

# Notación $O$ , omega y teta.

Asignatura: Estructura de Datos y Algoritmos

Integrantes:

- Badillo Aguilar Diego
- Herrera Argumedo Luis Diego
- Reyes Fuentes José Manuel
- Wong Sánchez Yibran Lee

# Introducción - ¿Por qué el análisis de algoritmos?



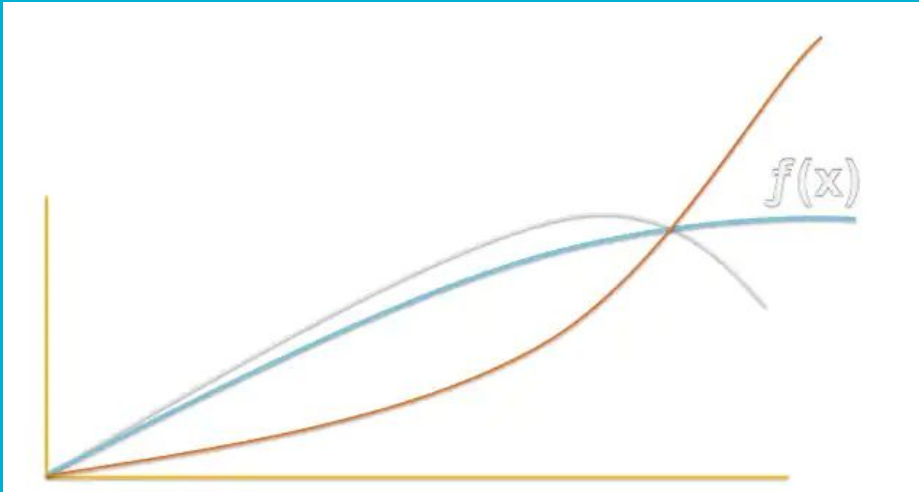
Determinar tiempos de respuesta (runtime) ☐ Determinar recursos computacionales ☐

**Aproximación teórica** ☐ Generaliza el número de operaciones que requiere un algoritmo para encontrar la solución a un problema

---

# Notación Asintótica

Ayuda a describir el comportamiento de un algoritmo en tiempo (cuánto tarda en ejecutarse) o en memoria (cuánta memoria va a necesitar para ejecutarse) basado en la cantidad de elementos de entrada



## Ventajas

Elección de algoritmos eficientes  
para resolver problemas  
específicos □ No depende de lenguajes  
de programación ni de hardware

## Desventajas

Para muchos casos, en análisis no  
es trivial



Para comenzar con el análisis del algoritmo necesitamos:

- Conocer la complejidad del problema que resuelve el algoritmo
- Conocer la dimensión de la entrada (número de elementos)
- Determinar el número de operaciones a realizar

La complejidad del algoritmo se expresan a través de una función matemática

Un algoritmo puede estar compuesto de dos o más operaciones, por lo que determinar la complejidad depende de identificar la operación más costosa en el algoritmo

[illegible]

**Peor Caso:** caso más extremo, donde se considera el tiempo máximo para solucionar un problema

**Caso promedio:** caso en el cual, bajo ciertas restricciones, se realiza un análisis del algoritmo

**Mejor caso:** caso ideal en el cual el algoritmo tomará el menor tiempo para dar una respuesta

# Análisis de Casos



# Notación Asintótica

Para determinar la complejidad de un algoritmo, se siguen los siguientes pasos: □

- Se analiza el algoritmo para determinar una función que represente el número de operaciones a realizar por el mismo
- Se define en términos de funciones matemáticas, el orden de la función □
- Se clasifica de acuerdo a su complejidad



# Notación O (Límite superior)

En las ciencias de la computación, la notación O grande (también llamada notación asintótica) se usa para describir la complejidad de los algoritmos o el rendimiento de los algoritmos.

Esto significa que la notación O grande es usada para describir el tiempo de ejecución o el espacio utilizado por un algoritmo relativo al input cuando el input se hace arbitrariamente grande.

En general, la notación O grande muestra el peor de los casos, porque matemáticamente hablando, se supone que la O grande representa la cota superior de una función.



# Tipos de notaciones $O$

$O(1)$ : constante.

$O(n)$ : lineal.

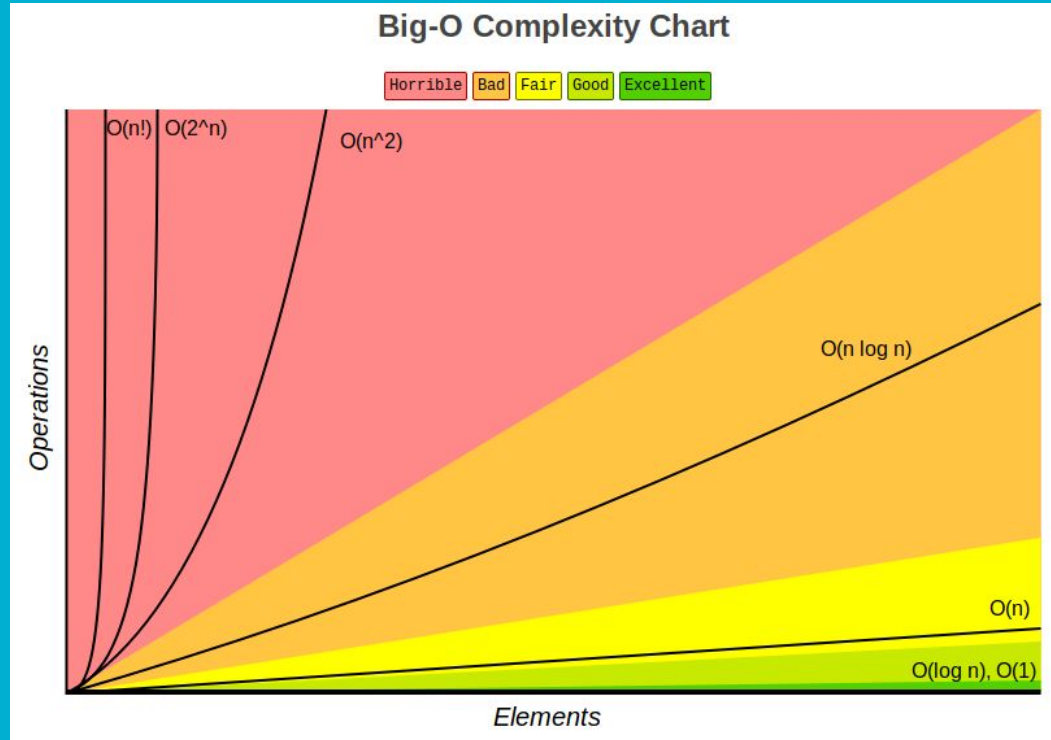
$O(\log n)$ : logarítmica.

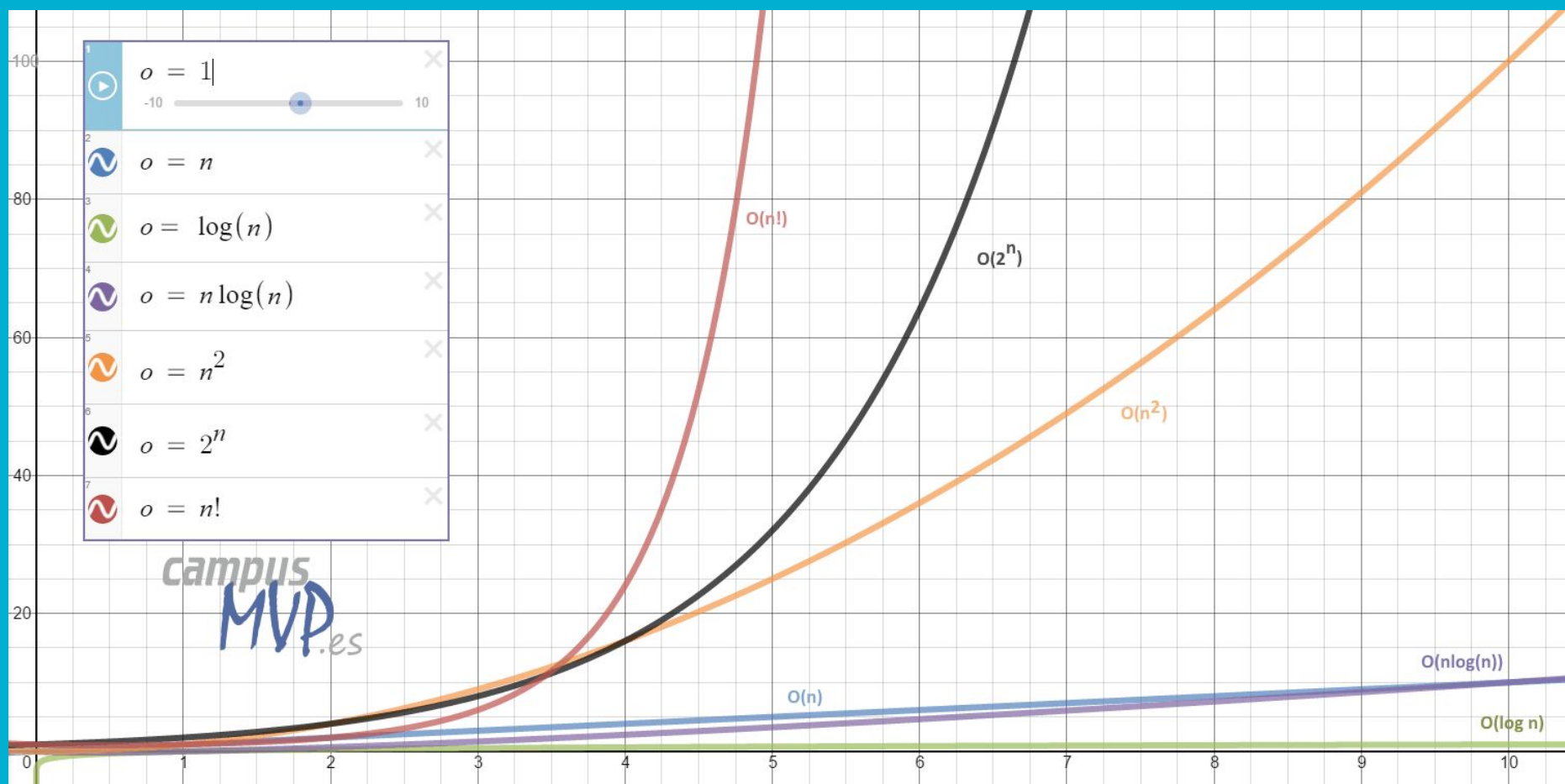
$O(n \log n)$ : " "

$O(n^2)$ : cuadrática.

$O(2^n)$ : exponencial.

$O(n!)$ : explosión  
combinatoria.





En el eje horizontal tenemos el número de elementos, y en el vertical el tiempo

# Notación “o” (o pequeña)

Se utiliza para comparar tiempos del mismo orden

Aparecen constantes, por lo que es necesario hacer superposiciones sobre: conste de las operaciones, conteo de operaciones

En little-o, debe ser que hay un mínimo  $x$  después de lo cual la desigualdad se mantiene sin importar lo pequeño que sea  $k$  siempre que no sea negativo o cero.

En resumen

# Big-O

$\geq$  Complejidad  
en el tiempo

# Little-o

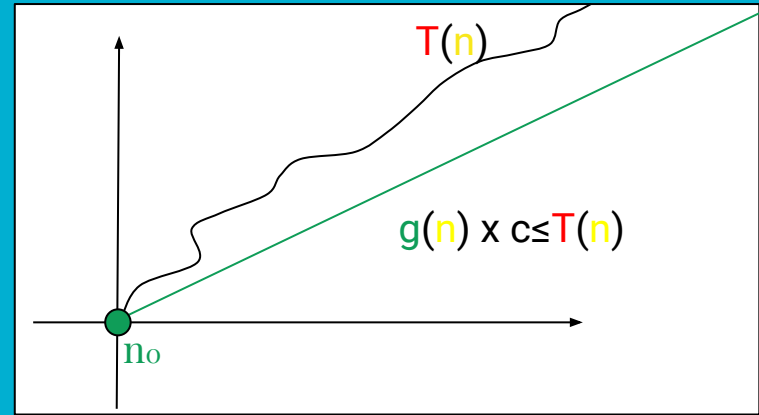
$>$  Complejidad  
en el tiempo

# Notación Big Omega $\Omega$ (Límite inferior)

- Busca encontrar un crecimiento menor o igual que la complejidad en el tiempo
- Notación que sirve para indicar la mínima cantidad de recursos que un algoritmo necesita para alguna clase de entrada. La cota inferior de un algoritmo, denotada por el símbolo  $\Omega$ , pronunciado "Gran Omega" u "Omega", tiene la siguiente definición:

$$T(n) = \Omega(g(n))$$

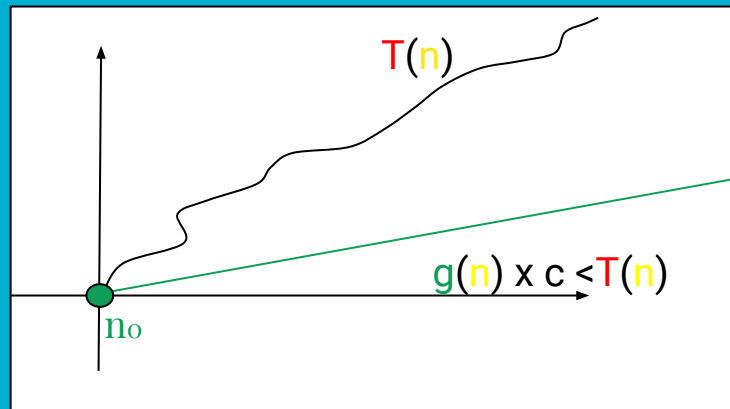
$$g(n) \times c \leq T(n)$$



# Notación little Omega $\omega$ (Límite inferior)

- Muy parecido a Big  $\Omega$
- Busca encontrar un crecimiento menor que la complejidad en el tiempo
- Tiene la siguiente definición:

$$\underbrace{T(n) = \Omega(g(n))}_{g(n) \times c < T(n)}$$



Mayor	$2^n$
	$n^2$
	$n$
	$\log(n)$
Menor	$1$

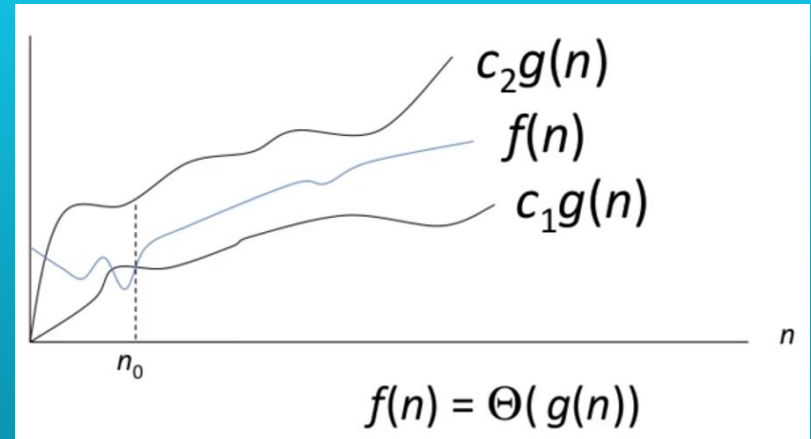
# Notación Theta

Big Omega nos dice el límite inferior del tiempo de ejecución de una función, y Big O nos dice el límite superior.

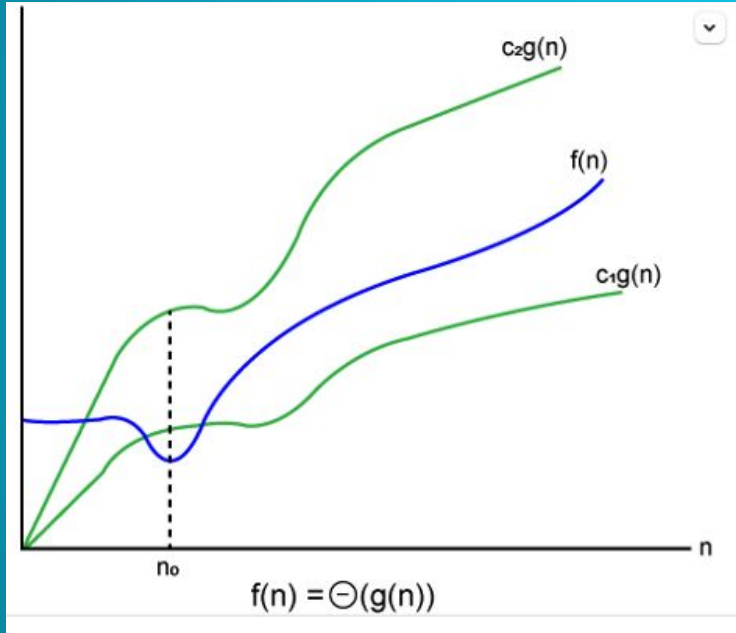
Pero, ¿qué pasa cuando son iguales? Entonces podemos dar un límite theta ( $\Theta$ ): nuestra función se ejecutará en ese tiempo, sin importar la entrada que le demos.

$c_1, c_2$  son constantes positivas

Idea principal: a partir de  $n_0$ ,  $f(n)$  siempre queda en medio de  $c_1g(n)$  y  $c_2g(n)$



# Big Theta



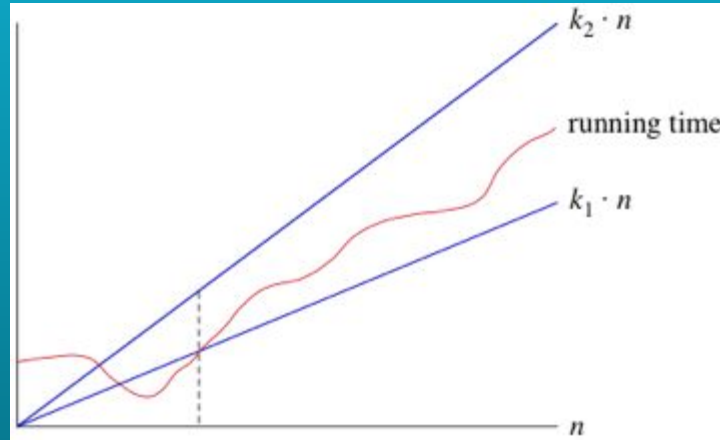
Big theta es el valor de rendimiento exacto del algoritmo o un rango útil entre límites estrechos superior e inferior.

Que en este caso serían Big-O y Big Omega.



# Theta

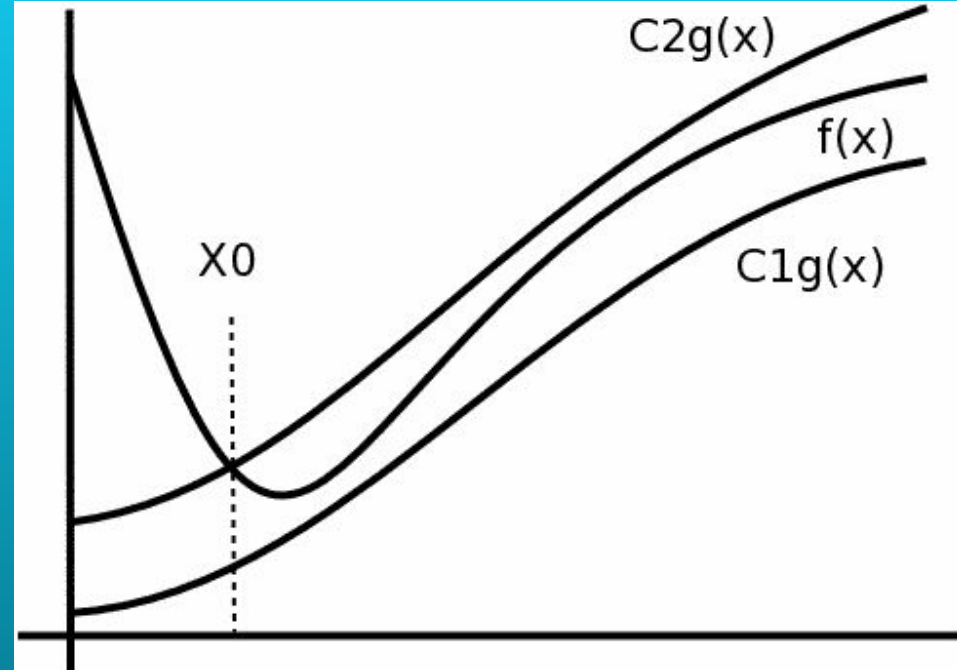
Para valores pequeños de “n”, no nos importa cómo se compara el tiempo de ejecución con  $C_1$  o  $C_2$ . Pero una vez que “n” se hace suficientemente grande, sobre o a la derecha de la línea punteada, el tiempo de ejecución debe estar entre  $C_1$  y  $C_2$ . Mientras existan estas constantes, decimos que el tiempo de ejecución es  $\Theta(n)$  /  $\Theta(n)$



# Theta ajustada

Cuando usamos la notación  $\Theta$ , estamos diciendo que tenemos una cota asintóticamente ajustada sobre el tiempo de ejecución.

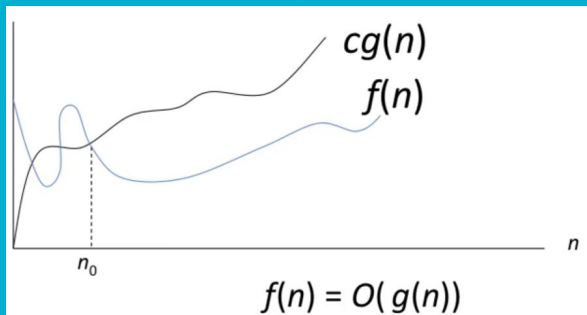
"Asintóticamente" porque importa solo para valores grandes de  $n$ . Se le llama "Cota ajustada" porque ajustamos el tiempo de ejecución dentro del rango de una constante hacia arriba y hacia abajo.



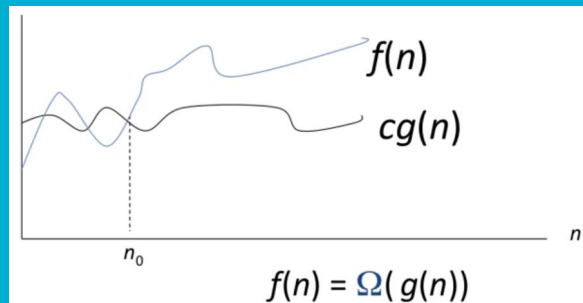
# Relación entre notaciones $O$ , $\Omega$ y $\Theta$

- Notación  $O$ : límite asintótico superior.
- Notación  $\Omega$ : límite asintótico inferior.
- Notación  $\Theta$ : límite asintótico ajustado.

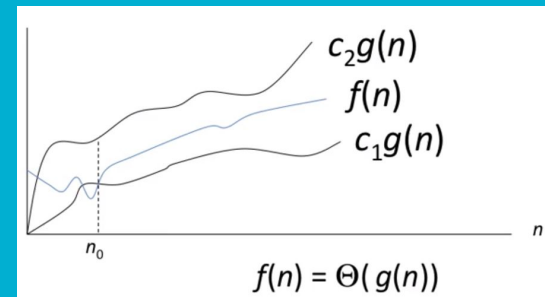
## Notación $O$



## Notación $\Omega$



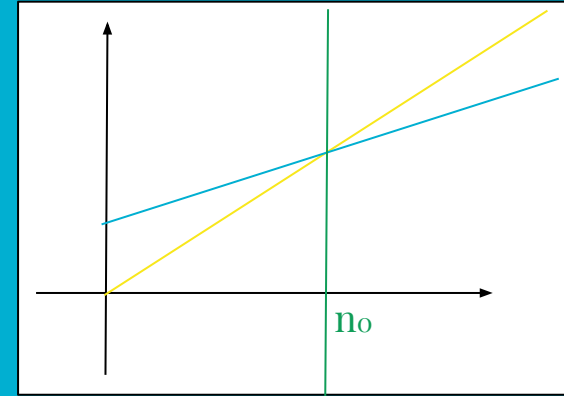
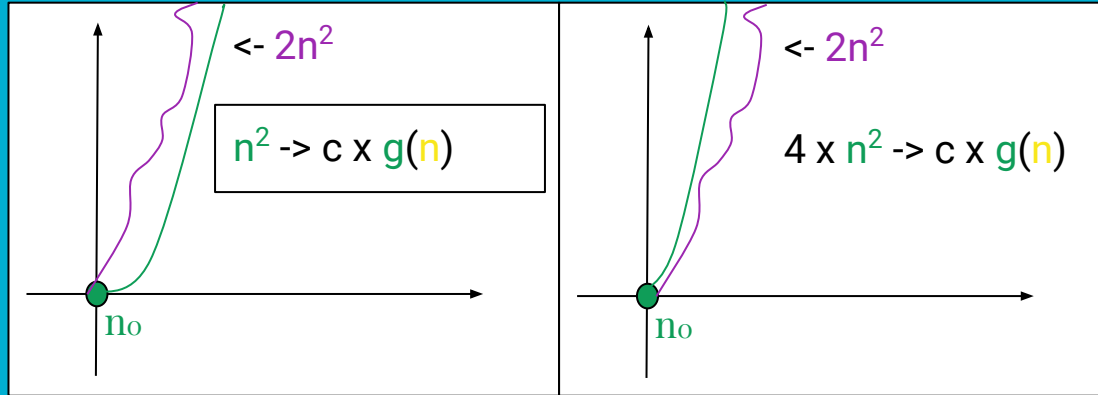
## Notación $\Theta$



# Condiciones de funcionamiento

**C**

**n<sub>0</sub>**



$$T(n) = O(g(n))$$

$\leftrightarrow$

$$c \times g(n) \geq T(n),$$


$$c > 0, n \geq n_0$$

# Ejemplos

## Algunos ejemplos:

- "La entrega estará allí durante su vida".  
(gran O, límite superior)
  - "Puedo pagarte al menos un dólar". (omega grande, límite inferior)
  - "El máximo de hoy será de 25°C y el mínimo de 19°C". (big-theta, estrecho)
  - "Es un kilómetro a pie de la playa".  
(big-theta, exacto)
-

## Ejemplo



```
function catArray(array) {  
  for (let i = 0; i < array.length; i++) {  
    for (let j = 0; j < array.length; j++) {  
      if (i === j && array[i] === array[j])  
        console.log(`miau! ${i} element is equals to ${j} element`)  
    }  
  }  
}
```

# Referencias

- <http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Matematicas%20para%20Computacion/Apuntes/Notacion%20O%20grande.pdf>
- [https://www.cs.buap.mx/~iolmos/ada/Tema2\\_NotacionAsintotica.pdf](https://www.cs.buap.mx/~iolmos/ada/Tema2_NotacionAsintotica.pdf)
- <https://ilusionity.com/77-big-theta-and-asymptotic-notation-explained>