Algorithms — I

Asymptotic Notation

What is an asymptote? Asymptotic notation?

Big "oh" (Order of a f^n)

$f(n) = O(g(n)) \equiv f(n)$ is big oh of $g(n)$ iff $\exists c > 0$ and $\exists n_0 > 0$ such that $f(n) \leq c * g(n)$ for all $n \geq n_0$.

Examples:

$3n+2 = O(n)$ as $3n+2 \leq 4n$ for all $n \geq 2$.
$3n+3 = O(n)$ as $3n+3 \leq 4n$ for all $n \geq 3$.
$100n+6 = O(n)$ as $100n+6 \leq 101n$ for all $n \geq 6$.
$10n^2+4n+2 = O(n^2)$ as $10n^2+4n+2 \leq 11n^2$ for all $n \geq 5$
$1000n^2+100n-6 = O(n^2)$ as $1000n^2+100n-6 \leq 1001n^2$ for all $n \geq 100$
$6*2^n+n^2 = O(2^n)$ as $6*2^n+n^2 \leq 7*2^n$ for all $n \geq 4$.

Also, see
$3n+3 = O(n^2)$ as $3n+3 \leq 3n^2$ for all $n \geq 2$.
$10n^2+4n+2 = O(n^4)$ as $10n^2+4n+2 \leq 10n^4$ for all $n \geq 2$.

But
$3n+2 \neq O(1)$ as $3n+2 \not\leq c$ for any c and all $n \geq n_0$,
similarly, $10n^2+4n+2 \neq O(n)$.

$O(1) \rightarrow$ constant; $O(n) \rightarrow$ linear; $O(n^2) \rightarrow$ quadratic; $O(n^3) \rightarrow$ cubic
$O(2^n) \rightarrow$ exponential etc.

Also $10 = O(1)$, but we don't study "O" of constants.

For sufficiently large n, $O(\log n)$ is faster than $O(n)$;
$O(n\log n)$ is faster than $O(n^2)$ but slower than $O(n)$;
$O(n^2)$ is faster than $O(2^n)$ but slower than $O(n)$.

$f(n) = O(g(n))$ only provides an upper bound for $f(n)$ for all $n \geq n_0$.
But, it does not say how good the bound is.

See $n = O(2^n)$; $n = O(n^{2.5})$; $n = O(n^3)$ and so on.

To be useful, $g(n)$ must be as small as possible.

There is no such thing as $O(g(n)) = f(n)$.

Omega

$f(n) = \Omega(g(n))$ iff $\exists c > 0$ and $\exists n_0 > 0$ such that $f(n) \geq c \times g(n)$ for all $n \geq n_0$.

Examples:

$3n+2 = \Omega(n)$ as $3n+2 \geq 3n$ for all $n \geq 1$ (holds for $n \geq 0$ also, but we must find out $n_0 > 0$).

$3n+3 = \Omega(n)$ as $3n+3 \geq 3n$ for all $n \geq 1$.

$100n+6 = \Omega(n)$ as $100n+6 \geq 100n$ for all $n \geq 1$.

$10n^2+4n+2 = \Omega(n^2)$ as $10n^2+4n+2 \geq n^2$ for all $n \geq 1$.

$6*2^n+n^2 = \Omega(2^n)$ as $6*2^n+n^2 \geq 2^n$ for all $n \geq 1$.

Also, see

$3n+3 = \Omega(1)$

$10n^2+4n+2 = \Omega(n)$

$10n^2+4n+2 = \Omega(1)$

$6*2^n+n^2 = \Omega(n^{100})$

$6*2^n+n^2 = \Omega(n^{50.2})$

$6*2^n+n^2 = \Omega(n^2)$

- - - - - - - -

$f(n) = \Omega(g(n))$ only provides a lower bound for $f(n)$ for all $n \geq n_0$.

But it does not say how good the bound is.

To be useful, $g(n)$ must be as large as possible.

Theta

$f(n) = \Theta(g(n))$ iff $\exists c_1, c_2 > 0$ and $\exists n_0 > 0$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

Examples:

$3n+2 = \Theta(n)$ as $3n+2 \geq 3n$ for all $n \geq 2$ and $3n+2 \leq 4n$ for all $n \geq 2$, so $c_1 = 3$, $c_2 = 4$, $n_0 = 2$.

$3n+3 = \Theta(n)$

$10n^2+4n+2 = \Theta(n^2)$

$6*2^n+n^2 = \Theta(2^n)$

But $3n+2 \neq \Theta(1)$, $3n+3 \neq \Theta(n^2)$, $10n^2+4n+2 \neq \Theta(n)$, $6*2^n+n^2 \neq \Theta(n^{100})$.

$f(n) = \Theta(g(n))$ provides both lower and upper bounds for $f(n)$.

We usually don't write $3n+3 = O(3n)$ or $10n^2+4n+2 = \Omega(4n^2)$ or $6*2^n+n^2 = \Theta(4*2^n)$ although these are perfectly OK. The coefficients used in $g(n)$ are always 1.

I

So to say, $f(n) = O(g(n)) \Rightarrow kf(n) = O(g(n)), O(kg(n)) = O(g(n))$ etc.

Theorems

1. If $f(n) = a_m n^m + \cdots + a_1 n + a_0$ then $f(n) = O(n^m)$

   Proof: $f(n) \leq \sum_{i=0}^{m} |a_i| n^i$

   $\leq n^m \sum_{i=0}^{m} |a_i| n^{i-m}$

   $\leq n^m \sum_{i=0}^{m} |a_i|$ as $n^{i-m} \leq 1$

   so, $f(n) = O(n^m)$

2. If $f(n) = a_m n^m + \cdots + a_1 n + a_0$ and $a_m > 0$ then $f(n) = \Theta(n^m)$
3. If $f(n) = a_m n^m + \cdots + a_1 n + a_0$ and $a_m > 0$ then $f(n) = \Omega(n^m)$

Properties

1. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$

   If $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$ then $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$
2. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

   If $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$ then $f_1(n) + f_2(n) = \Omega(\min\{g_1(n), g_2(n)\})$
3. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n) f_2(n) = O(g_1(n) g_2(n))$

   If $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$ then $f_1(n) f_2(n) = \Omega(g_1(n) g_2(n))$.

Exercise:

1. draw a line from each of the three functions in the centre to the best big Ω value on the left and the best big O value on the right:

$\Omega(1/n)$		$O(1/n)$
$\Omega(1)$		$O(1)$
$\Omega(\log\log n)$		$O(\log\log n)$
$\Omega(\log^2 n)$		$O(\log^2 n)$
$\Omega(\sqrt[3]{n})$	$1/(\log n)$	$O(\sqrt[3]{n})$
$\Omega(n/\log n)$	$7n^5-3n+2$	$O(n/\log n)$
$\Omega(n)$	3^n	$O(n)$
$\Omega(n^{1.00001})$		$O(n^{1.00001})$
$\Omega(n^2/\log^2 n)$		$O(n^2/\log^2 n)$
$\Omega(n^2/\log n)$		$O(n^2/\log n)$
$\Omega(n^2)$		$O(n^2)$
$\Omega(n^3/2)$		$O(n^3/2)$
$\Omega(2^n)$		$O(2^n)$

2. For each of the following pairs of functions $f(n)$ and $g(n)$, either $f(n) = O(g(n))$ or $g(n) = O(f(n))$ but not both. Determine which is the case.

(i) $f(n) = (n^2-n)/2$, $g(n) = 6n$

(ii) $f(n) = n+2\sqrt{n}$, $g(n) = n^2$

(iii) $f(n) = n+\log n$, $g(n) = n\sqrt{n}$

(iv) $f(n) = n^2+3n+4$, $g(n) = n^3$

(v) $f(n) = n\log n$, $g(n) = n\sqrt{n}/2$

(vi) $f(n) = n+\log n$, $g(n) = \sqrt{n}$

I

3. True or False?

(i) $n^2 = O(n^3)$ (ii) $n^3 = O(n^2)$ (iii) $2n^2+1 = O(n)$

(iv) $n\log n = O(n\sqrt{n})$ (v) $\sqrt{n} = O(\log n)$ (vi) $\log n = O(\sqrt{n})$

4. For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$ or none of the other: *

5. Prove that $\lceil \log n \rceil = O(n)$

By looking at $\lceil \log n \rceil$ for small values of $n \geq 1$;

$n=1$, $\lceil \log n \rceil \leq n$

$n=2$, $\lceil \log n \rceil \leq n$

$n=3$ $\lceil \log n \rceil \leq n$

The proof is by induction on n.

Suppose that $\lceil \log(n-1) \rceil \leq n-1$

Then $\lceil \log n \rceil \leq \lceil \log(n-1) \rceil + 1$

$\leq (n-1)+1$

$= n$

Hence, $c=1$, and $n_0 = 1$.

6. Prove that $n^2 - 3n - 18 = \Omega(n)$

7. Prove that $n^3 - 3n^2 - n + 1 = \Theta(n^3)$

8. Prove that $n = O(2^n)$

9. Prove that $(n+1)^2 = O(n^2)$

We need to find a constant c such that $(n+1)^2 \leq cn^2$,

i.e, $n^2 + 2n + 1 \leq cn^2$

or $(c-1)n^2 - 2n - 1 \geq 0$

Take $c=4$.

The roots of $3n^2 - 2n - 1$ are $(2 \pm \sqrt{4+12}/2) = \{-1/3, 1\}$.

The second root gives us n_0.

$\therefore$ For all $n \geq 1$, $(n+1)^2 \leq 4n^2$, so $(n+1)^2 = O(n^2)$

10. Prove that $3n\lfloor \log n \rfloor = O(n^2)$

* $f(n) = n^2 + 3n + 4$, $g(n) = 6n + 7$;

$f(n) = \sqrt{n}$, $g(n) = \log(n+3)$;

$f(n) = n\sqrt{n}$, $g(n) = n^2 - n$.

Correctness of Algorithms

1. Prove that the following algorithm for the addition of natural numbers is correct.

```
float fn. add(y,z) { // Return y+z, where y,z ∈ N
        x=0; c=0; d=1;
        while (y>0)∨(z>0)∨(c>0) {
            a=y mod 2; b= z mod 2;
            if a⊕b⊕c then x=x+d;
            c = (a∧b)∨(b∧c)∨(c∧a);
            d =2d; y=⌊y/2⌋;
            z= ⌊z/2⌋;
        }
        return x;
}
```

- We see $2\lfloor n/2 \rfloor + (n \bmod 2) = n$ for all $n \in N$
- We claim that if $y, z \in N$ then add (y,z) returns the value $y+z$
- For each of the identifiers, use a subscript i to denote the value of the identifier after the i^{th} iteration of the while loop, for $i \geq 0$, with $i=0$ meaning the time immediately before the while loop is entered and immediately after the while statement.
- By inspection

$$a_{j+1} = y_j \bmod 2$$

$$b_{j+1} = z_j \bmod 2$$

$$y_{j+1} = \lfloor y_j/2 \rfloor$$

$$z_{j+1} = \lfloor z_j/2 \rfloor$$

$$d_{j+1} = 2d_j$$

- c_{j+1} is 1 iff at least two of a_{j+1}, b_{j+1} and c_j are 1

$\therefore \; c_{j+1} = \lfloor (a_{j+1} + b_{j+1} + c_j)/2 \rfloor$

- Also, d is added into x only when an odd number of a, b and c are 1.

$\therefore \; x_{j+1} = x_j + d_j((a_{j+1} + b_{j+1} + c_j) \bmod 2)$ — I

- We now prove $(y_j + z_j + c_j)d_j + x_j = y_0 + z_0$.

  It is called the loop invariant.
- The proof is by induction on j
- When $j=0$, its trivial since $c_0=0$, $d_0=1$ and $x_0=0$
- Now assuming $(y_j+z_j+c_j)d_j + x_j = y_0+z_0$,

  we see

$$
\begin{aligned}
&(y_{j+1}+z_{j+1}+c_{j+1})d_{j+1}+x_{j+1}\\
&= (\lfloor y_j/2\rfloor + \lfloor z_j/2\rfloor + \lfloor (y_j \bmod 2 + z_j \bmod 2 + c_j)/2\rfloor) 2d_j\\
&\quad + x_j + d_j((y_j \bmod 2 + z_j \bmod 2 + c_j) \bmod 2)\\
&= (\lfloor y_j/2\rfloor + \lfloor z_j/2\rfloor) 2d_j + x_j + d_j(y_j \bmod 2 + z_j \bmod 2 + c_j)\\
&= (y_j+z_j+c_j)d_j + x_j\\
&= y_0+z_0, \text{ an invariant!}
\end{aligned}
$$

- We now use it to show that the algorithm is correct. For that, we need to prove that the algorithm terminates with x containing the sum of y and z.
- By inspection, the values of y and z are ~~hal~~ halved (rounding, if they are odd) on every iteration of the loop. Therefore, they will eventually both be zero and stay that way. At the first point at which $y=z=0$, either c will equal zero or c will be assigned zero on the next iteration of the loop. Thus, eventually $y=z=c=0$ at which point the loop terminates.

  Now, we prove that x has the correct value on termination. Suppose, the loop terminates after t iterations, $t \geq 0$.
- By the loop invariant $(y_t+z_t+c_t)d_t+x_t = y_0+z_0$

  Since, $y_t=z_t=c_t=0$, we see $x_t = y_0+z_0$
- Thus the algorithm terminates with x containing the sum of the initial values of y and z, as required.

I

2. Prove that the following algorithm for swapping two numbers is correct.

```
void fn swap(x,y) { // Swap x and y
        x = x + y;
        y = x - y;
        x = x - y;
}
```

3. Computing x^n where x is a real number and $n \geq 0$ an integer:

First algorithm:

power $= x$;
for $i = 1$ to $n-1$ power $= x *$ power;

Second algorithm:

If n is an integral power of 2, i.e $n = 2^k$ for some integer k.

power $= x$;
for $i = 1$ to k power $=$ power^2

Third algorithm:

When n is not an integral power of 2.

Let $n = b_k b_{k-1} \cdots b_1 b_0$ in binary

So, $n = \sum_{q=0}^{k} b_q 2^q$

Then, $x^n = x^{\sum_{q=0}^{k} b_q 2^q}$

$= x^{b_0} * (x^2)^{b_1} * (x^4)^{b_2} * \cdots * (x^{2^k})^{b_k}$

For example x^{101}.

So, $n = 101 = 1100101$

Then $x^{101} = x^1 * (x^2)^0 * (x^4)^1 * (x^8)^0 * (x^{16})^0 * (x^{32})^1 * (x^{64})^1$

Also observe that $b_0 = n \bmod 2$ and

$\lfloor n/2 \rfloor = b_k b_{k-1} \cdots b_1$ in binary

Hence the algorithm

```
float power(float x, int n) {
    float product, psequence;
    product = 1;
    psequence = x;
    while (n) {
        if ((n%2) == 1) product = product * psequence;
        n = n/2;
        psequence = psequence * psequence;
    }
    return product;
}
```

Use the loop invariant

$ps_j \, x_j^{n_j} = x_0^{n_0}$

** $ps =$ product

I

Analysis of Algorithms

```
function m(n) {
    r = 0; - - - - - - - - - - - 1
    for i = 1 to n-1 - - - - - - 2
        for j = i+1 to n - - - - 3
            for k = 1 to j - - - 4
                r = r+1; - - - - 5
    return r; - - - - - - - - - 6
}
```

The for-loop on lines 4-5 has the same effect as $r = r + j$.

$\therefore$ the for-loop on lines 3-5 has the following effect:

for $j = i+1$ to n
    $r = r + j$,

equivalently,
$$r = r + \sum_{j=i+1}^{n} j.$$

Now,
$$\sum_{j=i+1}^{n} j = \sum_{j=1}^{n} j - \sum_{j=1}^{i} j = \frac{n(n+1)}{2} - \frac{i(i+1)}{2}$$

$\therefore$ The for-loop on lines 2-5 has the following effect:

for $i = 1$ to $n-1$
    $r = r + \frac{n(n+1)}{2} - \frac{i(i+1)}{2}$

equivalently,
$$r = r + \sum_{i=1}^{n-1}\left(\frac{n(n+1)}{2} - \frac{i(i+1)}{2}\right)$$

$$\sum_{i=1}^{n-1}\left(\frac{n(n+1)}{2} - \frac{i(i+1)}{2}\right) = \frac{n(n-1)(n+1)}{2} - \frac{1}{2}\sum_{i=1}^{n-1} i^2 - \frac{1}{2}\sum_{i=1}^{n-1} i$$

$$= \frac{(n-1)n(n+1)}{2} - \frac{n(n-1)(2n-1)}{12} - \frac{n(n-1)}{4} = n(n^2-1)/3.$$

The for-loop that begins on line 2 is executed for $O(n)$ iterations. The for-loop that begins on line 3 is executed for $O(n)$ iterations. The for-loop that begins on line 4 is executed for $O(n)$ iterations. Line 1 and 5 take $O(1)$ time.

Therefore, the function $m(n)$ runs in time $O(n^3)$.

Is this also $\Omega(n^3)$?

Is this also $\Theta(n^3)$?

Analyze the addition algorithm of natural numbers

```
float fn add(y,z) {// Return y+z where y,z ∈ N
    x=0; e=0; d=1;
    while (y>0) ∨ (z>0) ∨ (c>0) {
        a = y mod 2; b = z mod 2;
        if a ⊕ b ⊕ c then x = x+d;
        c = (a∧b) ∨ (b∧c) ∨ (c∧a);
        d = 2d; y = ⌊y/2⌋;
        z = ⌊z/2⌋;
    }
    return x;
}
```

Find the exact number of additions in

if $a \oplus b \oplus c$ then $x = x + d$

and then put a big-O around it.

Analyze the iterative algorithm for Fibonacci numbers

```
function fib(n) { // Return the nth Fibonacci number
    if (n == 0) return 0
    else {
        last = 0; current = 1;
        for i = 2 to n {
            temp = last + current;
            last = current;
            current = temp;
        }
    }
}
```

'am

p?

I