Algorithms

DATE : 03/03/2022

We talked last time about "polynomial time" algorithm.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

WALL-CLOCK RUNTIMES WITH A PROCESSOR PERFORMING 1 MILLION OPERATORS / s

Def: An algorithm has a (worst-case) runtime of $f(n)$ if the algorithm never makes more than $f(n)$ operations when run on inputs of size n.

So: How can we determine the (worst-case) runtime for an algorithm?
How can we say that an algorithm has a runtime $\leq f(n)$?

Given the varying nature of computational devices (and of their CPUs), it's very hard to count exactly the number of a given algorithm, or of a given piece of code.

```
DEF INC(A):
    RETURN A+1

INC(10
```

ALL OF THIS CAN BE DONE WITH C OPERATIONS

- "10" will be pushed onto the stack
- INC will be executed
- INC will add to its namespace a variable of name "A"
- The value "10" will be popped out of the stack, and will be assigned to A.
- You have to take the value of A, sum 1 to that value, and you have to return the result.

no way to tell how many operations the CPU does

It's easy to observe that there exists a constant $c > 1$ such that, that piece of code can be run with c elementary operations. It's hard to determine c.

ELEMENTARY OPERATIONS ⟶ CPU runs in one clock/s

UNPRECISE DEFINITION which might be true for older CPUs

The real definition won't be defined because there are many factors to take in consideration

NOTE: if we throw away constants, we won't be able to distinguish which algorithm is faster.

Suppose that, somehow, we determine that - in this case (code above) - $c = 27$.

```
DEF F(n):
  x=0
  FOR i in range(n):
     x= INC(x)
  return x
```

```
DEF INC(A):
  RETURN A+1
```

} 27 OPER.

$$\frac{n\,27 + c'n + c''}{n(27 + c') + c''}$$

RUNTIME $f(n)$

we have to throw away const.

```
y=0
FOR i IN RANGE(n)
   y+= f(i)
```

⟶ $n(n(27 + c') + c'') + c'''$

O, Ω, Θ - ASYMPTOTIC RUNTIMES

To avoid getting stuck into endlessly many constants, we just disregard constants.

For an algorithm A, let $T(n)$ be its runtime (worst-case) on inputs of size n.

What we aim to do is to represent a runtime $T(n) = 1{,}35n^2 + 0{,}16n + 250$ with it's most representative term, n^2

DEF : IF $T(n)$ and $f(n)$ are non-negative, increasing, functions, we say that "$T(n)$" is $O(f(n))$

IF $\exists\ c, n_0 > 0$, such that $\forall n \geq n_0$, $T(n) \leq c\, f(n)$.

$T(n) = n$ $\qquad$ $f(n) = n^3 - n^2$

$n > 1$

$T(1) = 1$ $\qquad$ $f(1) = 0$

$T(n) = O(f(n))$

$T(n) \leq O(f(n))$

$T(n) \in O(f(n))$

Suppose that an algorithm takes time $T(n) = pn^2 + qn + r$, $p, q, r \geq 0$ and $p \neq 0$

Then $\forall n \geq n_0 \triangleq 1$, it holds that $qn \leq qn^2 = n(qn)$

it $\Rightarrow$ $r \leq rn^2$.

Thus, $T(n) = pn^2 + qn + r \leq pn^2 + qn^2 + rn^2 = (p+q+r)n^2$

If we take $c = p+q+r$, we then have that $f(n) = n^2$

$T(n) \leq c \cdot f(n)$ $\quad \forall n \geq n_0$

$T(n) = O(f(n)) = O(n^2)$

$T(n) = pn^2 + qn + r$, $p, q, r \geq 0$, $p \neq 0$

Is it the case that $T(n) = O(n^3)$?

$T(n) \leq (p+q+r)n^2 \leq (p+q+r)n^3$ $\qquad$ $(\forall n \geq n_0 = 1)$

$c = p+q+r$

$T(n) = O(n^3)$

How can we prove that $O(n^2)$ is tighter to $T(n)$ than $O(n^3)$?

DEF: IF $T(n)$ and $f(n)$ are non-negative, increasing, functions, we say that "$T(n)$ is $\Omega(f(n))$" IF $\exists\ c, n_0 > 0$ such that $\forall n \geq n_0$, $T_n \geq c \cdot f(n)$.

$T(n) = pn^2 + qn + r \qquad p, q, r \geq 0 \text{ AND } p \neq 0$

$qn \geq 0 \cdot n = 0$

$r \geq 0$

$T(n) \geq p \cdot n^2 \qquad c = p \qquad n_0 = 0$

$T(n) \geq cf(n) \qquad f(n) = n^2$

THUS $T(n) = \Omega(f(n)) = \Omega(n^2)$

$T(n) \geq \Omega(n^2)$

If $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$, we write that $T(n) = \Theta(f(n))$.

$f(n) = \Theta(T(n))$

$T(n) \geq \Omega(f(n))$
$T(n) \leq O(T(n))$

Since, $T(n) = pn^2 + qn + r$

$T(n) \leq O(n^2)$, $T(n) \leq O(n^3)$

$T(n) \geq \Omega(n^2)$

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

TRANSITIVITY

L: IF $f = O(g)$ and $g = O(h)$, then $f = O(h)$
IF $f = \Omega(g)$ and $g = \Omega(h)$, " $f = \Omega(h)$.

P: $f = O(g)$ implies that $\exists\ n_0, c > 0$ s.t. $f(n) \leq c \cdot g(n)$ $\forall n \geq n_0$

$g = O(h)$ " " $\exists\ n_0', c' > 0$ s.t. $g(n) \leq c' h(n)$ $\forall n \geq n_0'$

Then, $\forall\ n \geq \max(n_0, n_0')$ IT HOLDS THAT :

$f(n) \leq c\,g(n)$ and $g(n) \leq c'h(n)$

$f(n) \leq cg(n) \leq c(c'h(n)) = (c \cdot c') \cdot h(n)$

THUS, FOR $c'' = c \cdot c'$ and for $n_0'' = \max(n_0, n_0')$
IT HOLDS THAT $f(n) \leq c'' \cdot h(n) \quad \forall n \geq n_0''$

THUS $f(n) = O(h(n))$

The Ω case is symmetric, so we skip it $\square$

C: If $f(n) = \Theta(g(n))$ AND $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$

SUM OF FUNCTIONS

L: if $f = O(h)$ and $g = O(h)$, then $f + g = O(h)$.

P: $\exists\ n_0, n_0', c, c'$ S.T. $f(n) \leq c \cdot h(n) \quad \forall n \geq n_0$ AND
$g(n) \leq c \cdot h(n) \quad \forall n \geq n_0'$

BUT THEN, $f(n) + g(n) \leq c \cdot h(n) + c' h(n) = (c + c') h(n)$ IF
$n \geq \max(n_0, n_0')$

WITH $c'' = c + c'$ and $n_0'' = \max(n_0, n_0')$, we get
$f + g = O(h)$.

$f(n) = n \log_2 n$

$f(n) + f(n) = O(f(n)) = O(n \cdot \log_2 n)$

$g = f = h = f$

$g(n) = f(n) + f(n)$ $\quad f(n) + f(n) + f(n) = O(n \cdot \log n)$

$$\sum_{i=1}^{\sqrt{n}} f(n) = \sqrt{n} \cdot f(n) = O(n^{3/2} \log n)$$

L: Let $K \geq 1$ BE A CONSTANT.

IF $f_1 = O(h)$, $f_2 = O(h)$, ..., $f_K = O(h)$ then $f_1 + f_2 + \dots + f_K = O(h)$

$O(100 \cdot n) = O(n)$

Let $T(n) = a_0 + a_1 n + a_2 n^2 + \dots + a_d n^d$, for some $d \geq 1$, with $a_d \neq 0$

$$T(n) = 3 + n^2 + 15n^3$$

we say that $T(n)$ is a polynomial of degree d.

L: IF $f(n)$ is a polynomial of degree "d" with non-negative coefficients, then $f(n) = O(n)$

P: $f(n) = \sum_{i=0}^{d} (a_i n^i)$. IF we let $f_i(n) = a_i n^i$, we have that $f(n) = \sum_{i=0}^{d} f_i(n)$.

But, then, $f_i(n) \leq |f_i(n)| = |a_i| n^i$

$$f(n) = \sum_{i=0}^{d} \leq \sum_{i=0}^{d} (|a_i| n^i) = \sum_{i=0}^{d} O(n^i)$$

(since d is a constant) $\leq \sum_{i=0}^{d} O(n) - O(n)$

L: For each $b > 1$, and for each $x > 0$, it holds $\log_b n = O(n^x)$

$$\log n \leq O(n^{0.001})$$

$$\log_a n = \frac{\log_b n}{\log_b a} \Rightarrow \log_a n = \left(\frac{1}{\log_b a}\right) \log_b (n)$$

$$\log_a n = \Theta(\log_b n)$$

L: For each $r > 1$, and for each $x > 0$ it holds $n^x = O(r^n)$

$$n^{100} \leq O(1{,}01^n)$$

$2^n = O(3^n)$ $\qquad$ $3^n \neq O(2^n)$

a^n vs b^n $\qquad$ $a = 2$ $b = 3$

$a = c \cdot b$ $\qquad$ $c = \frac{2}{3}$

$a^n = (c \cdot b)^n = c^n \cdot b^n$

IF $c > 1 \Rightarrow c^n \to \infty$

IF $c < 1 \Rightarrow c^n \to 0$

IF $c > 1$ $a^n = O(b^n)$

IF $c < 1$ $a^n = \Omega(b^n)$