COSC242 Lecture 2

We saw last time that the amount of work done by Insertion Sort, in the worst case, is roughly indicated by
$f(n) = 1 + 2 + 3 + \ldots + (n-1) = n(n-1)/2 = (n^2 - n)/2$

We'd like to tie this in to some special **landmark functions,** which are given by assigning to inputs n the outputs

$$f(n) = 1$$
$$f(n) = \log_2 n$$
$$f(n) = n$$
$$f(n) = n \log_2 n$$
$$f(n) = n^2$$
$$f(n) = n^3$$
$$\ldots$$
$$f(n) = 2^n$$
$$f(n) = n!$$

The landmarks tell us how bad an algorithm can get, i.e. how hard it might have to work.

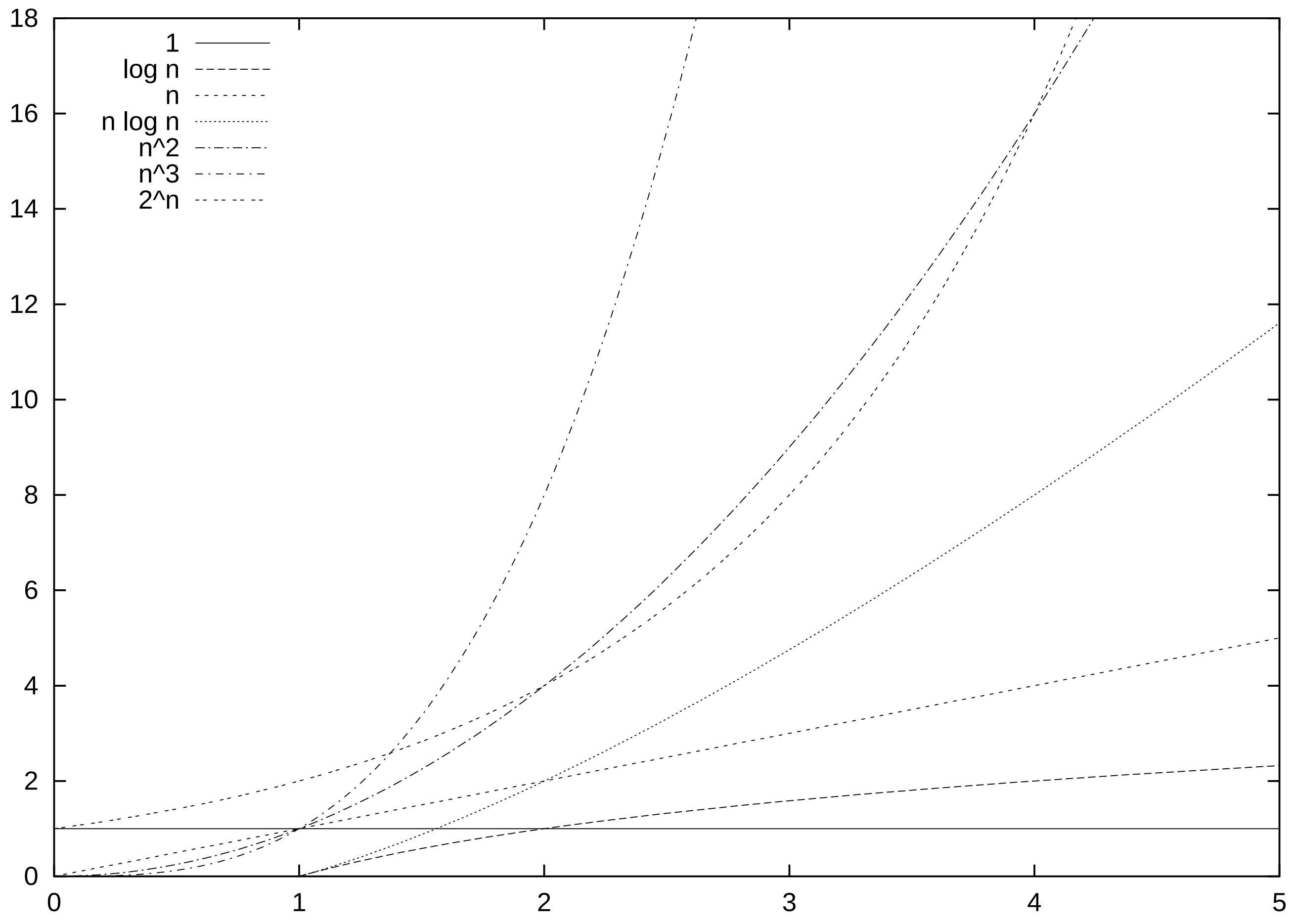
Given an algorithm, we estimate how much work it would have to do in the worst case (call this estimate the time-complexity function of the algorithm) and then we see which landmark it should be grouped with.

The best landmarks are at the top, the bad ones low down.

Rates of growth

The landmark functions have different rates of growth.

By the *rate of growth* of f we mean how fast its output $f(n)$ increases in size as the input n gets bigger. Intuitively, a function with a slow rate of growth **scales up more efficiently** than a function with a high rate of growth.



Introducing Θ and big-O

Suppose an algorithm's time-complexity function is f. Which landmark function g should f be grouped with?

We will write $f = \Theta(g)$ to say that f is grouped with g. But to understand Θ notation we first define "big-O" notation.

Intuitively, "$f = O(g)$" means f is no worse than g, i.e. f **scales up at least as well as** g, and maybe better. How can we make this precise?

It's tricky. Some values of f may make f look worse than g even though $f = O(g)$.

For instance, we'll show that $6n = O(n^2)$. But if $n = 1$ then $6n = 6$ whereas $n^2 = 1$ so for this value of n, the function $f(n) = 6n$ is worse than the landmark n^2.

Since we're interested in how the f scales up, we ignore small values of n and look at the picture when n gets big.

So our precise definition of "big-O" will allow $f = O(g)$ even if $f(n)$ is is bigger than $g(n)$ for small values of n, as long as f stays below g when n gets big enough.

Formal definition for big-O

Mathematicians like to condense their ideas down to a few short symbols. This allows them to say a lot in a small space, which helps avoid mistakes caused by important things being too far apart.

Definition of “big-O”:
$f = O(g)$ iff there are positive integers c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

In other words, to show that any function $f = O(g)$, we have to find two positive whole numbers, one called c and the other called n_0, so that any output $f(n)$ is no bigger than the output $g(n)$ multiplied by the constant c (at least, any output produced by inputs from the starting point n_0 onwards).

Note that c and n_0 are **positive**, so $c \geq 1$ and $n_0 \geq 1$. Our definition does not allow $c = 0$ nor does it allow $n_0 = 0$.

Example (in class): Let’s show that $6n = O(n^2)$.

big-O proof for Insertion Sort

Consider insertion sort again, which has time complexity $f(n) = n(n-1)/2$. We now show that $n(n-1)/2 = O(n^2)$.

Proof: It is possible to find c and n_0 such that

$$n(n-1)/2 \leq c \cdot n^2 \text{ for all } n \geq n_0.$$

In fact, c and n_0 can both be 1 in this proof.

If $n \geq 1$ then $n \neq 0$ and so

$$n(n-1)/2 = (n^2 - n)/2 = n^2/2 - n/2 \leq 1 \cdot n^2 = n^2.$$

Can you see where we use the fact that $n \geq n_0$?

And can you see where we use the fact that $c = 1$?

And do you see that since $n(n-1)/2$ is actually $< n^2$, it is also $\leq n^2$?

Θ Notation

It's nice to be able to say that something is no worse than something else, but what about saying it is no better either?

Simple. If $f = O(g)$ says that f is no worse than g then it is also saying that g is no better than f.

So if we want to say that f is no better than g, we may write $g = O(f)$.

Suppose it is true that $f = O(g)$ and also that $g = O(f)$.

This says that f is no worse than g, and f is no better than g. In other words, f scales up about as well as g, so in terms of efficiency, f is **equivalent** to g.

When $f = O(g)$ and $g = O(f)$, then we may write $f = \Theta(g)$. In English, it says "f is equivalent to g in efficiency."

For example, we'll see that $f = \Theta(g)$ where $f(n) = 5$ and $g(n) = 1$. This makes sense: two straight lines scale up about equally well. More formally: linear functions are equivalent in efficiency.

Θ proof for Insertion Sort

To prove that $f = \Theta(g)$, the important thing is to remember that there are **two** things to show. We must prove that $f = O(g)$, and we must prove that that $g = O(f)$.

To take insertion sort again:

I claim that $\frac{n(n-1)}{2} = \Theta(n^2)$.

We've already shown that $\frac{n(n-1)}{2} = O(n^2)$,
so now just show that $n^2 = O(\frac{n(n-1)}{2})$.

If we choose $c = 3$ and $n_0 = 3$, then

$$n^2 \leq c \cdot \frac{n(n-1)}{2} \text{ for all } n \geq n_0$$

because:

$$\frac{3n(n-1)}{2} = \frac{3}{2}n^2 - \frac{3}{2}n = n^2 + \frac{n^2}{2} - \frac{3n}{2} = n^2 + \frac{(n^2-3n)}{2} \geq n^2$$

since $n \geq 3$ so that $\frac{(n \cdot n - 3 \cdot n)}{2} \geq 0$.

And now, since we have $\frac{n(n-1)}{2} = O(n^2)$ and $n^2 = O(\frac{n(n-1)}{2})$, we have that

$$\frac{n(n-1)}{2} = \Theta(n^2).$$

Exercises

In these exercises, don't just give a yes/no answer. Write out your reasoning in full, making use of the definitions.

1. Suppose $f(n) = 5$ for all n.
   (a) Is $f = O(1)$?
   (b) Is $f = \Theta(1)$?
2. Suppose $f(n) = 3n + 6$ for all n.
   (a) Is $f = O(1)$?
   (b) Is $f = O(n)$?
   (c) Is $f = \Theta(n)$?
   (d) Is $f = O(n^2)$?
   (e) Is $f = \Theta(n^2)$?
3. Suppose $f(n) = 2n^2 + 3$ for all n.
   (a) Is $f = O(n^2)$?
   (b) Is $f = \Theta(n^2)$?
   (c) Is $f = O(n^3)$?
   (d) To think about: Is $f = \Theta(n^3)$?