Chapter 5

Analyzing Algorithms

So far we have been proving statements about databases, mathematics and arithmetic, or sequences of numbers. Though these types of statements are common in computer science, you'll probably encounter algorithms most of the time. Often we want to reason about algorithms and even prove things about them. Wouldn't it be nice to be able to *prove* that your program is correct? Especially if you're programming a heart monitor or a NASA spacecraft?

In this chapter we'll introduce a number of tools for dealing with computer algorithms, formalizing their expression, and techniques for analyzing properties of algorithms, so that we can prove correctness or prove bounds on the resources that are required.

5.1 Binary (base 2) notation

Let's first think about numbers. In our everyday life, we write numbers in decimal (base 10) notation (although I heard of one kid who learned to use the fingers of her left hand to count from 0 to 31 in base 2). In decimal, the sequence of digits 20395 represents (parsing from the right):

$$5 + 9(10) + 3(100) + 0(1000) + 2(10000) =$$

$$5(10^0) + 9(10^1) + 3(10^2) + 0(10^3) + 2(10^4)$$

Each position represents a power of 10, and 10 is called the BASE. Each position has a digit from $[0, 9]$ representing how many of that power to add. Why do we use 10? Perhaps due to having 10 fingers (however, humans at various times have used base 60, base 20, and mixed base 20,18 (Mayans)). In the last case there were $(105)_{20,18}$ days in the year. Any integer with absolute value greater than 1 will work (so experiment with base -2).

Consider using 2 as the base for our notation. What digits should we use?[1] We don't need digits 2 or higher, since they are expressed by choosing a different position for our digits (just as in base 10, where there is no single digit for numbers 10 and greater).

Here are some examples of binary numbers:

$$(10011)_2$$

represents

$$1(2^0) + 1(2^1) + 0(2^2) + 0(2^3) + 1(2^4) = (19)_{10}$$

We can extend the idea, and imitate the decimal point (with a "binary point"?) from base 10:

$$(1011.101)_2 = 19\frac{5}{8}$$

How did we do that?[2] Here are some questions:

- How do you multiply two base 10 numbers?[3] Work out 37×43.
- How do you multiply two binary numbers?[4]
- What does "right shifting" (eliminating the right-most digit) do in base 10?[5]
- What does "right shifting" do in binary?[6]
- What does the rightmost digit tell us in base 10? In binary?

Convert some numbers from decimal to binary notation. Try 57. We'd like to represent 57 by adding either 0 or 1 of each power of 2 that is no greater than 57. So $57 = 32 + 16 + 8 + 1 = (111001)_2$. We can also fill in the binary digits, systematically, from the bottom up, using the % operator (the remainder after division operator, at least for positive arguments):

$$\begin{aligned} 57\%2 = 1 &\quad \textit{so} \quad (?????1)_2 \\ (57-1)/2 = 28\%2 = 0 &\quad \textit{so} \quad (????01)_2 \\ 28/2 = 14\%2 = 0 &\quad \textit{so} \quad (???001)_2 \\ 14/2 = 7\%2 = 1 &\quad \textit{so} \quad (??1001)_2 \\ (7-1)/2 = 3\%2 = 1 &\quad \textit{so} \quad (?11001)_2 \\ (3-1)/2 = 1\%2 = 1 &\quad \textit{so} \quad (111001)_2 \end{aligned}$$

Addition in binary is the same as (only different from...) addition in decimal. Just remember that $(1)_2 + (1)_2 = (10)_2$. If we add two binary numbers, this tells us when to "carry" 1:

$$\begin{array}{rr} & 1011 \\ + & 1011 \\ \hline & 10110 \end{array}$$

LOG$_2$

How many 5-digit binary numbers are there (including those with leading 0s)? These numbers run from $(00000)_2$ through $(11111)_2$, or 0 through 31 in decimal — 32 numbers. Another way to count them is to consider that there are two choices for each digit, hence 2^5 strings of digits. If we add one more digit we get twice as many numbers. Every digit doubles the range of numbers, so there are two 1-digit binary numbers (0 and 1), four 2-digit binary numbers (0 through 3), 8 3-digit binary numbers (0 through 7), and so on.

Reverse the question: how many digits are required to represent a given number. In other words, what is the smallest integer power of 2 needed to exceed a given number? $\log_2 x$ is the power of 2 that gives $2^{\log_2 x} = x$. You can think of it as how many times you must multiply 1 by 2 to get x, or roughly the number of digits in the binary representation of x. (The precise number of digits needed is $\lfloor (\log_2 x) + 1 \rfloor$, which is equal to (why?) $\lfloor \log_2 x \rfloor + 1$).

Loop invariant for base 2 multiplication

Integers are naturally represented on a computer in binary, since a gate can be in either an on or off (1 or 0) position. It is very easy to multiply or divide by 2, since all we need to do is perform a left or right shift (an easy hardware operation). Similarly, it is also very easy to determine whether an integer is even or odd.

Putting these together, we can write a multiplication algorithm that uses these fast operations:

```
public class MultiplicationExample {

  /**
   * mult multiplies m times n.
   * @arg m a natural number
   * @arg n an integer
   * @return mn
   * precondition: m >= 0
   */
  public static int mult(int m, int n) {
    int x = m;
    int y = n;
    int z = 0;
    // loop invariant: z = mn - xy
    while (x != 0) {
      if (x % 2 == 1) { // x odd
        z = z + y;
      }
      x = x >> 1; // x = x div 2 (right shift)
      y = y << 1; // y = 2y (left shift)
    }
    // post condition: z = mn
    return z;
  }

}
```

After reading this algorithm, there is no reason you should believe it actually multiplies two integers: we'll need to prove it to you. Let's consider the precondition first. We can always ensure that $m \geq 0$ (how?[7]). The postcondition states that z, the value that is returned, is equal to the product of m and n (that would be nice, but we're not convinced).

Let's look at the stated loop invariant. A LOOP INVARIANT is a relationship between the variables that is always true at the start and at the end of a loop iteration (we'll need to prove this). It's sufficient to verify that the invariant is true at the start of first iteration, and verify that if the invariant is true at the start of any iteration, it must be true at the end of the iteration.[8] Before we start the loop, we set $x = m$, $y = n$ and $z = 0$, so it is clear that $z = mn - xy = mn - mn = 0$. Now we need to show that if $z = mn - xy$ before executing the body of the loop, and $x \neq 0$, then after executing the loop body, $z = mn - xy$ is still true (can you write this statement formally?). Here's a sketch of a proof:

> Let $x', y', z', x'', y'', z'', m, n \in \mathbb{Z}$, and assume the $'$ elements related to the $''$ elements by the action of the loop. Assume $m \geq 0$. Observe that the values of m and n are never changed in the loop.
>
> Assume $z' = mn - x'y'$.
>
> Case 1: x' odd.
>
> Then $z'' = z' + y'$, $x'' = (x'-1)/2$, and $y'' = 2y'$.
>
> So
>
> $$\begin{aligned} mn - x''y'' &= mn - (x'-1)/2 \cdot 2y' \quad \text{(since } x' \text{ is odd)} \\ &= mn - x'y' + y' \\ &= z' + y' \\ &= z'' \end{aligned}$$
>
> Case 2: x' even.
>
> Then $z'' = z'$, $x'' = x'/2$, and $y'' = 2y'$.

So

$$\begin{aligned} mn - x''y'' &= mn - x'/2 \cdot 2y' \\ &= mn - x'y' \\ &= z' \\ &= z'' \end{aligned}$$

Since x' is either even or odd, in all cases $mn - x''y'' = z''$
Thus $mn - x'y' = z' \Rightarrow mn - x''y'' = z''$.
Since $x', x'', y', y'', z', z'', m, n$ are arbitrary elements of $\mathbb{Z}$,
$\forall x', x'', y', y'', z', z'', m, n \in \mathbb{Z}, mn - x'y' = z' \Rightarrow mn - x''y'' = z''$.

We should probably verify the postcondition to fully convince ourselves of the correctness of this algorithm. We've shown the loop invariant holds, so let's see what we can conclude when the loop terminates (i.e., when $x = 0$). By the loop invariant, $z = mn - xy = mn - 0 = mn$, so we know we must get the right answer (assuming the loop eventually terminates).

We should now be fairly convinced that this algorithm is in fact correct. One might now wonder, how many iterations of the loop are completed before the answer is returned?

5.2 Run time and constant factors

When calculating the running time of a program, we may know how many basic "steps" it takes as a function of input size, but we may not know how long each step takes on a particular computer. We would like to estimate the overall running time of an algorithm while ignoring constant factors (like how fast the CPU is). So, for example, if we have 3 machines, where operations take 3μs, 8μs and 0.5μs, the three functions measuring the amount of time required, $t(n) = 3n^2$, $t(n) = 8n^2$, and $t(n) = n^2/2$ are considered the same, ignoring ("to within") constant factors (the time required always grows according to a quadratic function in terms of the size of the input n).

The nice thing is that this means that lower order terms can be ignored as well! So $f(n) = 3n^2$ and $g(n) = 3n^2 + 2$ are considered "the same", as are $h(n) = 3n^2 + 2n$ and $j(n) = 5n^2$. Notice that

$$\forall n \in \mathbb{N}, n \geq 1 \Rightarrow f(n) \leq g(n) \leq h(n) \leq j(n)$$

but there's always a constant factor that can reverse any of these inequalities.

Really what we want to measure is the growth rate of functions (and in computer science, the growth rate of functions that bound the running time of algorithms). You might be familiar with binary search and linear search (two algorithms for searching for a value in a sorted array). Suppose one computer runs binary search and one computer runs linear search. Which computer will give an answer first, assuming the two computers run at roughly the same CPU speed? What if one computer is much faster (in terms of CPU speed) than the other, does it affect your answer? What if the array is really, really big?

How large is "sufficiently large?"

Is binary search a better algorithm than linear search?[9] It depends on the size of the input. For example, suppose you established that linear search has complexity $L(n) = 3n$ and binary search has complexity $B(n) = 9\log_2 n$. For the first few n, $L(n)$ is smaller than $B(n)$. However, certainly for $n > 10$, $B(n)$ is smaller, indicating less "work" for binary search.

When we say "large enough" n, we mean we are discussing the asymptotic behaviour of the complexity function, and we are prepared to ignore the behaviour near the origin.

5.3 Asymptotic notation: Making Big-O precise

We define $\mathbb{R}^{\geq 0}$ as the set of nonnegative real numbers, and define $\mathbb{R}^{+}$ as the set of positive real numbers. Now here's the precise definition of "The set of functions that, ignoring a constant, are eventually no more than f":

Definition: For any function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ (i.e., any function mapping naturals to nonnegative reals), let

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \exists c \in \mathbb{R}^{+}, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow g(n) \leq cf(n)\}.$$

Saying $g \in O(f)$ says that "g grows no faster than f" (or equivalently, "f is an upper bound for g), so long as we modify our understanding of "growing no faster" and being an "upper bound" with the practice of ignoring constant factors. Now we can prove some theorems.

Suppose $g(n) = 3n^2 + 2$ and $f(n) = n^2$. Then $g \in O(f)$. We need to prove that $\exists c \in \mathbb{R}^{+}, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 3n^2 + 2 \leq cn^2$. It's enough to find some c and B that "work" in order to prove the theorem.

Finding c means finding a factor that will scale n^2 up to the size of $3n^2 + 2$. Setting $c = 3$ almost works, but there's that annoying additional term 2. Certainly $3n^2 + 2 < 4n^2$ so long as $n \geq 2$, since $n \geq 2 \Rightarrow n^2 > 2$. So pick $c = 4$ and $B = 2$ (other values also work, but we like the ones we thought of first). Now concoct a proof of

$$\exists c \in \mathbb{R}^{+}, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 3n^2 + 2 \leq cn^2.$$

```
Let $c = 4$.
    Then $c \in \mathbb{R}^{+}$.
    Let $B = 2$.
        Then $B \in \mathbb{N}$.
        Let $n \in \mathbb{N}$ be arbitrary.
            Assume $n \geq B$.
              Then $n^2 \geq B^2 = 4$. (squaring is monotonic on natural numbers.)
              So $n^2 \geq 2$.
              So $3n^2 + n^2 \geq 3n^2 + 2$. (adding $3n^2$ to both sides of the inequality).
              So $3n^2 + 2 \leq 4n^2$.
            Thus, $n \geq B \Rightarrow 3n^2 + 2 \leq 4n^2$.
        Since $n$ is an arbitrary natural number, $\forall n \in \mathbb{N}, n \geq B \Rightarrow 3n^2 + 2 \leq 4n^2$.
    Since $B$ is a natural number, $\exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 3n^2 + 2 \leq cn^2$.
Since $c$ is a positive real number, $\exists c \in \mathbb{R}^{+}, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 3n^2 + 2 \leq cn^2$.
```

So, by definition, $g \in O(f)$.

Now suppose that $g(n) = n^4$ and $f(n) = 3n^2$. Is $g \in O(f)$? No. We can see intuitively that any constant that we multiply times $3n^2$ will be overwhelmed by the extra factor of n^2 in $g(n)$. But to show this clearly, we negate the definition and then prove the negation:

$$\forall c \in \mathbb{R}^{+}, \forall B \in \mathbb{N}, \exists n \in \mathbb{N}, n \geq B \wedge n^4 > c3n^2.$$

The parameter we have some control over is n, and we need to pick it so that $n \geq B$ and $n^4 > c3n^2$. Solve for n:

$$\begin{aligned} & n^4 > c3n^2 \\ \Leftrightarrow\ & n^4/n^2 > c3n^2/n^2 \quad \text{(when } n > 0\text{)} \\ \Leftrightarrow\ & n^2 > 3c \\ \Leftrightarrow\ & n > \sqrt{3c}. \end{aligned}$$

Notice that we were reasoning backwards (bottom up) here: we need to pick a condition on n so that $n^4 > c3n^2$ would hold (hence why everything had to be an equivalence). Notice also that we needed to assume that $n > 0$ (to avoid division by zero). So to satisfy the conditions, we can set $n = B + \lceil \sqrt{3c} \rceil + 1$. Since $\sqrt{3c}$ is not necessarily a natural number, we take its ceiling. Now we can generate the proof.

Let $c \in \mathbb{R}^+$. Let $B \in \mathbb{N}$.
    Let $n = B + \lceil\sqrt{3c}\rceil + 1$.
        Then $n \in \mathbb{N}$. (since $B \in \mathbb{N}$, $1 \in \mathbb{N}$, and $\lceil\sqrt{3c}\rceil \in \mathbb{N}$ (since $c > 0$) and $\mathbb{N}$ is closed under sums).
        So $n \geq B$ (since it is the sum of B and two other non-negative numbers).
        So $n \geq \lceil\sqrt{3c}\rceil + 1$. (since $B \geq 0$)
        So $n^2 > (\lceil\sqrt{3c}\rceil + 1)^2$.
        So $n^2 > 3c$. (ignoring some positive terms).
        So $n^4 > 3cn^2$.
    Since n is a natural number, $\exists n \in \mathbb{N}, n \geq B \wedge n^4 > c3n^2$.
Since c is an arbitrary element of $\mathbb{R}^+$ and B is an arbitrary element of $\mathbb{N}$, $\forall c \in \mathbb{R}^+, \forall B \in \mathbb{N}, \exists n \in \mathbb{N}, n \geq B \wedge n^4 > c3n^2$.
By definition, this means that $g \notin O(f)$.

A MORE COMPLEX EXAMPLE

Let's prove that $2n^3 - 5n^4 + 7n^6$ is in $O(n^2 - 4n^5 + 6n^8)$. We begin with:

Let $c =$ ____. Then $c \in \mathbb{R}^+$.
    Let $B =$ ____. Then $B \in \mathbb{N}$.
        Let $n \in \mathbb{N}$. Suppose $n \geq B$.
            Then $2n^3 - 5n^4 + 7n^6 \leq \cdots \leq c(n^2 - 4n^5 + 6n^8)$.
        Thus $\forall n \in \mathbb{N}, n \geq B \Rightarrow 2n^3 - 5n^4 + 7n^6 \leq c(n^2 - 4n^5 + 6n^8)$.
Since B is a natural number, and since c is a positive real number,
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 2n^3 - 5n^4 + 7n^6 \leq c(n^2 - 4n^5 + 6n^8)$.

To fill in the $\cdots$ we try to form a chain of inequalities, working from both ends, simplifying the expressions:

$$\begin{aligned}
2n^3 - 5n^4 + 7n^6 &\leq 2n^3 + 7n^6 && \text{(drop } -5n^4 \text{ because it doesn't help us in an important way)}\\
&\leq 2n^6 + 7n^6 && \text{(increase } n^3 \text{ to } n^6 \text{ because we have to handle } n^6 \text{ anyway)}\\
&= 9n^6\\
&\leq 9n^8 && \text{(simpler to compare)}\\
&= 2(9/2)n^8 && \text{(get as close to form of the simplified end result: now choose } c = 9/2\text{)}\\
&= 2cn^8\\
&= c(-4n^8 + 6n^8) && \text{(reading bottom up: decrease } -4n^5 \text{ to } -4n^8 \text{ because we have to handle } n^8 \text{ anyway)}\\
&\leq c(-4n^5 + 6n^8) && \text{(reading bottom up: drop } n^2 \text{ because it doesn't help us in an important way)}\\
&\leq c(n^2 - 4n^5 + 6n^8)
\end{aligned}$$

We never needed to restrict n in any way beyond $n \in \mathbb{N}$ (which includes $n \geq 0$), so now fill in $c = 9/2$, $b = 0$, and complete the proof.

Let's use this approach to reprove $n^4 \notin O(3n^2)$.

Let $c \in \mathbb{R}^+$.
    Let $B \in \mathbb{N}$.
        Let $n =$ ____.
        ...
        So $n \in \mathbb{N}$.
        ...
        So $n \geq B$.
        ...
        So $n^4 > c3n^2$.
Thus $\forall c \in \mathbb{R}^+, \forall B \in \mathbb{N}, \exists n \in \mathbb{N}, n \geq B \wedge n^4 > c3n^2$.

Here's our chain of inequalities (the third $\cdots$):

$$\begin{aligned} \text{And } n^4 &\geq n^3 && \text{(don't need full power of } n^4\text{)} \\ &= n \cdot n^2 && \text{(make form as close as possible)} \\ &> c \cdot 3n^2 && \text{(if we make } n > 3c \text{ and } n > 0\text{).} \end{aligned}$$

Now pick $n = \max(B, \lceil 3c+1 \rceil)$.
The first $\cdots$ is:

> Since $c > 0$, $3c + 1 > 0$, so $\lceil 3c+1 \rceil \in \mathbb{N}$.
> Since $B \in \mathbb{N}$, $\max(B, \lceil 3c+1 \rceil) \in \mathbb{N}$.

The second $\cdots$ is:

> $\max(B, \lceil 3c+1 \rceil) \geq B$.

We also note just before the chain of inequalities:

> $n = \max(B, \lceil 3c+1 \rceil) \geq \lceil 3c+1 \rceil \geq 3c+1 > 3c$.

Some points to note are:

- Don't "solve" for n until you've made the form of the two sides as close as possible.
- You're not exactly solving for n: you are finding a condition of the form $n >$ __ that makes the desired inequality true. You might find yourself using the max function a lot.
- Be careful that you aren't "solving" for n in the wrong direction: the first time we reasoned that $n^4 > c3n^2 \Rightarrow n > \sqrt{3c}$, but the proof needs the reverse direction. Luckily, each of the steps were reversible (i.e., they were all equivalences), yielding the needed line of reasoning.

Other bounds

In analogy with $O(f)$, consider two other definitions:

Definition: For any function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$, let

$$\Omega(f) = \{f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow g(n) \geq cf(n)\}.$$

To say "$g \in \Omega(f)$" expresses the concept that "g grows at least as fast as f." (f is a lower bound on g).

Definition: For any function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$, let

$$\Theta(f) = \{g : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \exists c_1 \in \mathbb{R}^+, \exists c_2 \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 f(n) \leq g(n) \leq c_2 f(n)\}.$$

To say "$g \in \Theta(f)$" expresses the concept that "g grows at the same rate as f." (f is a tight bound for g, or f is both an upper bound and a lower bound on g).

Some theorems

Here are some general results that we now have the tools to prove.

- $f \in O(f)$.
- $(f \in O(g) \wedge g \in O(h)) \Rightarrow f \in O(h)$.
- $g \in \Omega(f) \Leftrightarrow f \in O(g)$.
- $g \in \Theta(f) \Leftrightarrow g \in O(f) \wedge g \in \Omega(f)$.

Test your intuition about Big-O by doing the "scratch work" to answer the following questions:

- Are there functions f, g such that $f \in O(g)$ and $g \in O(f)$ but $f \neq g$?[10]
- Are there functions f, g such that $f \notin O(g)$, and $g \notin O(f)$?[11]

To show that $(f \in O(g) \wedge g \in O(h)) \Rightarrow f \in O(h)$, we need to find a constant $c \in \mathbb{R}^+$ and a constant $B \in \mathbb{N}$, that satisfy:

$$\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq ch(n).$$

Since we have constants that scale h to g and then g to f, it seems clear that we need their product to scale g to f. And if we take the maximum of the two starting points, we can't go wrong. Making this precise:

Theorem 1: For any functions $f, g, h : \mathbb{N} \to \mathbb{R}^{\geq 0}$, we have $(f \in O(g) \wedge g \in O(h)) \Rightarrow f \in O(h)$.

Proof:

```
Assume $f \in O(g) \wedge g \in O(h)$.
    So $f \in O(g)$.
    So $g \in O(h)$.
    So $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n > B \Rightarrow f(n) \leq cg(n)$. (by defn. of $f \in O(g)$).
    Let $c_g \in \mathbb{R}^+, B_g \in \mathbb{N}$ be such that $\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq c_g g(n)$.
    So $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow g(n) \leq ch(n)$. (by defn. of $g \in O(h)$).
    Let $c_h \in \mathbb{R}^+, B_h \in \mathbb{N}$ be such that $\forall n \in \mathbb{N}, n \geq B_h \Rightarrow g(n) \leq c_h h(n)$.
    Let $c = c_g c_h$. Let $B = \max(B_g, B_h)$.
        Let $n \in \mathbb{N}$ be arbitrary.
        Assume $n \geq B$.
            Then $n \geq B_h$ (definition of max), so $g(n) \leq c_h h(n)$.
            Then $n \geq B_g$ (definition of max), so $f(n) \leq c_g g(n) \leq c_g c_h h(n)$.
            So $f(n) \leq ch(n)$.
        So $n \geq B \Rightarrow f(n) \leq ch(n)$.
        Since $n$ is an arbitrary natural number, $\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq ch(n)$.
    Since $c$ is a positive real number, and since $B$ is a natural number,
    $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq ch(n)$.
    So $f \in O(g)$, by definition.
So $(f \in O(g) \wedge g \in O(h)) \Rightarrow f \in O(h)$.
```

To show that $g \in \Omega(f) \Leftrightarrow f \in O(g)$, it is enough to note the the constant, c, for one direction is positive, so its reciprocal will work for the other direction.[12]

Theorem 2: For any functions $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$, we have $g \in \Omega(f) \Leftrightarrow f \in O(g)$.

Proof:

$g \in \Omega(f)$
$\Leftrightarrow$ (definition)
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow g(n) \geq cf(n)$
$\Leftrightarrow$ (by letting $c' = 1/c$ and $B' = B$).
$\exists c' \in \mathbb{R}^+, \exists B' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B' \Rightarrow f(n) \leq c'g(n)$
$\Leftrightarrow$ (definition)
$f \in O(g)$

To show $g \in \Theta(f) \Leftrightarrow g \in O(f) \wedge g \in \Omega(f)$, it's really just a matter of unwrapping the definitions.

Theorem 3: For any functions $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$, we have $g \in \Theta(f) \Leftrightarrow g \in O(f) \wedge g \in \Omega(f)$.

PROOF:

$g \in \Theta(f)$
$\Leftrightarrow$ (definition)
$\exists c_1 \in \mathbb{R}^+, \exists c_2 \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 f(n) \leq g(n) \leq c_2 f(n)$.
$\Leftrightarrow$ (combined inequality, and $B = \max(B_1, B_2)$).
$\exists c_1 \in \mathbb{R}^+, \exists B_1 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B_1 \Rightarrow g(n) \geq c_1 f(n) \wedge \exists c_2 \in \mathbb{R}^+, \exists B_2 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B_2 \Rightarrow g(n) \leq c_2 f(n)$
$\Leftrightarrow$ (definition)
$g \in \Omega(f) \wedge g \in O(f)$

Taxonomy of results

A LEMMA is a small result needed to prove something we really care about. A THEOREM is the main result that we care about (at the moment). A COROLLARY is an easy (or said to be easy) consequence of another result. A CONJECTURE is something suspected to be true, but not yet proven.

Here's an example of a conjecture whose proof has evaded the best minds for over 70 years. Maybe you'll prove it.

Define $f(n)$, for $n \in \mathbb{N}$ by:

$$f(n) = \begin{cases} n/2, & n \text{ even} \\ 3n+1, & n \text{ odd} \end{cases}$$

Let's define $f^2(n)$ as $f(f(n))$, and define $f^{k+1}(n)$ as $f(f^k(n))$. Here's the conjecture:

CONJECTURE: $\forall n \in \mathbb{N}, \exists k \in \mathbb{N}, f^k(n) = 1$.

Easy to state, but (so far) hard to prove or disprove.

Here's an example of a corollary that recycles some of the theorems we've already proven (so we don't have to do the grubby work). To show $g \in \Theta(f) \Leftrightarrow f \in \Theta(g)$, I re-use theorems proved above and the commutativity of $\wedge$:

COROLLARY: For any functions $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$, we have $g \in \Theta(f) \Leftrightarrow f \in \Theta(g)$.

PROOF:

$g \in \Theta(f)$
$\Leftrightarrow$ (by Theorem 3)
$g \in O(f) \wedge g \in \Omega(f)$.
$\Leftrightarrow$ (by Theorem 2)
$g \in O(f) \wedge f \in O(g)$
$\Leftrightarrow$ (by commutativity of $\wedge$)
$f \in O(g) \wedge g \in O(f)$
$\Leftrightarrow$ (by Theorem 2)
$f \in O(g) \wedge f \in \Omega(g)$
$\Leftrightarrow$ (by Theorem 3)
$f \in \Theta(g)$.

A very important note

Note that asymptotic notation (the Big-O, Big-Ω, and Big-Θ definitions) bound the asymptotic growth rates of *functions*, as n approaches infinity. Often in computer science we use this asymptotic notation to bound functions that express the running times of algorithms, perhaps in best case or in worst case. Asymptotic notation *does not* express or bound the worst case or best case running time, only the functions expressing these values.

This distinction is subtle, but crucial to understanding both running times and asymptotic notation. If this warning doesn't seem important to you now, come back and read this again in a few weeks, months, or courses. You'll thank me later.

Exercises

1. Prove or disprove the following claims:
   (a) $7n^3 + 11n^2 + n \in O(n^3)$ [13]
   (b) $n^2 + 165 \in \Omega(n^4)$
   (c) $n! \in O(n^n)$
   (d) $n \in O(n \log_2 n)$
   (e) $\forall k \in \mathbb{N}, k > 1 \Rightarrow \log_k n \in \Theta(\log_2 n)$

2. Define $g(n) = \begin{cases} n^3/165, & n < 165 \\ \left\lceil \sqrt{6n^5} \right\rceil, & n \geq 165 \end{cases}$. Note that $\forall x \in \mathbb{R}, x \leq \lceil x \rceil < x + 1$.

   Prove that $g \in O(n^{2.5})$.

3. Let $\mathcal{F}$ be the set of functions from $\mathbb{N}$ to $\mathbb{R}^{\geq 0}$. Prove or disprove the following claims:
   (a) $\forall f \in \mathcal{F}, \forall g \in \mathcal{F}, f \in O(g) \Rightarrow (f + g) \in \Theta(g)$
   (b) $\forall f \in \mathcal{F}, \forall f' \in \mathcal{F}, \forall g \in \mathcal{F}, (f \in O(g) \wedge f' \in O(g)) \Rightarrow (f + f') \in O(g)$

4. For each function f in the left column, choose one expression $O(g)$ from the right column such that $f \in O(g)$. Use each expression exactly once.

(i) $3 \cdot 2^n \in$ _______	(a) $O(\frac{1}{n})$
(ii) $\frac{2n^4+1}{n^3+2n-1} \in$ _______	(b) $O(1)$
(iii) $(n^5+7)(n^5-7) \in$ _______	(c) $O(\log_2 n)$
(iv) $\frac{n^4 - n\log_2 n}{n^2+1} \in$ _______	(d) $O(n)$
(v) $\frac{n \log_2 n}{n-5} \in$ _______	(e) $O(n \log_2 n)$
(vi) $8 + \frac{1}{n^2} \in$ _______	(f) $O(n^2)$
(vii) $2^{3n+1} \in$ _______	(g) $O(n^{10})$
(viii) $n! \in$ _______	(h) $O(2^n)$
(ix) $\frac{5\log_2(n+1)}{1+n\log_2 3n} \in$ _______	(i) $O(10^n)$
(x) $(n-2)\log_2(n^3+4) \in$ _______	(j) $O(n^n)$

Chapter 5 Notes

[1]From 0 to $(2-1)$, if we work in analogy with base 10.

[2]To parse the 0.101 part, calculate $0.101 = 1(2^{-1}) + 0(2^{-2}) + 1(2^{-3})$.

[3]You should be able to look up this algorithm in an elementary school textbook.

[4]Same as the previous exercise, but only write numbers that have 0's and 1's, and do binary addition.

[5]Integer divides by 10.

[6]Integer divide by 2.

[7]If only one of m, n are negative, ensure n is the negative one, perhaps by swapping. If both of m, n are negative, negate both of them and then call `mult`.

[8]This is the principle of mathematical induction, which you'll study in detail in CSC 236.

[9]Better in the sense of time complexity.

[10]Sure, $f = n^2$, $g = 3n^2 + 2$.

[11]Sure. f and g don't need to both be monotonic, so let $f(n) = n^2$ and

$$g(n) = \begin{cases} n, & n \text{ even} \\ n^3, & n \text{ odd} \end{cases}$$

So not every pair of functions from $\mathbb{N} \to \mathbb{R}^{\geq 0}$ can be compared using Big-O.

[12]Let's try the symmetrical presentation of bi-implication.

[13]The claim is true.
Let $c = 8$. Then $c \in \mathbb{R}^+$.
  Let $B = 12$. Then $B \in \mathbb{N}$.
    Let $n \in \mathbb{N}$. Assume $n \geq B$.
      Then $n^3 = n \times n^2 \geq 12 \times n^2 = 11 \times n^2 + n^2$. (since $n \geq B = 12$).
      So $n^2 \geq 12n$. (since $n \geq 12$, multiplying both sides by $n > 0$).
      So $12 > 1 \Rightarrow 12n > n$. (Multiplying both sides by $n > 0$).
      So $n^3 \geq 12n^2 = 11n^2 + n^2 \geq 11n^2 + 12n \geq 11n^2 + n$.
      So $7n^3 \geq 7n^3$.
      Thus $cn^3 = 8n^3 = 7n^3 + n^3 \geq 7n^3 + 11n^2 + n$. (adding the two inequalities).
    So $n \geq B \Rightarrow 7n^3 + 11n^2 + n \leq cn^3$.
    Since n is an arbitrary element of $\mathbb{N}$, $\forall n \in \mathbb{N}$, $n \geq B \Rightarrow 7n^3 + 11n^2 + n \leq cn^3$.
  Since B is a natural number, $\exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 7n^3 + 11n^2 + n \leq cn^3$.
Since c is a real positive number, $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 7n^3 + 11n^2 + n \leq cn^3$.
By definition, $7n^3 + 11n^2 + n \in O(n^3)$.