Analysing Algorithms (time complexity)

Arash Rafiey

August 30, 2016

Usually interested in **running time** (but sometimes also memory requirements).

Example: One of the simplest sorting algorithms

Input : n numbers in array $A[1], \ldots, A[n]$

1. for $(i = 1; i < n; i++)\{$
2. $\quad element = A[i]$; int $index = i$;
3. $\quad$ for $(j = i + 1; j <= n; j++)$
4. $\quad\quad$ if $(A[j] < element)$ $\{$
5. $\quad\quad\quad element = A[j]$; $index = j$;
6. $\quad\quad\}$
7. $\quad$ SWAP$(A[i], A[index])$;
8. $\}$

Example: $n = 6,\ A = [14, 13, 12, 15, 16, 11]$

i	element	index	"new" $A[]$
1	11	5	$[11, 13, 12, 15, 16, 14]$
2	12	2	$[11, 12, 13, 15, 16, 14]$
3	13	2	$[11, 12, 13, 15, 16, 14]$
4	14	5	$[11, 12, 13, 14, 16, 15]$
5	15	5	$[11, 12, 13, 14, 15, 16]$

Simple sort example

Clearly depending on n (loops depend on n)

– Body of outer loop (over i) is executed $n-1$ times

– Each increment takes constant time, c_1

– Line 2 takes constant time, c_2

– Body of inner loop (over j) is executed $n-i$ times

– Again, each increment takes time c_1

– Suppose comparison in line 4 takes constant time, c_3

– If condition is true, then another c_2, otherwise 0

– Thus lines 4–6 take at most $c_3 + c_2$

– Swap in line 7 takes constant time, c_4

Input : n numbers in array $A[1], \ldots, A[n]$

```
1. for (i = 1; i < n; i++ ){
2.     element = A[i]; index = i;
3.     for (j = i+1; j <= n; j++)
4.       if (A[j] < element) {
5.       element = A[j]; index = j;
6.       }
7.     SWAP(A[i], A[index]);
8. }
```

Putting everything together: (the worst case)

$$\sum_{i=1}^{n-1} \left[c_1 + c_2 + \left(\sum_{j=i+1}^{n} c_1 + c_3 + c_2 \right) + c_4 \right]$$

Putting everything together: (the worst case)

$$\sum_{i=1}^{n-1} \left[c_1 + c_2 + \left(\sum_{j=i+1}^{n} c_1 + c_3 + c_2 \right) + c_4 \right]$$

Let's simplify this expression a bit:
Let $d_1 = c_1 + c_2 + c_4$
Let $d_2 = c_1 + c_2 + c_3$
Note: d_1 and d_2 are constants

Putting everything together: (the worst case)

$$\sum_{i=1}^{n-1}\left[c_1 + c_2 + \left(\sum_{j=i+1}^{n} c_1 + c_3 + c_2\right) + c_4\right]$$

Let's simplify this expression a bit:
Let $d_1 = c_1 + c_2 + c_4$
Let $d_2 = c_1 + c_2 + c_3$
Note: d_1 and d_2 are constants
Now we have

$$\sum_{i=1}^{n-1}\left[d_1 + \left(\sum_{j=i+1}^{n} d_2\right)\right]$$

Note that $\sum_{j=i+1}^{n} d_2 = (n-i)\cdot d_2 = n\cdot d_2 - i\cdot d_2$

$$\sum_{i=1}^{n-1} (d_1 + n \cdot d_2 - i \cdot d_2)$$

Terms d_1 and $n \cdot d_2$ do not depend on i, so this is equal to

$$(n-1) \cdot d_1 + (n-1) \cdot n \cdot d_2 - d_2 \cdot \sum_{i=1}^{n-1} i$$

We know that

$$\sum_{i=1}^{k} i = 1 + 2 + 3 + \cdots + k = \frac{k \cdot (k+1)}{2}$$

Thus the expression becomes

$$
\begin{aligned}
& (n-1) \cdot d_1 + (n-1) \cdot n \cdot d_2 - d_2 \cdot \frac{(n-1) \cdot n}{2} \\
= \quad & (n-1) \cdot d_1 + d_2 \cdot \frac{(n-1) \cdot n}{2} \\
= \quad & n \cdot d_1 - d_1 + n^2 \cdot d_2/2 - n \cdot d_2/2 \\
= \quad & n^2 \cdot d_2/2 + n \cdot (d_1 - d_2/2) - d_1
\end{aligned}
$$

Thus the expression becomes

$$\begin{aligned} & (n-1)\cdot d_1 + (n-1)\cdot n\cdot d_2 - d_2\cdot\frac{(n-1)\cdot n}{2} \\ = \;& (n-1)\cdot d_1 + d_2\cdot\frac{(n-1)\cdot n}{2} \\ = \;& n\cdot d_1 - d_1 + n^2\cdot d_2/2 - n\cdot d_2/2 \\ = \;& n^2\cdot d_2/2 + n\cdot(d_1 - d_2/2) - d_1 \end{aligned}$$

With $e_1 = d_2/2$ and $e_2 = d_1 - d_2/2$ (note: e_1 and e_2 are constants) we obtain

$$e_1\cdot n^2 + e_2\cdot n - d_1$$

Since e_1, e_2, and d_1 are constants, the running time depends **quadratically** on n.
This is the idea behind **asymptotic analysis**: we don't care about constants (either multiplicative or additive), or about lower-order terms.

Theta-notation

For a given function $g(n)$, $\Theta(g(n))$ denotes the set

$$\Theta(g(n)) = \{f(n) : \quad \text{there exist positive constants } c_1, c_2, n_0 \text{ such that } c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ for all } n \geq n_0\}$$

Intuition: $f(n)$ belongs to the family $\Theta(g(n))$ if $\exists$ constants c_1, c_2 s.t. $f(n)$ can fit between $c_1 \cdot g(n)$ and $c_2 \cdot g(n)$, for all n sufficiently large.
Correct notation: $f(n) \in \Theta(g(n))$
Usually used: $f(n) = \Theta(g(n))$.
We also say that "$f(n)$ is in $\Theta(g(n))$".

Examples of Θ-notation:

$f(n) = 2n^2 = \Theta(n^2)$

because with $g(n) = n^2$ and $c_1 = 1$ and $c_2 = 2$ we have

$0 \leq c_1 g(n) \leq f(n) = 2 \cdot n^2 \leq c_2 \cdot g(n)$.

Examples of Θ-notation:

$f(n) = 2n^2 = \Theta(n^2)$
because with $g(n) = n^2$ and $c_1 = 1$ and $c_2 = 2$ we have
$0 \leq c_1 g(n) \leq f(n) = 2 \cdot n^2 \leq c_2 \cdot g(n)$.

$f(n) = 8n^5 + 17n^4 - 25 = \Theta(n^5)$
because $f(n) \geq 7 \cdot n^5$ for n large enough

n	$8n^5 + 17n^4 - 25$	n^5	$7n^5$
1	$8 \cdot 1 + 17 \cdot 1 - 25 = 0$	1	7
2	$8 \cdot 32 + 17 \cdot 16 - 25 = 503$	32	224

and $f(n) \leq 8n^5 + 17n^5 = 25n^5$, thus $c_1 = 7$, $c_2 = 25$ and $n_0 = 2$ are good enough.

More intuition:

for all $n \geq n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor.

We say that $g(n)$ is an **asymptotically tight bound** for $f(n)$.

More intuition:

for all $n \geq n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor.

We say that $g(n)$ is an **asymptotically tight bound** for $f(n)$.

Back to sorting example

We had running time $T(n) = e_1 \cdot n^2 + e_2 \cdot n - d_1$. Now we can say: $T(n) = \Theta(n^2)$,

we have formal means to "get rid" of lower-order terms and constant coefficients.

Why is this true?

Must find positive c_1, c_2, n_0 such that

$$c_1 n^2 \leq e_1 n^2 + e_2 n - d_1 \leq c_2 n^2$$

for all $n \geq n_0$.
Dividing by n^2 gets us

$$c_1 \leq e_1 + \frac{e_2}{n} - \frac{d_1}{n^2} \leq c_2$$

Suppose e_1, e_2, d_1 are positive (other cases similar).
Obviously, for $n \geq e_2$ we have $e_2/n \leq 1$ and thus $e_1 + e_2/n - d_1/n^2 \leq e_1 + 1$ and thus $c_2 = e_1 + 1$ and $n_0 = e_2$ does the job for the right-hand inequality.
Also, for $n^2 \geq d_1 \Leftrightarrow n \geq \sqrt{d_1}$ we have $d_1/n^2 \leq 1$ and thus $e_1 + e_2/n - d_1/n^2 \geq e_1 - 1$ and therefore $c_1 = e_1 - 1$ is sufficient.
With $n_0 = \max\{e_2, \sqrt{d_1}\}$ both conditions are fulfilled simultaneously.

Theta notation respects the main term

However, $n^3 \neq \Theta(n^2)$
Recall: for $n^3 = \Theta(n^2)$ we would have to find constants c_1, c_2, n_0 with

$$0 \leq c_1 n^2 \leq n^3 \leq c_2 n^2$$

for $n \geq n_0$.
Intuition: there's a factor of n between both functions, thus we **cannot** find a constant c_2!
Suppose, for purpose of contradiction, that there **are** constants c_2 and n_0 with $n^3 \leq c_2 \cdot n^2$ for $n \geq n_0$.
Dividing by n^2 yields $n \leq c_2$, which cannot possibly hold for arbitrarily large n (c_2 must be a constant).

Big-O-notation

When we're interested in **asymptotic upper bounds** only, we use O-notation (read: "big-O").
For given function $g(n)$, define $O(g(n))$ (read: "big-O of g of n" or also "order g of n") as follows:

$$O(g(n)) = \{f(n) : \begin{array}{l}\text{there exist positive constants} \\ c, n_0 \text{ such that} \\ f(n) \leq c \cdot g(n) \\ \text{for all } n \geq n_0\}\end{array}$$

We write $f(n) = O(g(n))$ to indicate that $f(n)$ is member of set $O(g(n))$.
Obviously, $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$; we just drop the left inequality in the definition of $\Theta(g(n))$.

Big-Omega-notation

Like O-notation, but for lower bounds
For a given function $g(n)$, $\Omega(n)$ denotes the set

$$\Omega(g(n)) = \{f(n) : \quad \text{there exist positive constants } c, n_0 \text{ such that } c \cdot g(n) \leq f(n) \text{ for all } n \geq n_0\}$$

Saying $T(n) = \Omega(n^2)$ means growth of $T(n)$ is at least the of n^2.
Clearly, $f(n) = \Theta(g(n))$ iff $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$.

o-notation

Similar to O

$f(n) = O(g(n))$ means we can upper-bound the growth of f by the growth of g (up to a constant factor)

$f(n) = o(g(n))$ is the same, **except** we require the growth of f to be **strictly** smaller than the growth of g:

For a given function $g(n)$, $o(n)$ denotes the set

$$o(g(n)) = \{f(n) : \quad \text{for } \textbf{any} \text{ pos constant } c \text{ there exists a pos constant } n_0 \text{ such that } c \cdot f(n) < g(n) \text{ for all } n \geq n_0\}$$

Intuition: $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

In other words, f is o(something) if there is **no** constant factor between f and something.

Examples:

$n = o(n^2)$

$\log n = o(n)$

$n = o(2^n)$

$n^{1,000} = o(1.0001^n)$

$1 = o(\log n)$

Example

Show that $n = o(n^2)$.

Example

Show that $n = o(n^2)$.
We need to show that for every $c > 0$, $\exists$ an n_0 such that for every $n \geq n_0$ we have $cn < n^2$.

This means $c < n$ and hence if we set $n_0 = \lceil c \rceil$ we have $cn < n^2$ for every $n \geq n_0$.

Example

Show that $n = o(n^2)$.
We need to show that for every $c > 0$, $\exists$ an n_0 such that for every $n \geq n_0$ we have $cn < n^2$.

This means $c < n$ and hence if we set $n_0 = \lceil c \rceil$ we have $cn < n^2$ for every $n \geq n_0$.

Show that $\log n = o(n)$.

Example

Show that $n = o(n^2)$.
We need to show that for every $c > 0$, $\exists$ an n_0 such that for every $n \geq n_0$ we have $cn < n^2$.

This means $c < n$ and hence if we set $n_0 = \lceil c \rceil$ we have $cn < n^2$ for every $n \geq n_0$.

Show that $\log n = o(n)$.
We need to show that for every $c > 0$, $\exists$ an n_0 such that for every $n \geq n_0$ we have $c \log n < n$.
Set $n = 2^m$. Now we need for find an m such that $cm < 2^m$ for every $m > m_0$.
We note that $2m \leq 2^m$ for every positive integer $m > 0$. Now if we set $m = 2\lceil c \rceil$ then we have $c < 2^{\lceil c \rceil}$ and also $2\lceil c \rceil \leq 2^{\lceil c \rceil}$.
Therefore $c(2\lceil c \rceil) < 2^{2\lceil c \rceil}$ and hence we set $n_0 = 2^{2\lceil c \rceil}$.

omega-notation

ω is to Ω what o is to O:
$f(n) = \omega(g(n))$ iff $g(n) = o(f(n))$
For a given function $g(n)$, $\omega(n)$ denotes the set

$$\omega(g(n)) = \{f(n) : \quad \text{for } \textbf{any} \text{ pos constant } c \text{ there exists a pos constant } n_0 \text{ such that } c \cdot g(n) < f(n) \text{ for all } n \geq n_0\}$$

In other words:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

if the limit exists.
I.e., $f(n)$ becomes **arbitrarily** large relative to $g(n)$.

Exercises

1) Arrange the following functions in ascending order

$f_1(n) = n^{2.5}$

$f_2(n) = \sqrt{2n}$

$f_3(n) = n + 10$

$f_4(n) = 10^n$

$f_5(n) = 100^n$

$f_6(n) = n^2 \log n.$

$$f_2(n) = \sqrt{2n}$$
$$f_3(n) = n + 10$$
$$f_6(n) = n^2 \log n$$
$$f_1(n) = n^{2.5}$$
$$f_4(n) = 10^n$$
$$f_5(n) = 100^n$$

2) Arrange the following functions in ascending order

$g_1(n) = 2^{\sqrt{\log n}}$

$g_2(n) = 2^n$

$g_3(n) = n^{\frac{4}{3}}$

$g_4(n) = n(logn)^3$

$g_5(n) = n^{logn}$

$g_6(n) = 2^{2^n}$

$g_7(n) = 2^{n^2}$