O-notation

D.E ZEGOUR

École Supérieure d'Informatique

ESI

O-notation

Introduction

The execution time of a program depends on the following factors:
- Program data
- Quality of the code generated by the compiler
- Machine characteristics (speed and nature of instructions)
- Algorithm complexity

$T(n) = c\, f(n)$
n : number of data
c : constant grouping the mentioned factors
f(n) : growth rate of T(n)

Example : $T(n) = c\, n^2$
T(n) could be the number of executed instructions.

In practice, average time is often challenging to determine.

Try to find the average case, otherwise find the worst-case scenario.

O-notation

Definition

It will be said that $T(n)$ est $O(f(n))$ if there exist $c > 0$ and $n0 > 0$ such that $T(n) \leq c\, f(n)$ for all $n \geq n0$.

A program with an execution time of $O(f(n))$ is a program that has $f(n)$ as its growth rate.

It can also be said that $f(n)$ is an upper bound on the growth rate of $T(n)$.

The constant depends on factors related to the machine and the code.

O-notation

Example

$T(n) = O(n^2)$, this means
There exist a constant $c > 0$ and a constant $n0 > 0$ such that for $n > n0$
$T(n) \leq c\, n^2$

Let's suppose that $T(0) = 1$, $T(1) = 4$ and in general case
$T(n) = (n + 1)^2$
Let's prove that $T(n)$ is $O(n^2)$

Therefore, one must find constants $c > 0$ and $n0 > 0$ such that for any $n > 0$, we have
$T(n) <= c\, n^2$,
which means $(n + 1)^2 <= cn^2$

O-notation

Example

$(n + 1)^2 \leq cn^2$

$n^2 + 2n + 1 <= c\ n^2$
$(c-1)\ n^2 - 2n - 1 \geq 0$

Therefore for c = 4 and n0 = 1 we have $T(n) \leq cn^2$ $T(n) = O(n^2)$
(or $(c-1)n^2 - 2n - 1 \geq 0$

Delta = 4c

For c = 4, Two roots 1 and -1/3

O-notation

Ω-notation

To specify the lower bound of the growth rate of $T(n)$, we will use the notation $\Omega(g(n))$, which means: there exists a constant $c > 0$ such that:

$T(n) \geq c\, g(n)$ for an infinite number of values of n.

Example : $T(n) = n^3 + 2n^2$ is $\Omega(n3)$ because for $c = 1$ $T(n) \geq n^3$ for $n = 0, 1, 2, ...$

O-notation

Operations : Sum rule

If $T1(n) = O(f(n))$ and
$T2(n) = O(g(n))$ are the execution times of two program fragments, P1 and P2,
then the execution time of P1 followed by P2 is
$T1(n) + T2(n) = O(Max(f(n), g(n)))$

Proof:

$T1(n) = O(f(n))$ ==> There exist $c1 > 0$ and $n1 > 0$ such that for any $n > n1$: $T1(n) \leq C1\ f(n)$

$T2(n) = O(g(n))$ ==> There exist $c2 > 0$ and $n2 > 0$ such that for any $n > n2$: $T2(n) \leq C2\ g(n)$

$$T1(n) + T2(n) \leq c1\ f(n) + c2\ g(n) \quad \text{for } n0 = \max(n1, n2)$$
$$\leq (c1 + c2) \max(f(n), g(n))$$

Therefore, there exist $c = c1 + c2$ and $n0 = \max(n1,n2)$

Consequence : If $g(n) \leq f(n)$ for any $n > n0$, then $(f(n) + g(n)) = O(f(n))$
Example : $O(n^2 + n) = O(n^2)$

O-notation

Operations : Sum rule

-It is used to calculate the execution time of a sequence of steps in a program.

P1 $O(n^2)$

P2 $O(n^3)$

P3 $O(n\log_2(n))$

$O(Max(n^2, n^3)) = O(n^3)$

$O(Max(n^3, nLog_2(n)) = O(n^3)$

O-notation

Operations : Product rule

If $T1(n) = O(f(n))$ and
$T2(n) = O(g(n))$,
then $T1(n)\ T2(n) = O(f(n)\ g(n))$

Proof

$T1(n) = O(f(n))$ ==> There exist $c1 > 0$ and $n1 > 0$ such that for any $n > n1$: $T1(n) \le c1\ f(n)$

$T2(n) = O(g(n))$ ==> There exist $c2 > 0$ and $n2 > 0$ such that for any $n > n2$: $T2(n) \le c2\ g(n)$

$T1(n) * T2(n) \le c1 * c2\ f(n)\ g(n)$ for $n > n0$ with $n0 = \max(n1, n2)$

Therefore, there exist $c = c1 * c2$ and $n0 = \max(n1, n2)$ such that $T1(n) * T2(n) \le c\ f(n)\ g(n)$.

Or $T1(n)*T2(n)=O(f(n)g(n))$.

Consequence : $O(\ c\ f(n)\ ) = O(\ f(n)\ )$ if $c > 0$.
Example : $O(n^2/2) = O(n^2)$

O-notation

Measuring iterative algorithms

1. Assignment, reading, or writing: O(1)

2. Sequence of steps: rule of sum.
Therefore, the time of the sequence is determined by the step with the longest execution time.

3. Alternative: consider the worst-case scenario.

4. Loop: Rule of product.
It is the product of the number of iterations of the loop by the longest possible time for the execution of the body.

O-notation

Measuring iterative algorithms : Example

```
(1)   FOR I := 1 , N - 1
(2)       FOR J := N , I + 1 , - 1
(3)           IF ELEMENT ( A [ J - 1 ] ) > ELEMENT ( A [ J ] )
(4)               Temp := ELEMENT ( A [ J - 1 ] ) ;
(5)               ASS_ELEMENT ( A [ J - 1 ] , ELEMENT ( A [ J ] ) ) ;
(6)               ASS_ELEMENT ( A [ J ] , Temp ) ;
              ENDIF
          ENDFOR
      ENDFOR
```

Bubble sort of an array A[1..n]

(4), (5), and (6) each take O(1).

Rule of sum: (4), (5), and (6) is O(Max(1, 1, 1)) = O(1).

For the IF statement, O(Max(1, 1)) = O(1)

Loop (2) to (6): body: O(1); loop: O(n-i)
Rule of product: O((n-i) O(1) = O(n-i) .

Loop (1) to (6): body: O(n-i) or O(n) (previous result);
loop: O(n-1) or O(n)
Rule of product: O(n) O(n-i) = O(n^2)