Complexity

1 Background: Analysis of Algorithms

You know from other computer science courses that we can analyze the running time complexity of an algorithm and categorize it using "big-oh" notation. For example, if we were to sum the elements in an array with n elements, we would initialize the sum variable once, initialize the index once, and then iteratively add the next array item to the sum and increment our index, each time checking to see if the index has reached n. This would entail n executions of the body of the loop and $n + 1$ executions of the guard. If we assume each static operation takes c_0, c_1, . . . units of computing time, then we have

```
int sum = 0;            c_0
int i = 0;              c_1
while(i < 0) {          c_2 · (n + 1)
    sum += a[i]         c_3 · n
    i++                 c_4 · n
}
```

Thus the running time of the algorithm (in some unknown unit of time), as a function of the size of the array, is

$$T(n) = c_0 + c_1 + c_2 \cdot (n + 1) + c_3 \cdot n + c_4 \cdot n$$

which, with a change of variables, simplifies to

$$T(n) = d_0 + d_1 \cdot n$$

Since the term in n dominates and since the d_1 coefficient is unknown and depends upon, among other things, the machine on which we write the program, we say that this time function is "on the order of" n, we say that the algorithm *is* $O(n)$, and we even abuse mathematical notation so far as to say $T(n) = O(n)$.

We will not be analyzing algorithms (except for the one analysis we did above). Instead we will explain carefully what we mean when we categorize algorithms and their time function into big-oh and related categories. The first thing to note is that $T(n)$ is a function; it is the function which relates the size of the input to the running time of the algorithm. Furthermore, strictly speaking, it is the function, not the algorithm, that "is" $O(n)$. Or goal is to understand what that means, and our tool will be that great tool which we always use to mold informal muck into mathematical rigor: the theory of sets.

2 Big-oh categorization

The *complexity class on the order* of a function f is the set of functions that are asymptotically bounded by a f scaled by some constant. Symbolically, if $f : \mathbb{W} \to \mathbb{R}^+$, then

complexity class on the order

$$O(f) = \{\ g : \mathbb{W} \to \mathbb{R}^+ \mid \exists\ N \in \mathbb{W} \text{ and } c \in \mathbb{R}^+ \text{ such that } \forall\ n > N,\ g(n) \leq c \cdot f(n)\ \}$$

Don't be intimidated by all those symbols. You've taken calculus, and this is not much different from what you need for understanding limits. Some examples below will illuminate this.

But first notice that $O(n)$, for example, is technically a set. It's a set of functions. This should make sense, since what we're doing with complexity classes is categorization. Moreover, even though we write informally that $T(n) = O(n)$ or $d_0 + d_1 \cdot n = O(n)$, what we really mean is that $T(n) \in O(n)$.

We should make another comment on notation, that it is still far from perfect. We should name our set $O(\langle$ some function $\rangle)$. But how do we express our "some function"? In the formal definition above, I assumed we used the function *name*, hence $O(f)$. However, we traditionally write it with the function *description* (or *rule* or *body*), that is $O(n)$. This is confusing because there's nothing in the notation to show that n is a function, and not just a value or parameter. We have no way of referring to just a function, not a function name or a function description.

But there does exist a notation that allows us to do this: lambda notation. In such notation, the expression $\lambda n.(3 \cdot n^2 + 5)$ would mean "the function that takes a parameter called n and returns $3 \cdot n^2 + 5$." This is exactly equivalent to anonymous functions in ML; compare with

```
fn n => 3 * n * n + 5
```

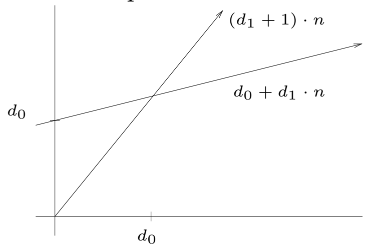
Thus it would be best to call this category $O(\lambda n.n)$. Unfortunately, lambda notation is not familiar enough for this to be practical, and it's not as easy for algebraic manipulation in practice.

Theorem 1 *For all $d_0, d_1 \in \mathbb{R}$, $d_0 + d_1 \cdot n \in O(n)$.*

Proof. Pick $c = d_1 + 1$ and $N = d_0$. Then if $n > N$,

$$d_0 + d_1 \cdot n < n + d_1 \cdot n = (d_1 + 1) \cdot n = c \cdot n$$

□

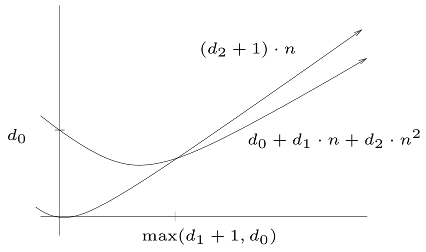


Theorem 2 *For all $d_0, d_1, d_2 \in \mathbb{R}$, $d_0 + d_1 \cdot n + d_2 \cdot n^2 \in O(n^2)$.*

Proof. Pick $c = d_2 + 1$ and $N = \max(d_1 + 1, d_0)$. Then if $n > N$,

$$\begin{array}{rcll}
\frac{d_0}{n} & < & 1 & \text{since } n > d_0 \\
\frac{d_0}{n} + d_1 & < & n & \text{since } n > d_1 + 1 \\
d_0 + d_1 \cdot n & < & n^2 & \text{by multiplying by } n \\
d_0 + d_1 \cdot n + d_2 \cdot n^2 & < & n^2 + d_2 \cdot n^2 & \text{by adding } d_2 \cdot n^2 \\
& = & (d_2 + 1) \cdot n & \\
& = & c \cdot n &
\end{array}$$

□



If these proofs look intimidating (how would one come up with that sequence of inequalities?), notice that it's much easier to reason our way in the reverse order from which we present things in the proof. To derive the previous proof, start by saying "I want to find c and N so that $d_0+d_1\cdot n+d_2\cdot n^2 < c \cdot n$." Since we know (intuitively) that the n^2 term is going to dominate, we have to pick a c that's a little bit bigger than d_2. Thus $c = d_2 + 1$ will work. Now we want to show

$$d_0 + d_1 \cdot n + d_2 \cdot n^2 < (d_2 + 1) \cdot n$$

Then, using algebraic manipulation, you can (almost) isolate n to the form

$$\frac{d_0}{n} + d_1 < n$$

Then what remains is the insight that if $n > d_0$, then $\frac{d_0}{n} < 1$, and thus $\frac{d_0}{n} + d_1 < d_1 + 1$. So if n is also greater than $d_1 + 1$, we have $\frac{d_0}{n} + d_1 < n$. This helps us choose $N = \max(d_1 + 1, d_0)$.

Finally, since big-oh represents an upper bound, we can say

Theorem 3 *For all* $d_0, d_1 \in \mathbb{R}$, $d_0 + d_1 \cdot n \in O(n^2)$.

Proof. Pick $c = d_1 + 1$ and $N = \max(d_0, 1)$. Then if $n > N$,

$$d_0 + d_1 \cdot n < n + d_1 \cdot n = (d_1 + 1) \cdot n = c \cdot n < c \cdot n^2$$

We know the last part of that is true only because $n > N \geq 1$.

□

Corollary 1 $O(n) \subseteq O(n^2)$.

This draws heavily on Cormen et al, *Introduction to Algorithms*, second edition, McGraw-Hill / MIT Press, 2001.