 10$. Try again!

   OK, let $c = \frac{1}{100}$. Can we find an $n_0$ such that $7n + 8 < \frac{n}{100}$? Of course not. Therefore, $7n + 8 \notin o(n)$, meaning $g(n) = n$ is not a loose upper-bound on $7n + 8$.

3. Is $7n + 8 \in o(n^2)$?

   Again, to claim this we need to be able to argue that for any $c$, we can find an $n_0$ that makes $7n+8 < c \cdot n^2$. Let's try examples again to make our point, keeping in mind that we need to show that we can find an $n_0$ for any $c$.

   If $c = 100$, the inequality is clearly true. If $c = \frac{1}{100}$, we'll have to use a little more imagination, but we'll be able to find an $n_0$. (Try $n_0 = 1000$.)

   At this point, it should be clear that, regardless of the $c$ we choose, we'll be able to get $c \cdot n^2$ to dominate $7n+8$ eventually (that is, we can find a large enough $n_0$ to make the definition hold). Thus, $7n+8 \in o(n^2)$.