The set of functions O(g(n))

1. Examples of sets O(…) and the definition of the set O(g(n))
2. Proofs of f(n) in O(g(n))
3. Set O(g(m, n)) and others

1. Examples of sets O(…) and the definition of the set O(g(n))

Let f(n) be the number of basic steps made by executing an algorithm, depending on input value n. We want to classify such functions (and algorithms) depending on how f(n) behaves as n grows large —we don't care about what happens when n is small. For example:

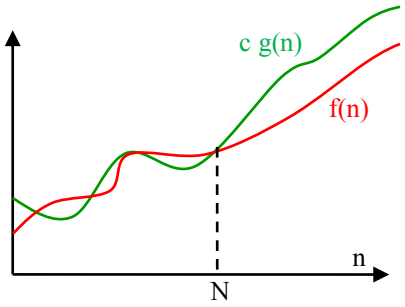
O(1) is the set of constant functions f(n).
Example: $f(n) = 5$ is in O(1). Since O(1) is a set, we can write this as $f(n) \in O(1)$.
Example: $f(n) = 800$ is in O(1).

O(n) is the set of functions that are no larger than linear in n.
Example: $f(n) = n + 50$ is in O(n).
Example: $f(n) = 5$ is in O(n).
Example: $f(n) = n^2$ is *not* in O(n).

$O(n^2)$ is the set of functions that are no larger than quadratic in n.|
Example: $f(n) = n^2 - 10n + 900$ is in $O(n^2)$.
Example: $f(n) = 5$ is in $O(n^2)$.
Example: $f(n) = n^3$ is *not* in $O(n^2)$.

Such a classification will allow us to compare, discuss, and choose algorithms for a particular task. For example, if we have two sorting algorithms, one taking time $O(n^2)$[1] and the other taking time $O(n^3)$, we would rather use the one taking time $O(n^2)$ if large arrays are to be sorted.

We need a general definition of the set of functions O(g(n)), where g(n) is itself a function. This definition of O(g(n)) is best understood using the graph to the right. The x-axis is n, starting at 0 and growing to the right. A function f(n) is shown in red, and a function c * g(n) (for some positive constant c) is shown in green. For $n < N$, sometimes $f(n) \le c * g(n)$ and sometimes $f(n) > c * g(n)$. But for all integers n, $N \le n$, $f(n) \le c * g(n)$. Therefore, f(n) is in O(g(n)), which we write as $f(n) \in O(g(n))$.



With the aid of this graph, we define the set O(g(n)) as follows:

Definition. Function f(n) is in set O(g(n)) iff there exist positive constants c and N such that for all n, $n \ge N$, $f(n) \le c * g(n)$.

2. Proofs of: f(n) ∈ O(g(n))

We give three proofs of $f(n) \in O(g(n))$. Reading these proofs and developing your own proofs will give you more understanding for the definition of O(g(n)). In developing such proofs, we find it best to start with f(n) and change it using =, <, and ≤ steps into c * g(n). As the proof progresses, some step may not be possible without deciding on what c or N should be, and the structure of the proof being developed will help us choose them.

Proof of $f(n) = 30*n + 6 \in O(n)$. Here, $g(n) = n$. We start with f(n) and end up with c*g(n):

```
    f(n)
=     <Definition of f(n)>
    30n + 6
≤     <Choose N = 6, so we consider only values of n that are >= 6. This allows us to replace 6 by n >
```

[1] "The algorithm taking time $O(n^2)$" is an informal way of saying this: Let the number of basic steps executed by the algorithm be bounded above by f(n); then $f(n) \in O(n^2)$.

$30n + n$
= <Arithmetic>
$31n$
= <Chose c = 31, and use the definition of $g(n)$>
$c * g(n)$

Proof of $6n^2 + 6n - 5 \in O(n^2)$. Here, $f(n) = 6n^2 + 6n - 5$ and $g(n) = n^2$. We transform $f(n)$ into $c * g(n)$.

$6n^2 + 6n + 5$
< <Choose $N = 3$, because for $n \geq 3$, $5 < n^2$>
$6n^2 + 6n + n^2$
< <For $n \geq 3$, $6n < 6n^2$>
$6n^2 + 6n^2 + n^2$
= <Arithmetic>
$13n^2$
= <Choose $c = 13$, use definition of $g(n)$>
$c * g(n)$

Proof of $8(\log n) + 6 \in O(\log n)$. This proof requires knowing a little bit about logarithms. In general, we will not require proofs that depend on knowledge of algorithms because not everyone has that knowledge. More important than that little bit of knowledge is the ability to develop these proofs. The proof below relies only on a basic property of logs to the base 2: If $x = 2^y$, the $\log x = y$.

$8(\log n) + 6$
≤ <Choose $N = 2^6 = 64$, because for $n \geq 2^6$, $\log n \geq 6$>
$8(\log n) + \log n$
= <Arithmetic>
$9(\log n)$
= <Choose $c = 9$, use definition of $g(n)$>
$c * g(n)$

3. The set O(g(m, n)) and others

Suppose an algorithm uses two arrays of sizes m and n. Then the number of basic steps performed during execution will be some function $f(m, n)$ —it may depend on the sizes of the two arrays. For example, the number of basic steps could be $5m + 6m \log n + 10$. We would say that this function is in $O(m \log n)$. Thus, we need to define $O(g(m, n))$, where g has two parameters —and perhaps $O(g(m, n, p))$ if an algorithm uses three arrays, etc.

Below is a formal definition of $O(g(m, n))$, just for completeness, but it's not something to be concerned about at this point, and we won't ask you to define it formally. However every once in a while, the use of something like $O(m \log n)$ will arise naturally.

Definition. Function $f(m, n)$ is in set $O(g(m, n))$ iff there exist positive constants c and N such that for all n, $n \geq N$, and all m, $m \geq N$, $f(m, n) \leq c * g(m, n)$.