>>> IF013 - Fundamentos Teóricos de Informática
>>> Licenciatura de Sistemas - UNPSJB - Sede Trelew

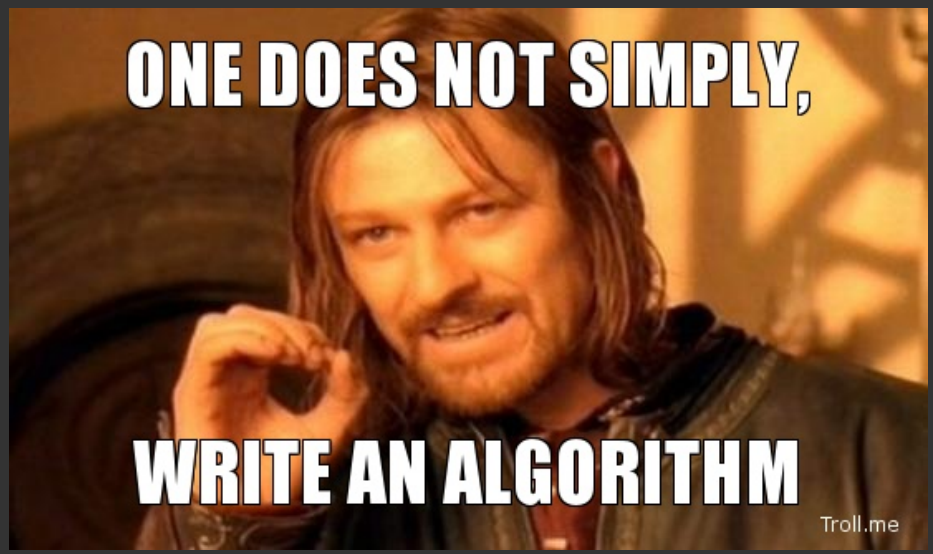
Name: Celia Cintas[†], Pablo Navarro[‡], Samuel Almonacid[§]
Date: December 11, 2017

[†]cintas@cenpat-conicet.gob.ar, cintas.celia@gmail.com, @RTFMCelia
[‡]pnavarro@cenpat-conicet.gob.ar, pablo1n7@gmail.com
[§]almonacid@cenpat-conicet.gob.ar, almonacid.samuel.tw@gmail.com

>>> Análisis de Algoritmos

¿en función de qué elegirías un programa?: ¿de su elegancia?, ¿de la legibilidad?, ¿de su usabilidad?, ¿de su velocidad de ejecución?, ¿de la memoria que consume? Nosotros consideraremos aquí criterios basados en la eficiencia.

* El coste o complejidad espacial, la cantidad de memoria que consume.
* El coste o complejidad temporal, la cantidad de tiempo que necesita para resolver un problema.



>>> Aproximación Empírica a Análisis de Algoritmos

Dado un valor n, retornar la suma de los valores entre 0 y n.

Solución iterativa:

```
def first_sum(n):
    final_sum = 0
    for x in range(n + 1):
        final_sum += x

    return final_sum
```

Basado en $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$:

```
def second_sum(n):
    return (n * (n + 1))/2
```

```
assert(first_sum(1000) == second_sum(1000))
```

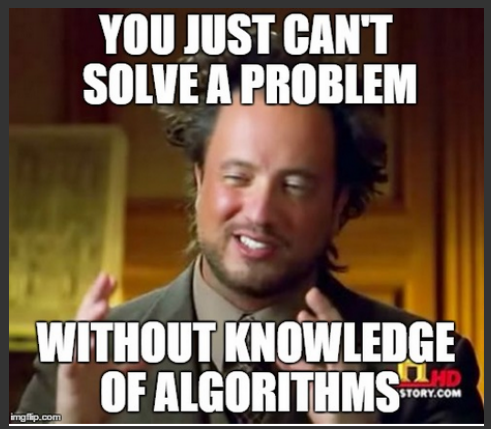
```
%timeit first_sum(1000)
best of 3: 45 µs per loop

%timeit second_sum(1000)
best of 3: 170 ns per loop
```

>>> Aproximación Empírica a Análisis de Algoritmos (Cont.)

Algunos problemas ...

* Dependiente del lenguaje, difícil análisis relativo entre distintos lenguajes de programación.
* La implementación del *benchmarking* depende del lenguaje elegido y por cada implementación debe agregarse código de medidas de tiempo.
* Difícil de analizar cuando tenemos variación de n.
* Dependiente del *hardware* y SO utilizado para realizar corridas.

¿Podemos prescindir de estos experimentos y seguir obteniendo resultados que nos permitan comparar algoritmos? ¿es posible estimar el tiempo de ejecución sin necesidad de implementar y ejecutar los programas?



>>> Abstracción del costo de operaciones elementales.

No nos interesa que nuestros estudios dependan del coste concreto de cada operación elemental. Bastará con que sepamos cuántas operaciones elementales ejecuta un programa y cómo depende ese número del n del problema a resolver.

Definición

Un paso es un segmento de código cuyo tiempo de proceso no depende del n del problema considerado y está acotado por alguna constante.

* operaciones aritméticas.
* operaciones lógicas.
* comparación entre escalares.
* acceso a variables escalares.
* acceso a un elemento de un arreglo.
* asignaciones.
* lectura/escritura de escalar.

Ejemplo

```
def sumatorio(n):
  s = 0 # 1 paso
  for i in range(n): # n veces
    s = s + i  # 1 paso
  return s # 1 paso
```

$$T(n) = 1n + 1 + 1 = 1n + 2$$

```
int sumatorio(int n)
{
  int s, i;
  s = 0; // 1 paso
  for (i=1; i<=n; i++)// 2n + 2 veces
    s = s + i; // 1 paso
 return s; // 1 paso
}
```

$$T(n) = 2n + 2 + 1n + 1 + 1 = 3n + 4$$

Cómo evoluciona el coste de un algoritmo con el n de entrada, más allá del lenguaje de programación o de los detalles de implementación. ¿Es el costo constante?, ¿crece linealmente con n? ¿o crece con el cuadrado de n?.

>>> Complejidad temporal: Análisis Asintótico

* Mejor Caso.
* Peor Caso.
* Caso Promedio.

Existen varias aproximaciones para el Análisis Asintótico:

* Cota superior asintótica ($O()$).
* Cota inferior asintótica ($\Omega()$).
* Cota ajustada asintótica ($\mathrm{Z}()$).

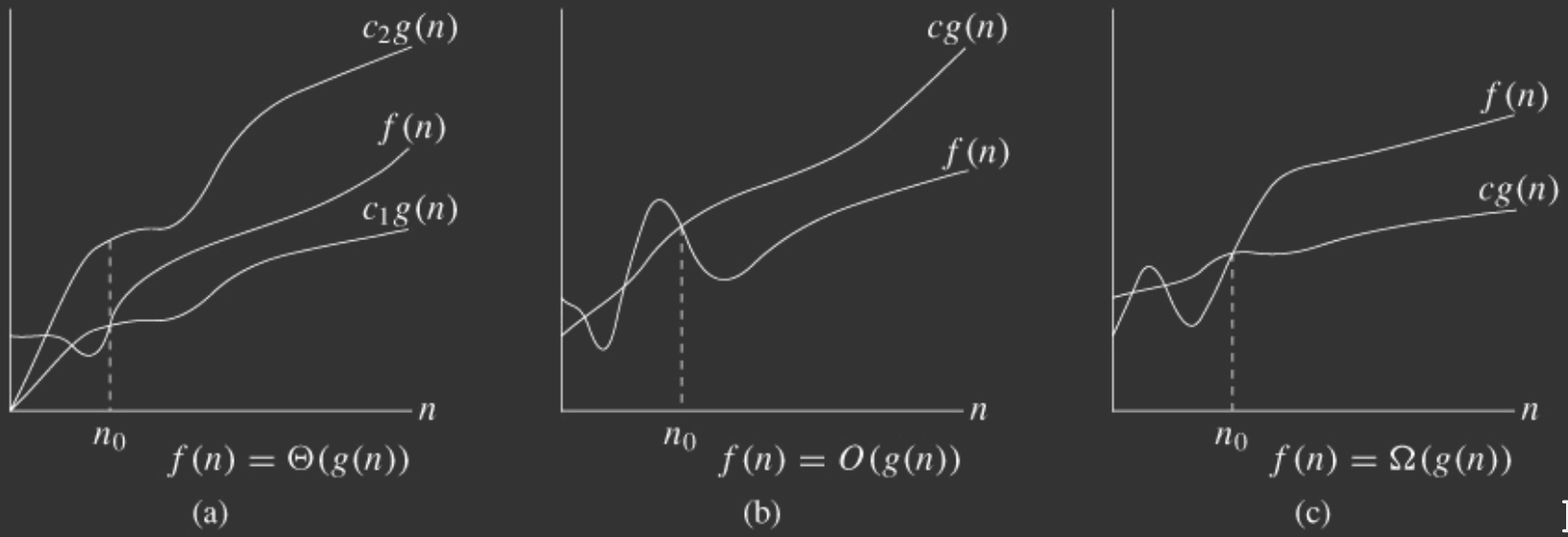


Figura de http://web.engr.oregonstate.edu/~huanlian/teaching/cs570/

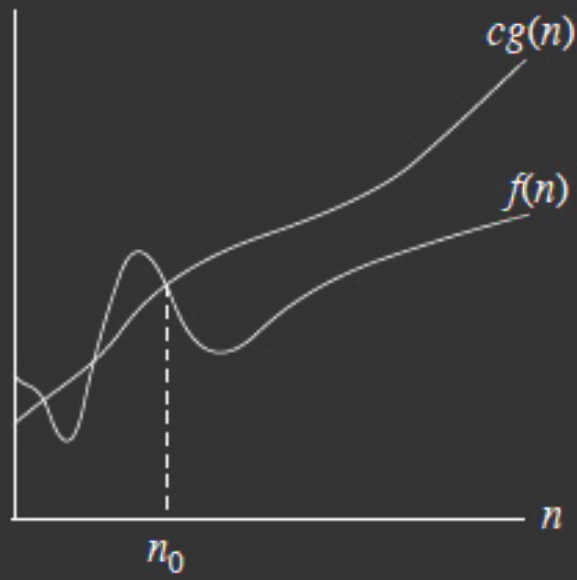
>>> Notación $O()$

Sean f y g dos funciones definidas sobre $\mathbb{R}$:

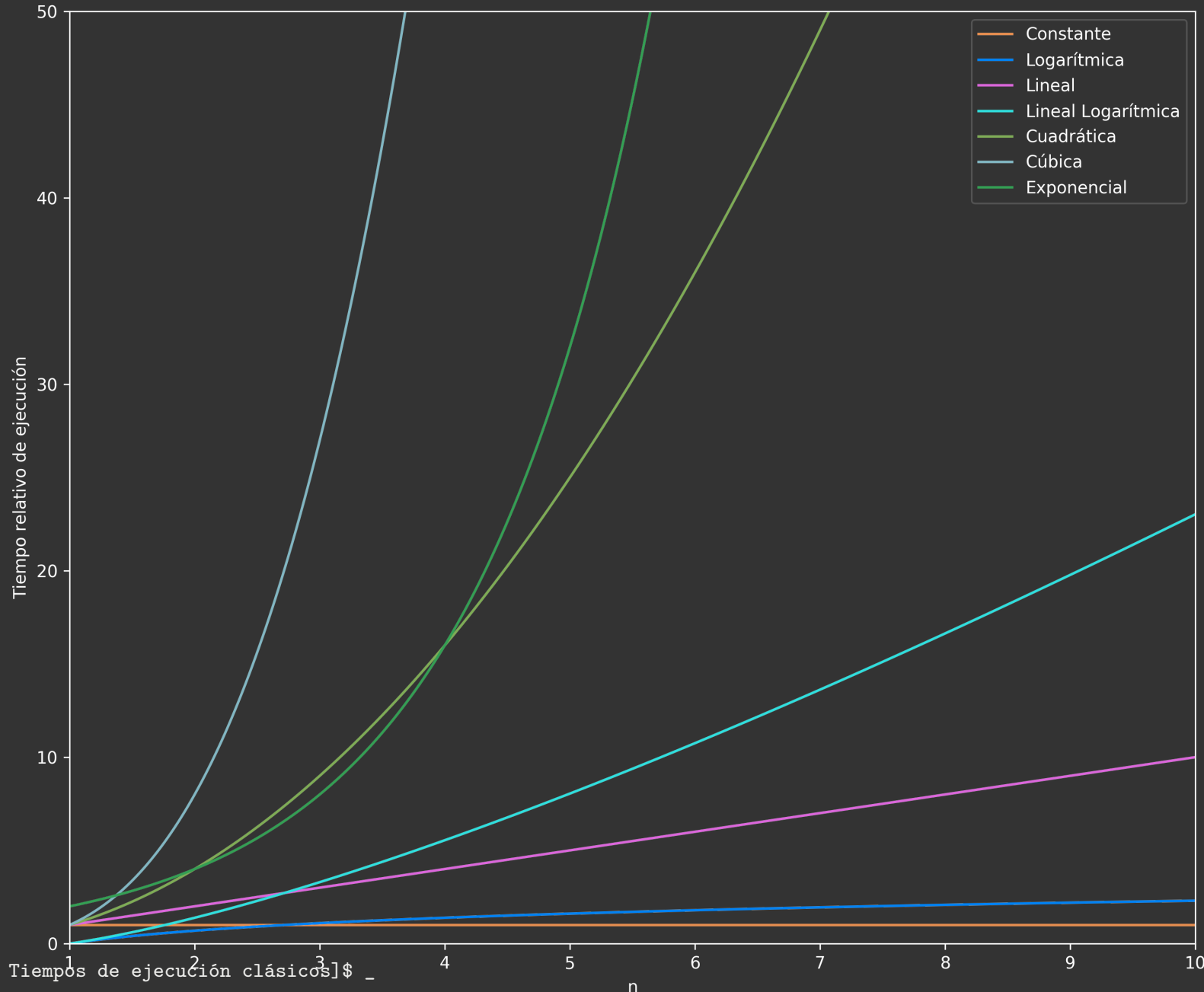
$$f(n) = O(g(n)) \text{ con } n \to \infty \tag{1}$$

Si solo si existe un valor $c > 0$ para una gran cantidad de valores de n y un valor n_0 talque:

$$|f(n)| \leq c|g(n)| \text{ para todo } n \geq n_0 \tag{2}$$



>>> Tiempos de ejecución clásicos O()



$O(1)$
$O(\log n)$
$O(n)$
$O(n \log n)$
$O(n^2)$
$O(n^3)$
$O(c^n)$

>>> Tiempos de ejecución clásicos $O()$ (Cont.)

Coste	$n = 1$	$n = 5$	$n = 10$	$n = 50$	$n = 100$
Constante	1 μs	1 μs	1 μs	1 μs	1 μs
Logarítmico	1 μs	1.7 μs	2 μs	2.7 μs	3 μs
Lineal	1 μs	5 μs	10 μs	50 μs	100 μs
$n \log n$	1 μs	4.5 μs	11 μs	86 μs	201 μs
Cuadrático	1 μs	25 μs	100 μs	2.5 ms	10 ms
Cúbico	1 μs	125 μs	1 ms	125 ms	1s
Exponencial (2^n)	1 μs	32 μs	1 ms	1 año y 2 meses	40 millones de eones

Coste	$n = 1000$	$n = 10000$	$n = 100000$
Constante	1 μs	1 μs	1 μs
Logarítmico	4 μs	5 μs	6 μs
Lineal	1 ms	10 ms	100 ms
$n \log n$	3 ms	40 ms	500 ms
Cuadrático	1 s	100 s	16 minutos y medio
Cúbico	16 minutos y medio	1 día y medio	casi 32 años

Tabla de *Andrés Marzal e Isabel Gracia, Lenguajes y Sistemas, Universitat Jaume I.*

>>> Tiempos de ejecución clásicos $O()$ (Cont.)

* Orden Constante: $O(1)$

```
def func_o_const(lst):
    print(lst[0])

func_o_const([1, 2, 3, 4])
```

* Orden Lineal: $O(n)$

```
def print_list(lst):
    for val in lst:
        print(val)

print_list([1, 2, 3])
```

* Orden Cuadrático: $O(n^2)$

```
def func_quad(lst):
    for item_1 in lst:
        for item_2 in lst:
            print(item_1, item_2)

func_quad([0, 1, 2, 3])
```

* Orden Logarítmico: $O(\log n)$

Ejemplos de algoritmos divide y vencerás.

>>> Ejemplo: Notación O() (Cont.)

```
a=5
b=6
c=10
for i in range(n):
    for j in range(n):
        x = i * i
        y = j * j
        z = i * j
for k in range(n):
    w = a*k + 45
    v = b*b
d = 33
```

>>> Notación O() (Cont.)

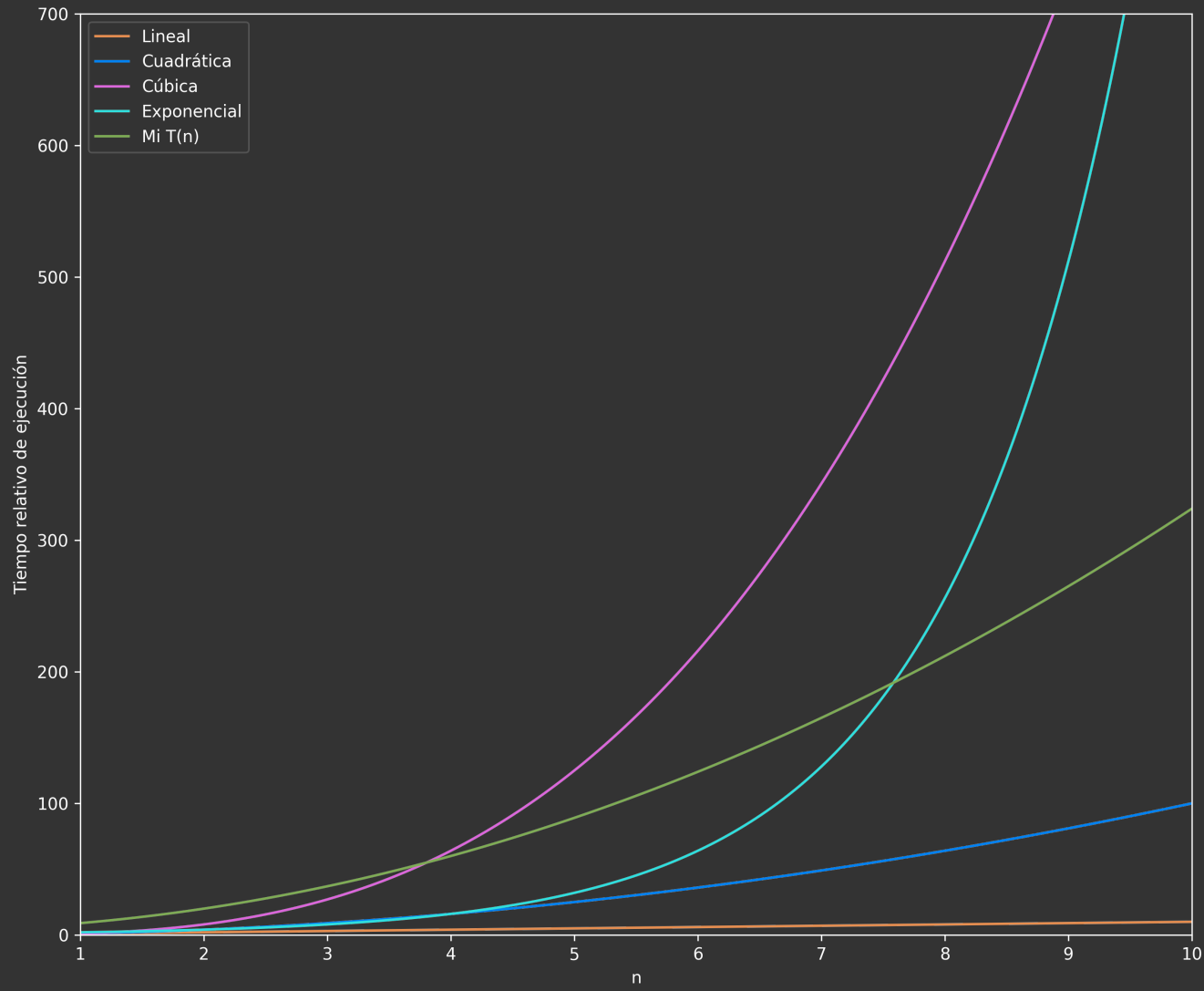
```
a=5
b=6
c=10
for i in range(n):
    for j in range(n):
        x = i * i
        y = j * j
        z = i * j
for k in range(n):
    w = a*k + 45
    v = b*b
d = 33
```

$$T(n) = 3 + 3n^2 + 2n + 1 = 3n^2 + 2n + 4$$

>>> Notación O() (Cont.)

$$T(n) = 3 + 3n^2 + 2n + 1 = 3n^2 + 2n + 4$$

```
a=5
b=6
c=10
for i in range(n):
    for j in range(n):
        x = i * i
        y = j * j
        z = i * j
for k in range(n):
    w = a*k + 45
    v = b*b
d = 33
```



>>> Hagamos unos Ejercicio!

Ejercicios con Notación O()

>>> Gracias!

Bibliografía

1. Think Complexity: Complexity Science and Computational Modeling. Allen Downey - O'Reilly 2012.
2. Problems on Algorithms. Second Edition. Ian Parberry and William Gasarch, July 2002.
3. Cormen, Leiserson, Rivest, Stein. Introduction to Algorithms, MIT Press.
4. Dasgupta, Papadimitriou, Vazirani. Algorithms, McGraw-Hill Press.
5. Fotakis. Course of Algorithms and Complexity at the National Technical University of Athens.