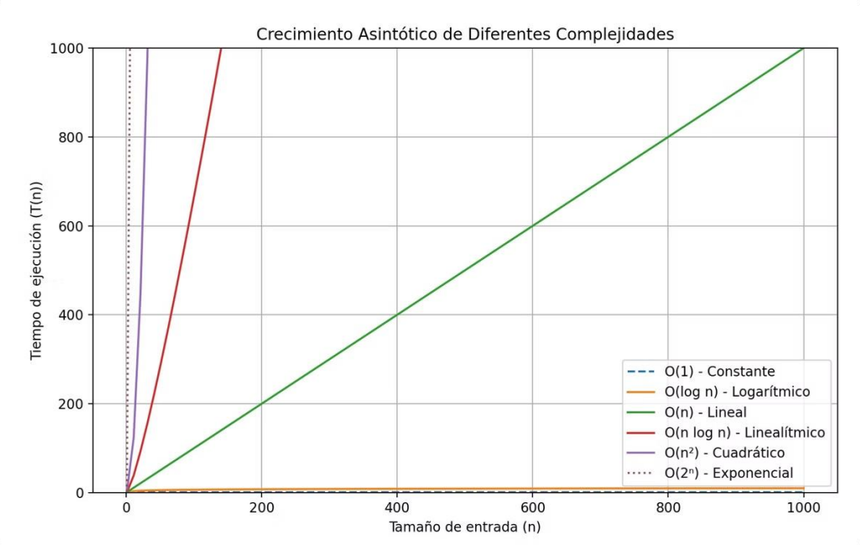
Notación Big-O

Conceptos clave y ejemplos de código

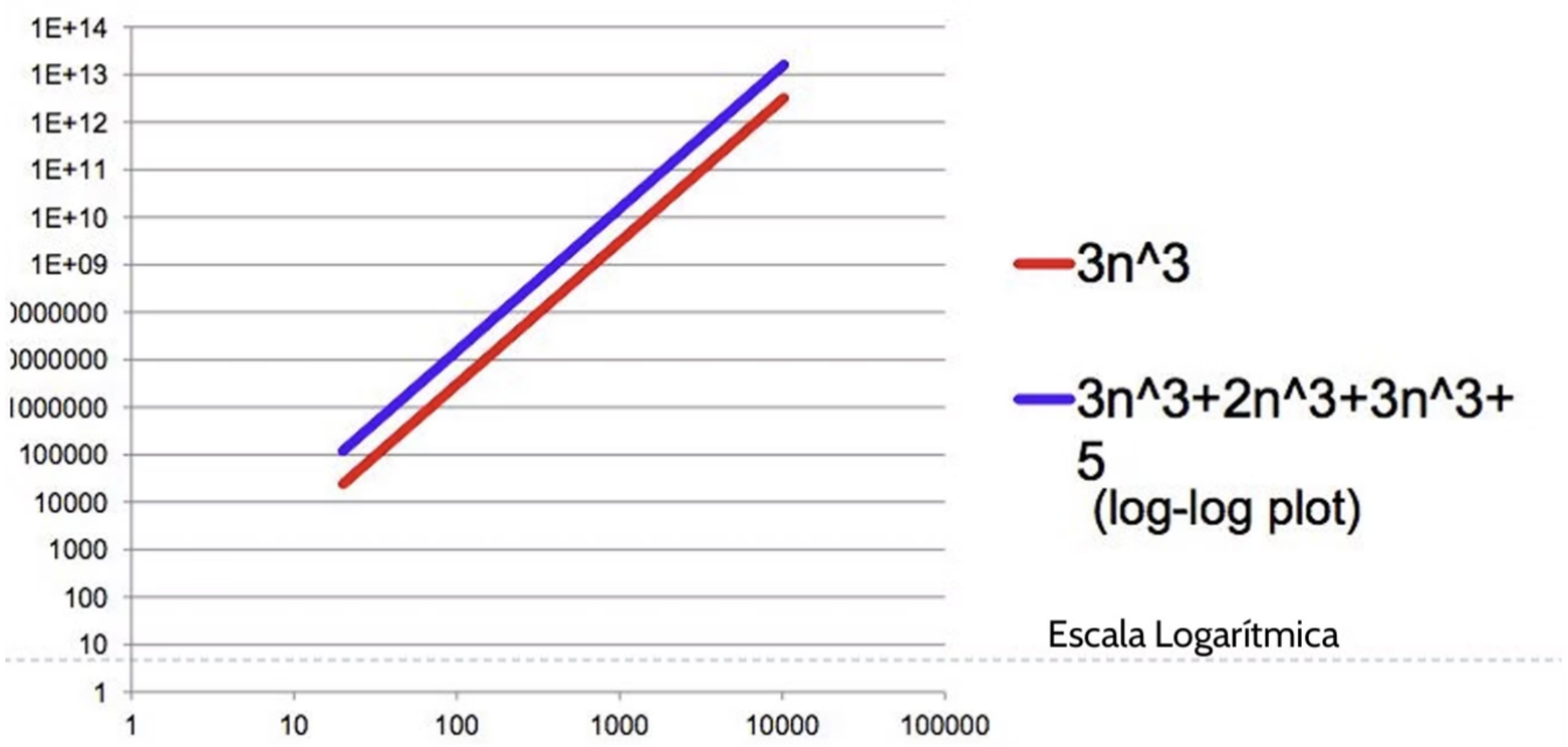


Análisis Teórico

- Supón que tienes dos algoritmos, A y B, con las siguientes funciones de tiempo:
- TA (n)=3n^3
- TB (n)=3n^3+2n^2+3n+5
- ¿Qué algoritmo es más eficiente?

Análisis Teórico

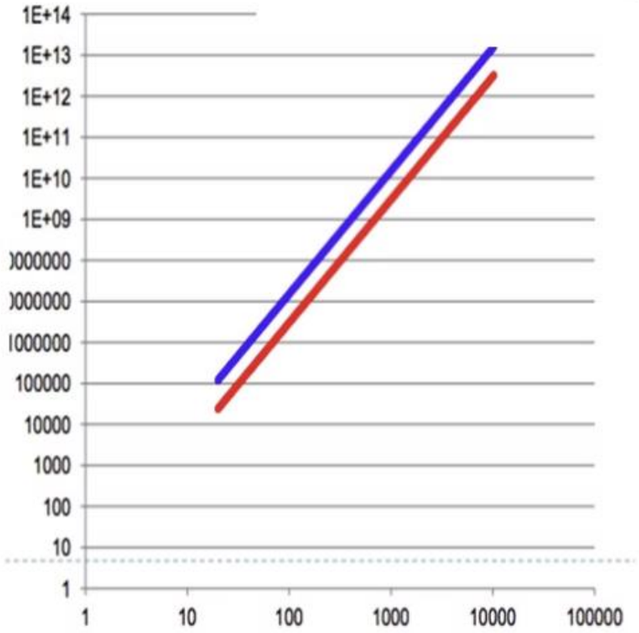
- TA (n)= 3n^3
- TB (n)=3n^3+2n^2+3n+5



Análisis Teórico

Dos funciones, f y g, son asintóticamente equivalentes cuando:

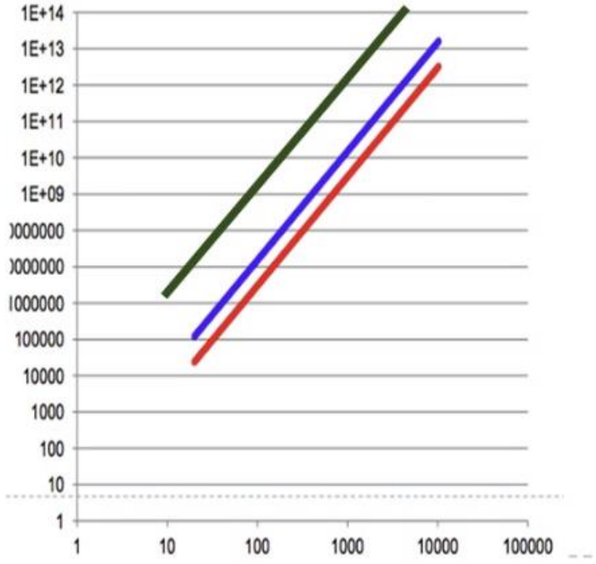
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$



$$\lim_{n\to\infty} \frac{3n^3+2n^2+3n+5}{3n^3} = 1$$

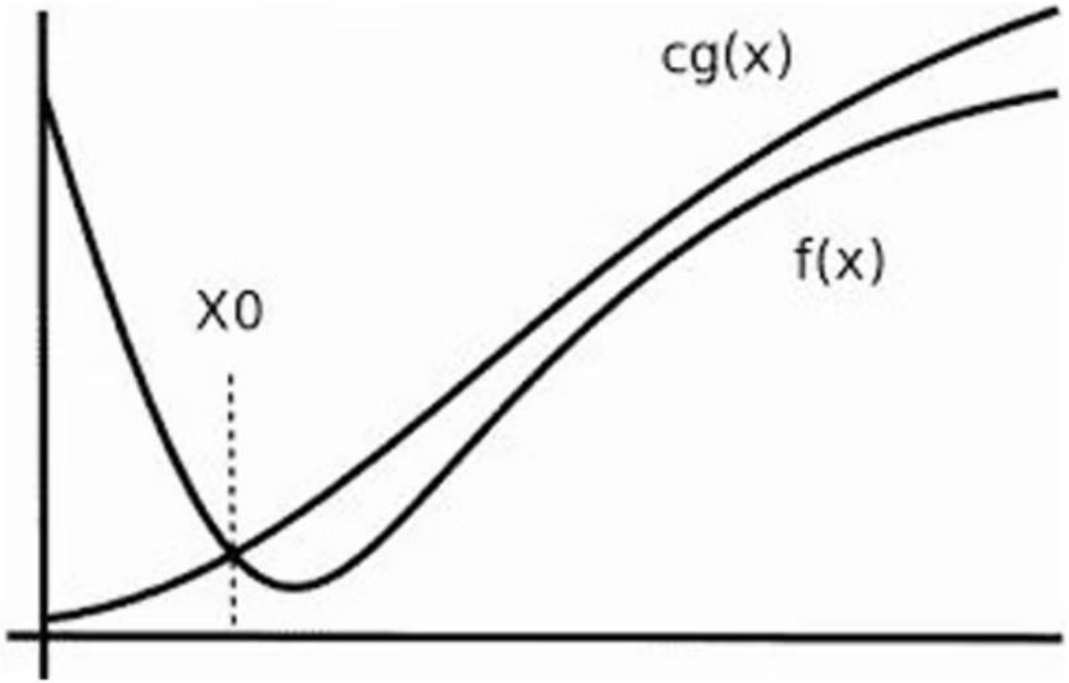
Análisis Teórico

- Tiempo de ejecución depende:
- ○ De la máquina en la que se ejecuta el programa.
- ○ Del compilador utilizado para generar el programa.
- Para facilitar la comparación de las funciones temporales, vamos a aproximar cada función temporal a una cota superior (análisis asintótico)



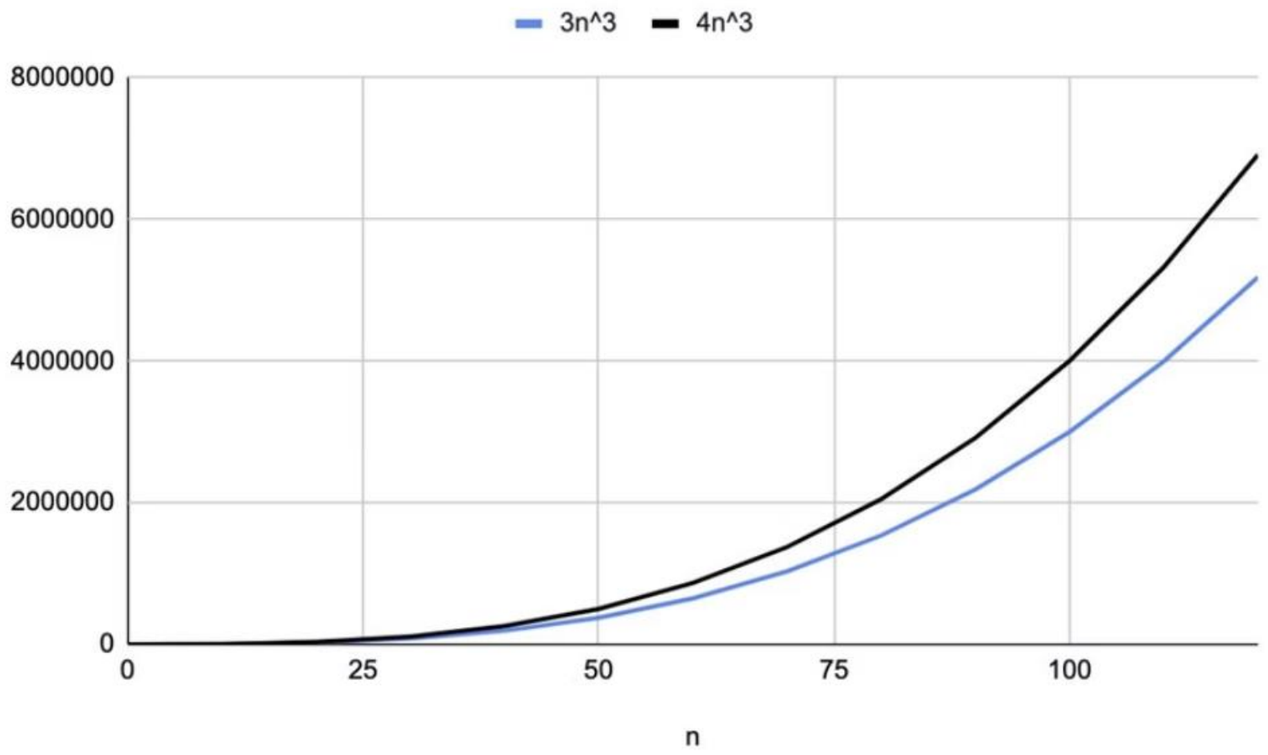
Análisis Teórico – cota superior asintótica

- Dadas dos funciones, f(n) y g(n), g(n) es de orden superior f(n), si existen n0 >0 y c>0, se cumple: f(n) <= cg(n), para todo n >= n0



Análisis Teórico – cota superior asintótica

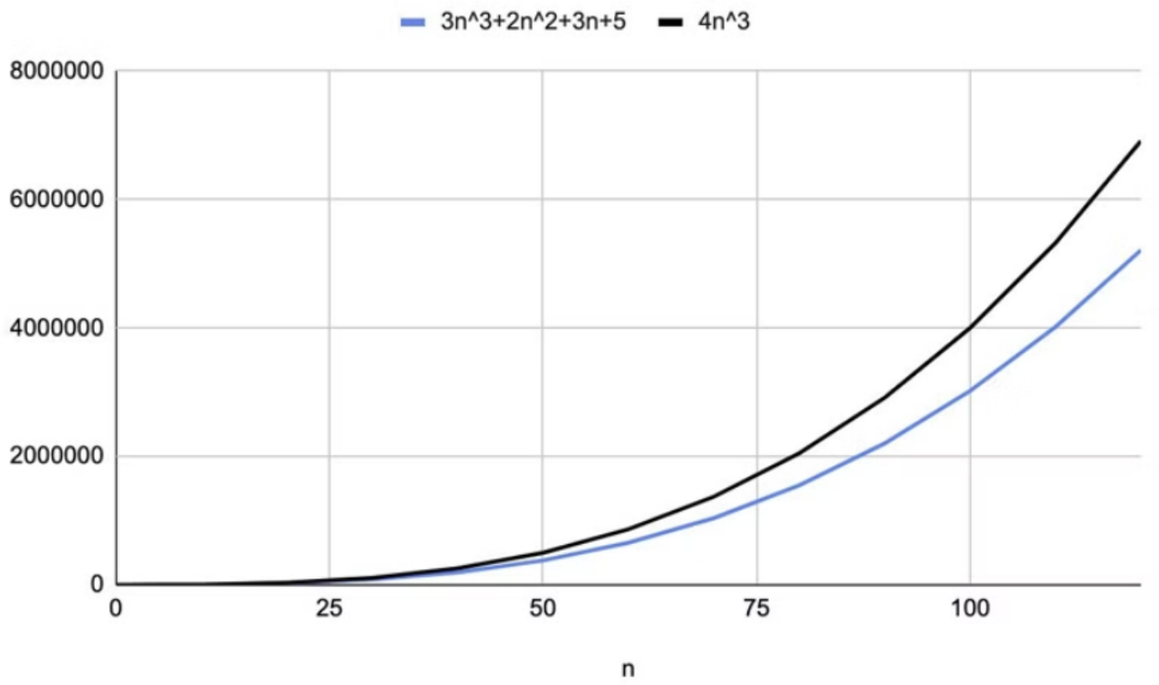
- TA (n)= 3n^3 es de orden superior n^3 porque existen n0 =0, c=4, tales que TA (n) <= cn^3 , para todo n>=n0



n	3n^3	4n^3
0	0	0
5	375	500
10	3000	4000
20	24000	32000
30	81000	108000
40	192000	256000
50	375000	500000
60	648000	864000
70	1029000	1372000
80	1536000	2048000
90	2187000	2916000
100	3000000	4000000
110	3993000	5324000
120	5184000	6912000

Análisis Teórico – cota superior asintótica

- T B (n)= 3n^3+2n^2+3n+5 es de orden superior n^3 porque existen n0 =10, c=4, tales que TB (n) <= cn^3 , para todo n>=n0



n	3n^3+2n^2+3n+5	4n^3
0	5	0
5	445	500
10	3235	4000
20	24865	32000
30	82895	108000
40	195325	256000
50	380155	500000
60	655385	864000
70	1039015	1372000
80	1549045	2048000
90	2203475	2916000
100	3020305	4000000
110	4017535	5324000
120	5213165	6912000

Análisis Teórico – cota superior asintótica

- ¿Cómo proponer una cota superior para una función T(n)

1. Buscar el término que crece más rápido (término de mayor grado).
2. Eliminar su coeficiente.

Análisis Teórico – cota superior asintótica

- Buscar un límite superior a la función T(n)

1. **Buscar el término que crece más rápido.**
2. Eliminar su coeficiente.

●TA (n)=3n^3 -> 3n^3

●TB (n)=3n^3+2n^2+3n+5 -> 3n^3

Análisis Teórico – cota superior asintótica

- Buscar un límite superior a la función T(n)

1. Buscar el término que crece más rápido.
2. **Eliminar su coeficiente.**

- TA (n)=3n^3 -> 3n^3 -> n^3
- TB (n)=3n^3+2n^2+3n+5 -> 3n^3 -> n^3
- Cota superior también se llama función **Big O**

Análisis Teórico – cota superior asintótica

T(n)		Big-O
n+2	->n	n
(n-1)*(n+1)/2	-> n^2/2	n^2
7*n^4+5n^2+1	-> 7*n^4	n^4
n*(n-1)	-> n^2	n^2
3*n+log(n)	-> 3*n	n

Órdenes de Complejidad – BIG -O

notación	nombre
O(1)	Constante.
O(log n)	Logarítmica.
O(n)	Lineal.
O(n log n)	Lineal-logarítmica.
$O(n^2)$	Cuadrática.
O(2^n)	Exponencial
O(n!)	factorial

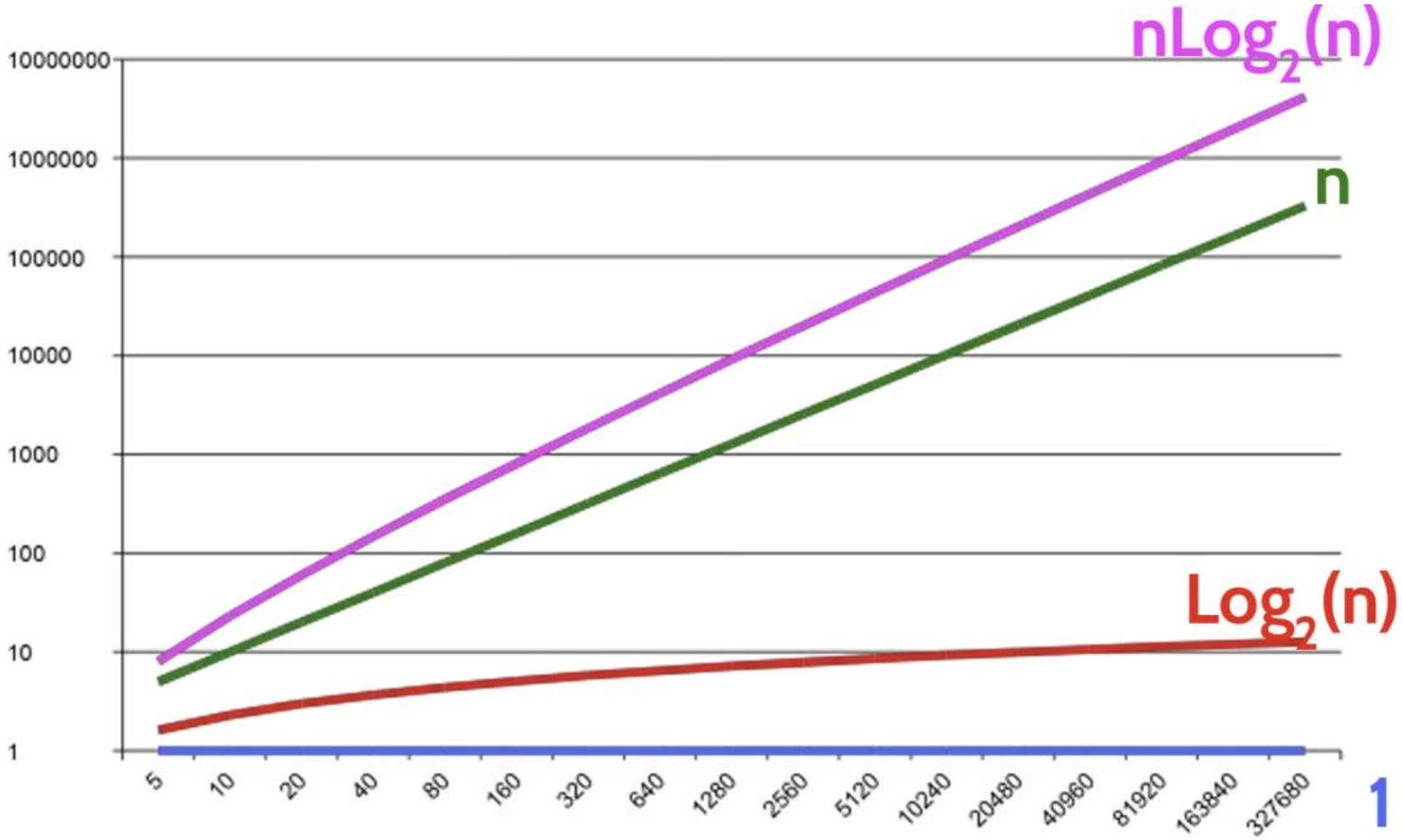
Órdenes de Complejidad – BIG -O

Buenas noticias!!!: Un conjunto pequeño de instrucciones:

```
1 < log n < n < nlog n < n^2 < n^3 < ... < 2^n < n!
```

Órdenes de Complejidad – BIG -O

Órdenes eficientes de complejidad

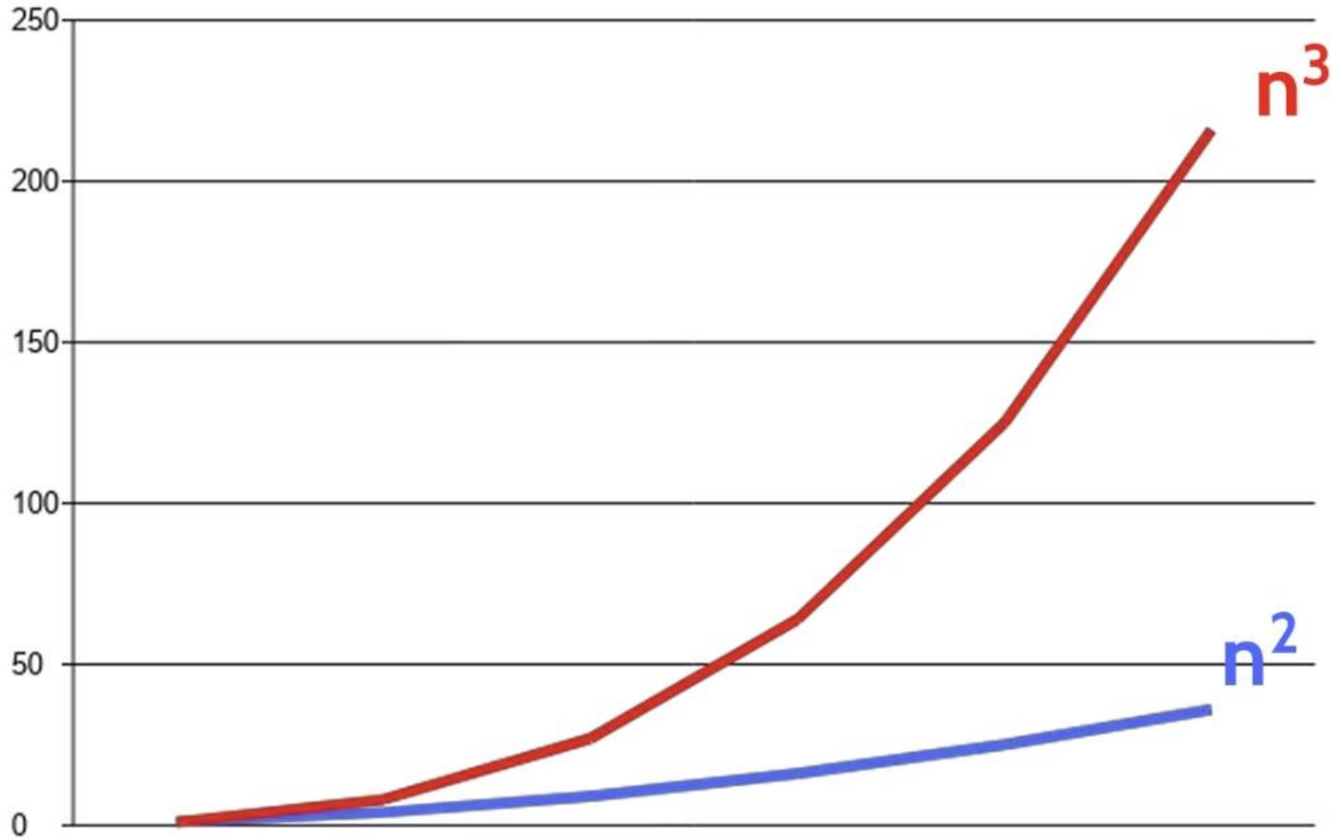


Órdenes de Complejidad – BIG -O

Notación	Nombre	Descripción	Ejemplo
1	Constante.	Independiente del tamaño	Borrar el primer elemento de una cola
$\log_2 n$	Logarítmica.	Dividir por la mitad	Busqueda Binaria
n	Lineal.	Bucle	Suma de los elementos de una lista
$n \log_2 n$	Lineal-logarítmica.	Divide y Venceras	Mergesort, Quiksort

Órdenes de Complejidad – BIG -O

Órdenes tratables de complejidad

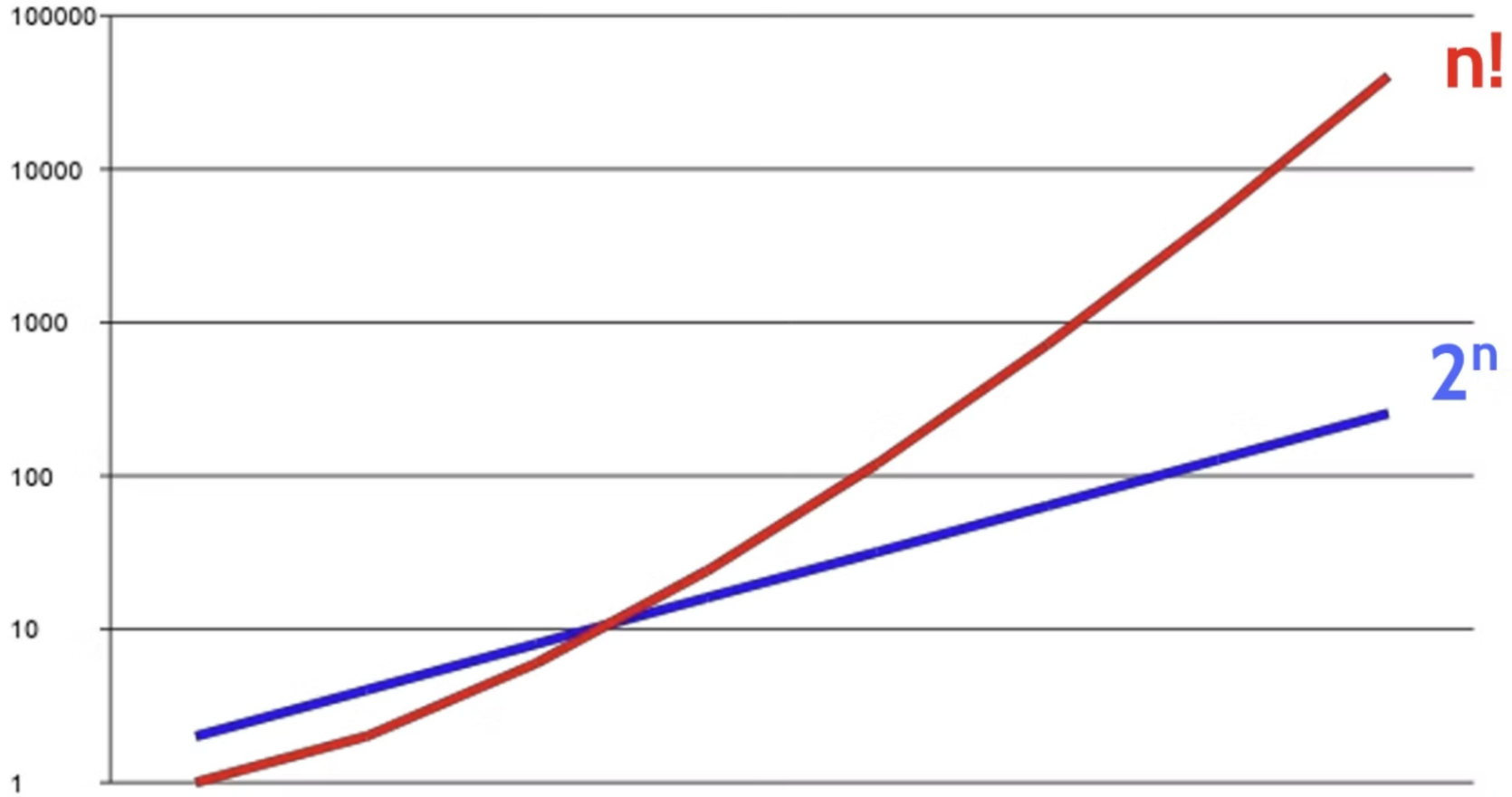


Órdenes de Complejidad – BIG -O

Notación	Nombre	Descripción	Ejemplo
n^2	Cuadrática.	Bucles dobles	Sumar dos matrices; Metodo burbuja
n^3	Cubica.	Bucles triples	Multiplicar 2 matrices

Órdenes de Complejidad – BIG -O

Órdenes intratables de complejidad



Órdenes de Complejidad – BIG -O

Notación	Nombre	Descripción	Ejemplo
k^n	Exponencial	Busqueda fuerza bruta	Adivinar una password
n!	Factorial	Busqueda fuerza bruta	Enumerar todas las posibles particiones de un conjunto.

Análisis BIG-O

```
def contains(data: list, x: int) -> int:
    for c in data:  # n veces
        if c==x:    #1 operacion n veces - O(n)
            return True # 1 operacion – O(1)
    return False #1 operacion
```

Análisis BIG-O

```
def sum_list(data: list) -> int:
    total=0 # 1 operacion – O(1)
    for c in data: # n veces
        total = total + c # 2 operaciones n veces – O(n)
    return total
```

Resumen

- El análisis teórico consiste en encontrar una función (cota superior) para la función temporal del algoritmo.
- El conjunto de cotas superiores (o funciones Big-O) es un conjunto pequeño: ○ $1 < \log n < n < n \log n$
- Este conjunto nos va a permitir clasificar y comparar las funciones temporales de nuestros algoritmos sin necesidad de implementar ni requerir un entorno de pruebas sw/hw.