Complejidad Computacional

Tiempo de ejecución

El tiempo de ejecución de un algoritmo con "n" datos será T(n), donde se tomará en cuenta el peor caso, esto es, el tiempo máximo de ejecución que tomaría en ejecutarse con cualesquiera n datos.

Ejercicio

Calcula el T(n) del siguiente pseudocódigo, suponiendo que cada operación toma una unidad de tiempo:

```
for ( i = 0; i < n; i++ )
  if ( a[ i ] > a [ i + 1 ] )
     temp = a[ i ]
     a[ i ] = a[ i + 1]
     a[ i + 1 ] = temp
```

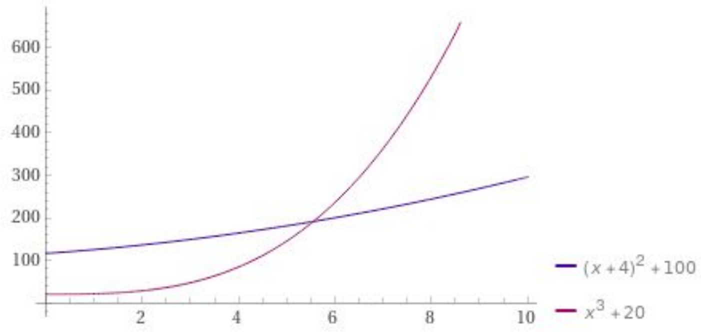
Notación O grandota (Big-Oh)

Sea $T(n)$ el tiempo de ejecución de un algoritmo y sea $f(n) : \mathbb{R}^+ \to \mathbb{R}^+$ entonces:

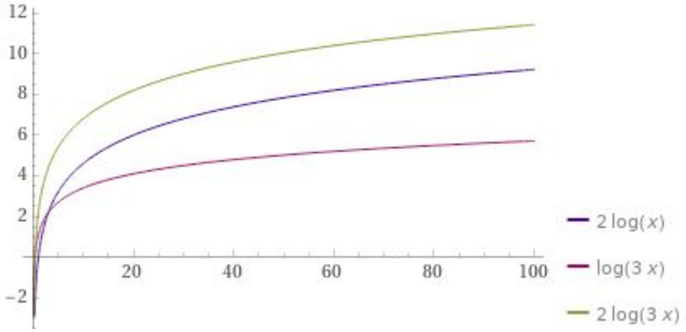
$$T(n) \in O(f(n)) \Leftrightarrow \exists_{c,s>0} \forall_{n \geq s}\ T(n) \leq cf(n)$$

¿Cómo se lee?

En otras palabras, si a partir de un punto inicial ***s*** la función T(n) es menor que f(n) multiplicado por una constante ***c***, entonces T(n) ϵ O(f(n))



$(x+4)^2+100$ ϵ **O(**$x^3 + 20$**)**



2log(x) ϵ **O(**log(3x)**)**

Ejemplo

Demostrar que $T(n) = (n+1)^3 \in O(n^3)$

$(n+1)^3 = n^3 + 3n^2 + 3n + 1$

Para $n \geq 1$ se cumple que

$3n^2 \leq 3n^3$, $3n \leq 3n^3$ y $1 \leq n^3$

Por lo que

$(n+1)^3 = n^3 + 3n^2 + 3n + 1 \leq n^3 + 3n^3 + 3n^3 + n^3 = 8n^3$

Así que para $n \geq 1$ tenemos que $(n+1)^3 \leq 8n^3$

Con $s = 1$ y $c = 8$ se cumple que $\forall_{n \geq s}\ T(n) \leq cn^3$

$\therefore T(n) \in O(n^3)$

Algunas propiedades

- Sea $k > 0$, $T(n) \epsilon O(f(n)) \Leftrightarrow T(n) \epsilon O(k \cdot f(n))$
- $T(n) \epsilon O(f(n)) \Leftrightarrow T(n) \epsilon O(f(n) + k)$
- $T(n) \epsilon O(f(n))$ y $f(n) \epsilon O(g(n)) \Rightarrow T(n) \epsilon O(g(n))$
- $T_1(n) \epsilon O(f_1(n))$ y $T_2(n) \epsilon O(f_2(n)) \Rightarrow T_1(n) + T_2(n) \epsilon O(max(f_1(n), f_2(n)))$
- $T_1(n) \epsilon O(f_1(n))$ y $T_2(n) \epsilon O(f_2(n)) \Rightarrow T_1(n) \cdot T_2(n) \epsilon O(f_1(n) \cdot f_2(n))$

Ejercicio

Demostrar que $T(n) \in O(kf(n)) \Leftrightarrow T(n) \in O(f(n))$, con $k>0$

Demostración de $\Rightarrow$

$T(n) \in O(kf(n)) \Rightarrow \exists_{s,c} \forall_{n \geq s} T(n) \leq ckf(n)$

Sea $c' = ck$

$\Rightarrow \exists_{s,c'} \forall_{n \geq s} T(n) \leq c'f(n)$

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

$\therefore T(n) \in O(kf(n)) \Rightarrow T(n) \in O(f(n))$

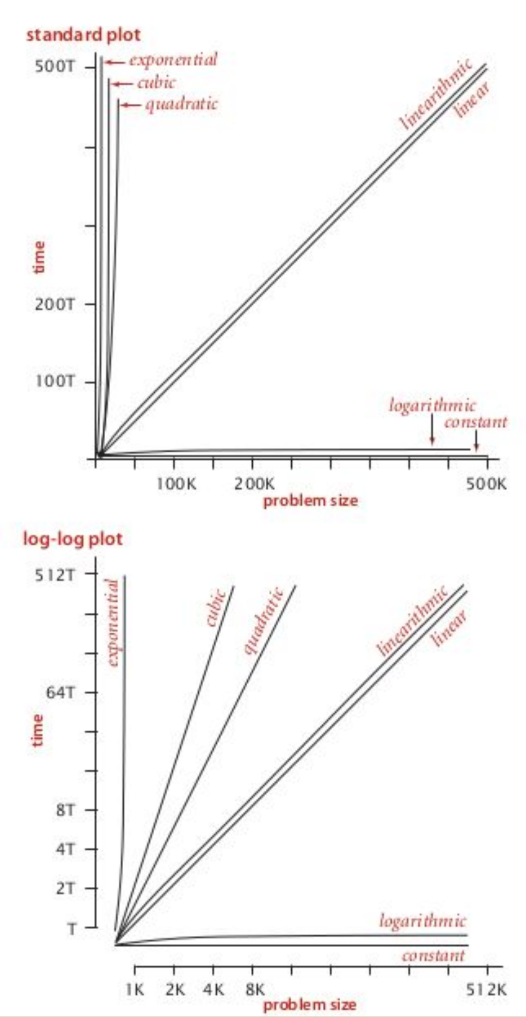
Demostrar que $T(n) \in O(f(n)) \Rightarrow T(n) \in O(kf(n))$ se deja como ejercicio al lector.

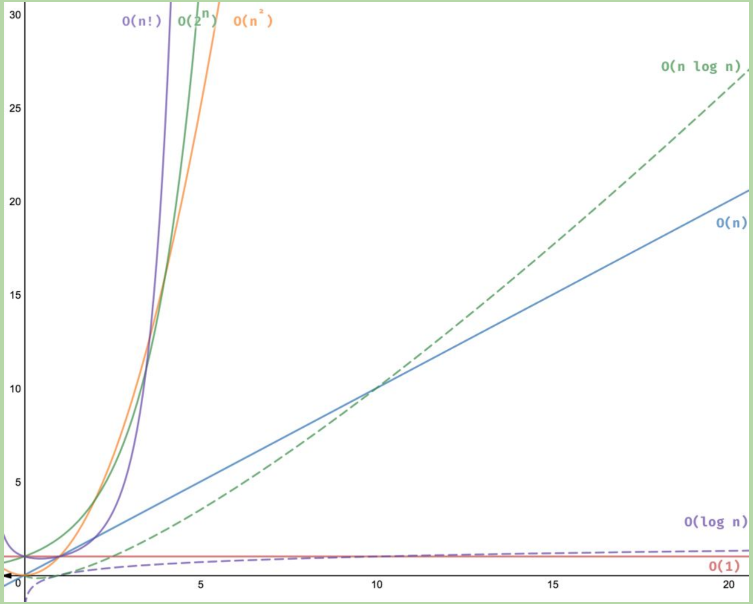
Ordenes de crecimiento más comunes

La eficiencia disminuye en cada renglón

Notación O grandota	Descripción
$O(1)$	constante
$O(\log n)$	logarítmico
$O(n)$	lineal
$O(n * \log n)$	n * log n
$O(n^2)$	cuadrático
$O(n^3)$	cúbico
$O(2^n)$	exponencial

EDD


standard plot
500T
exponential
cubic
quadratic
linearithmic
linear
time
200T
100T
logarithmic
constant
100K
200K
500K
problem size
log-log plot
512T
exponential
cubic
quadratic
linearithmic
linear
64T
time
8T
4T
2T
T
logarithmic
constant
1K
2K
4K
8K
512K
problem size


O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)
30
25
20
15
10
5
0
5
10
15
20

¿Y ahora qué?

Ahora podemos comparar algoritmos respecto a la complejidad de su ejecución.

Ejemplo: Comparemos dos formas de sumar los primeros ‘n’ naturales

```
suma = 0
for ( i = 0; i < n; i++ )
    suma += i
return suma
```

VS

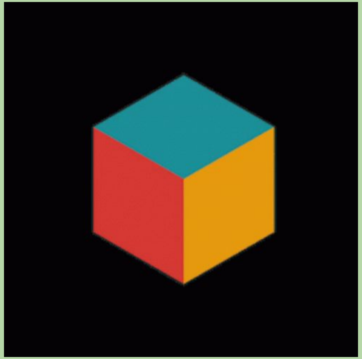
```
suma = n * (n + 1) / 2
return suma
```

Pregunta

Se tienen las funciones $f(n) = 1$, $g(n) = n$ y $h(n) = 2^n$, ¿Cómo se relacionan repecto a la notación O grandota?

Espacio de ejecución

El espacio de ejecución de un algoritmo con "n" datos será S(n), donde se toma en cuenta el peor caso, esto es, la cantidad máxima de espacio en memoria que tomaría al ejecutarse con cualesquiera n datos.



Se puede usar también la notación O grandota y se aplican las mismas propiedades.