

ANÁLISIS Y DISEÑO DE ALGORITMOS Periodo I - 2023

Jesús Aranda
jesus.aranda@correounivalle.edu.co

Universidad del Valle
Escuela de Ingeniería de Sistemas y Computación

Este documento es una adaptación del material original del profesor Oscar Bedoya



Notación O

Notación Ω

Notación Θ

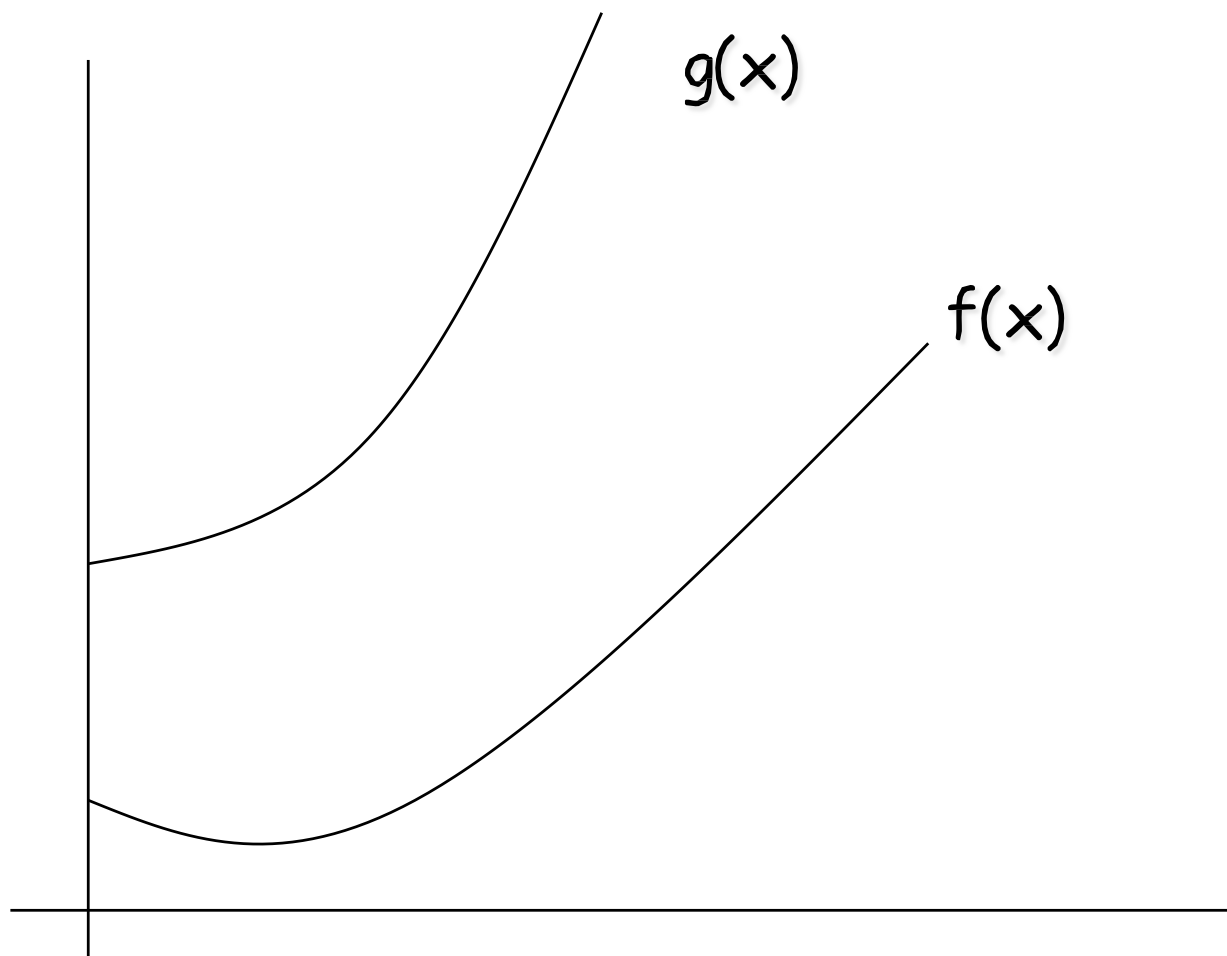
Notación o

Notación ω

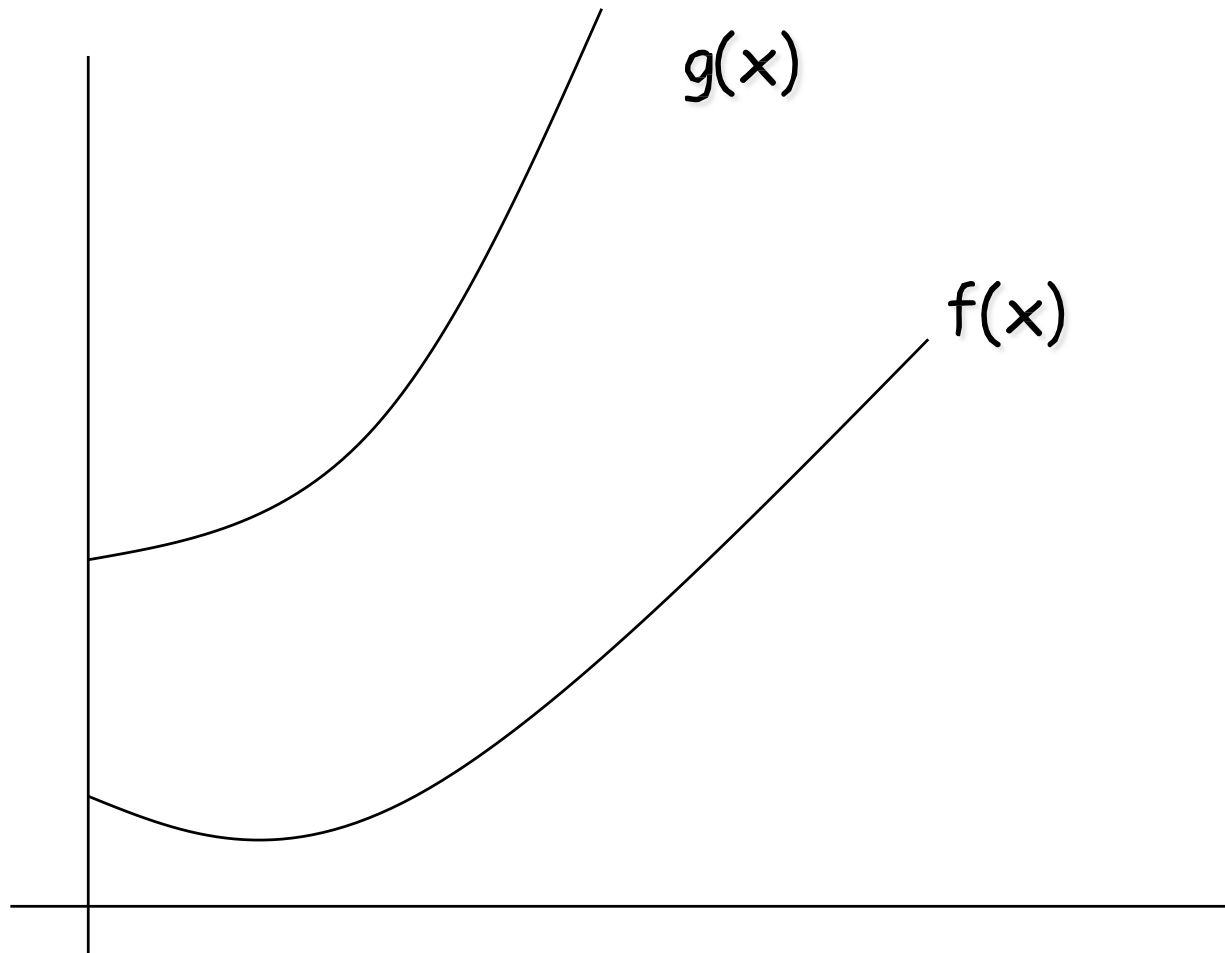
Terminología de complejidades

Clasificación de problemas

Crecimiento de funciones

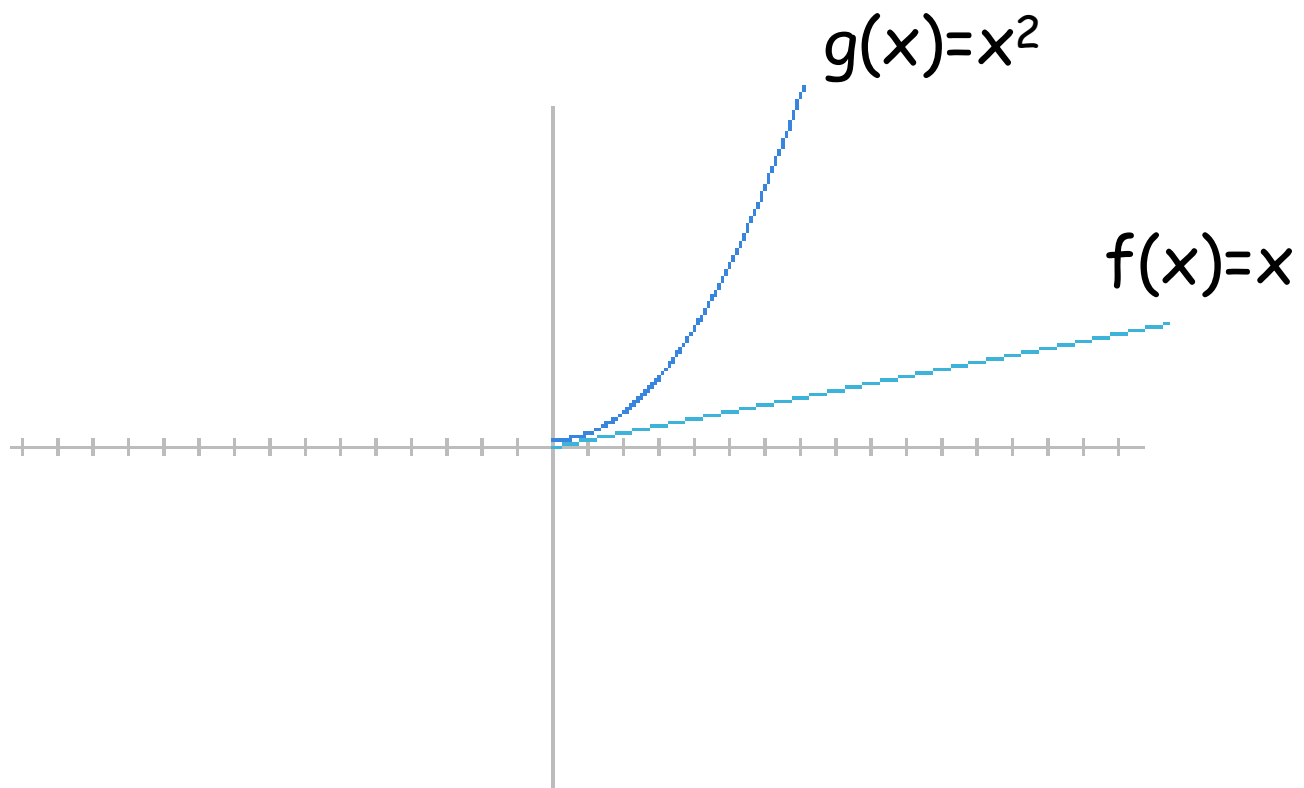


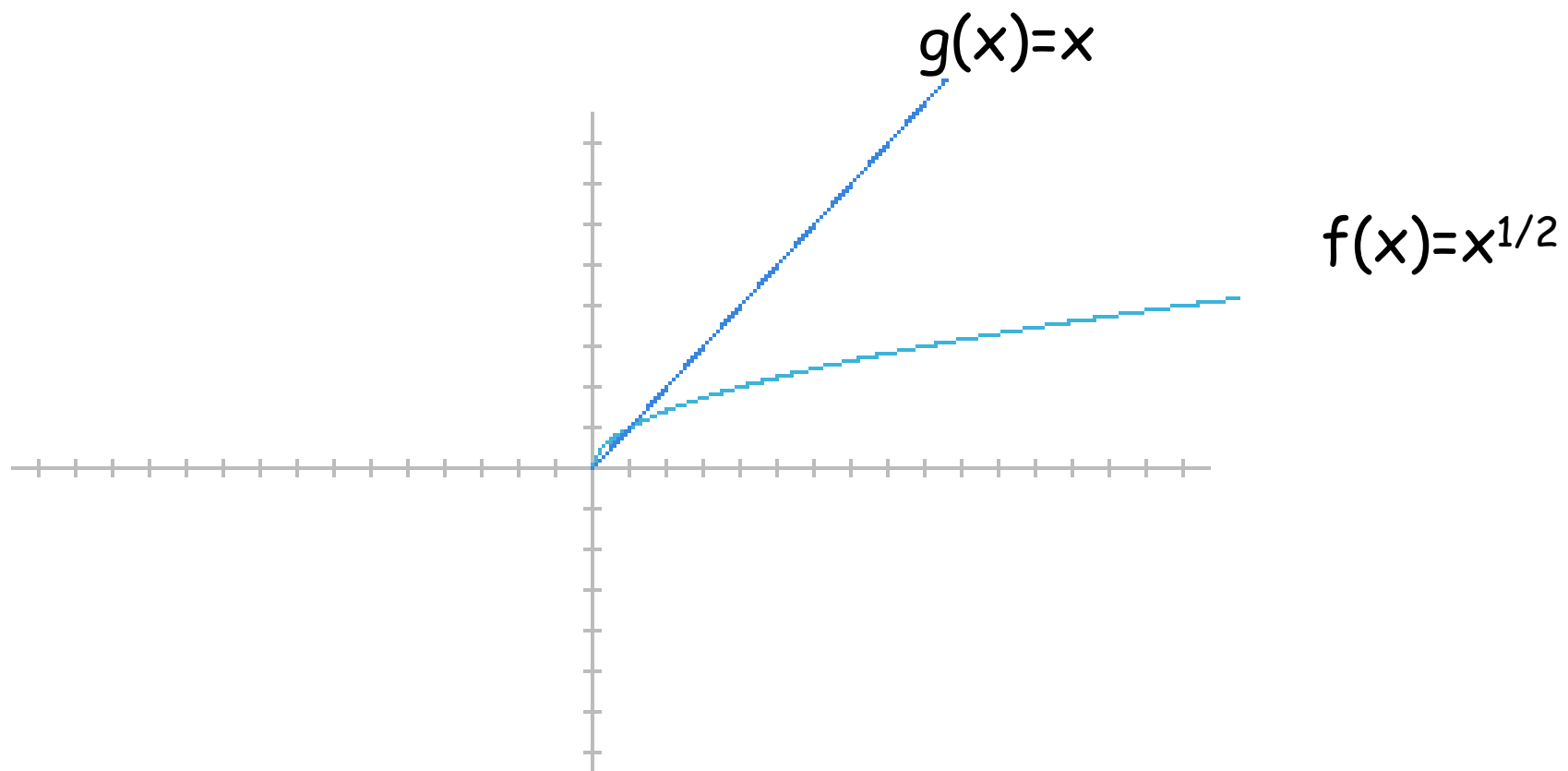
Crecimiento de funciones

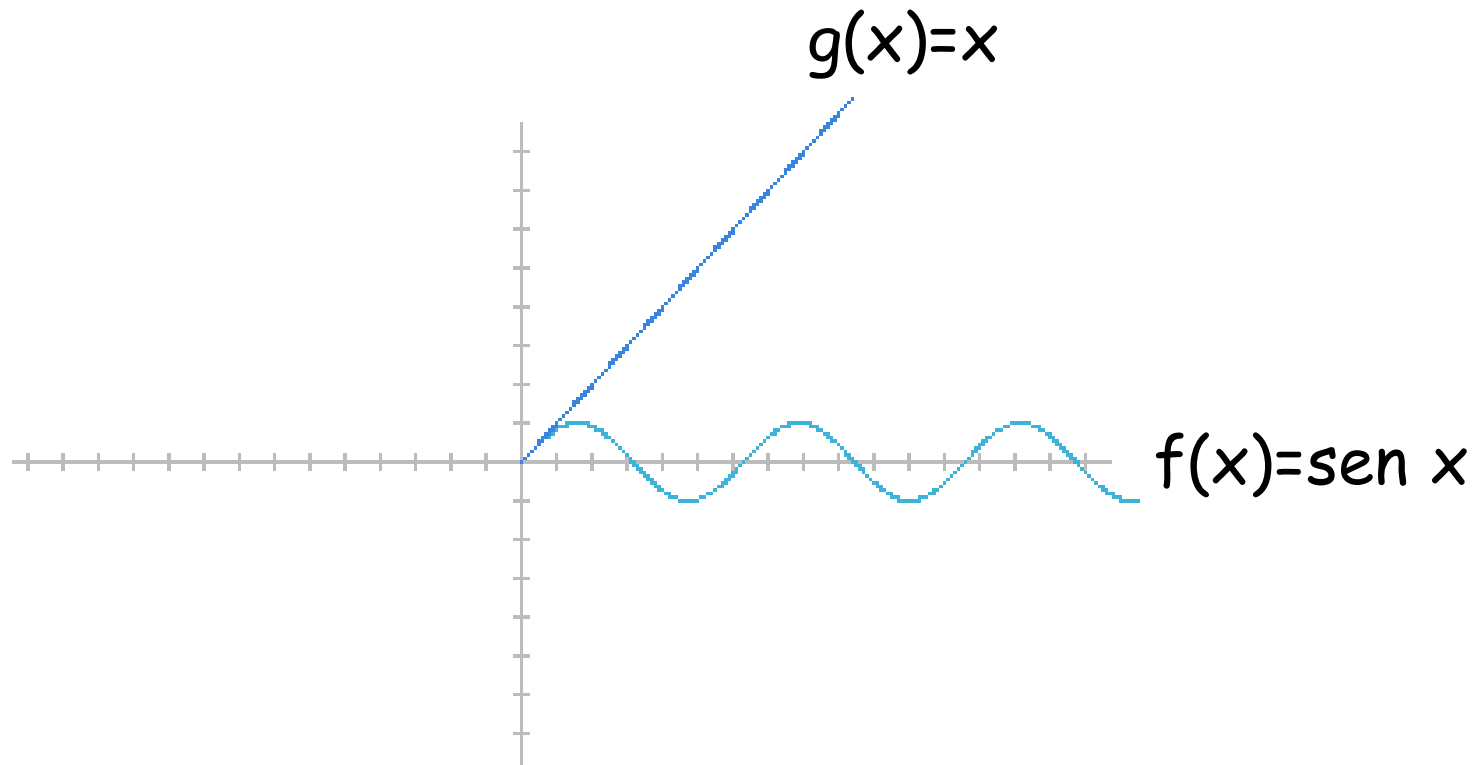


Siempre se cumple que $f(x) \leq g(x)$ para cualquier valor x

$g(x)$ es una cota superior de $f(x)$

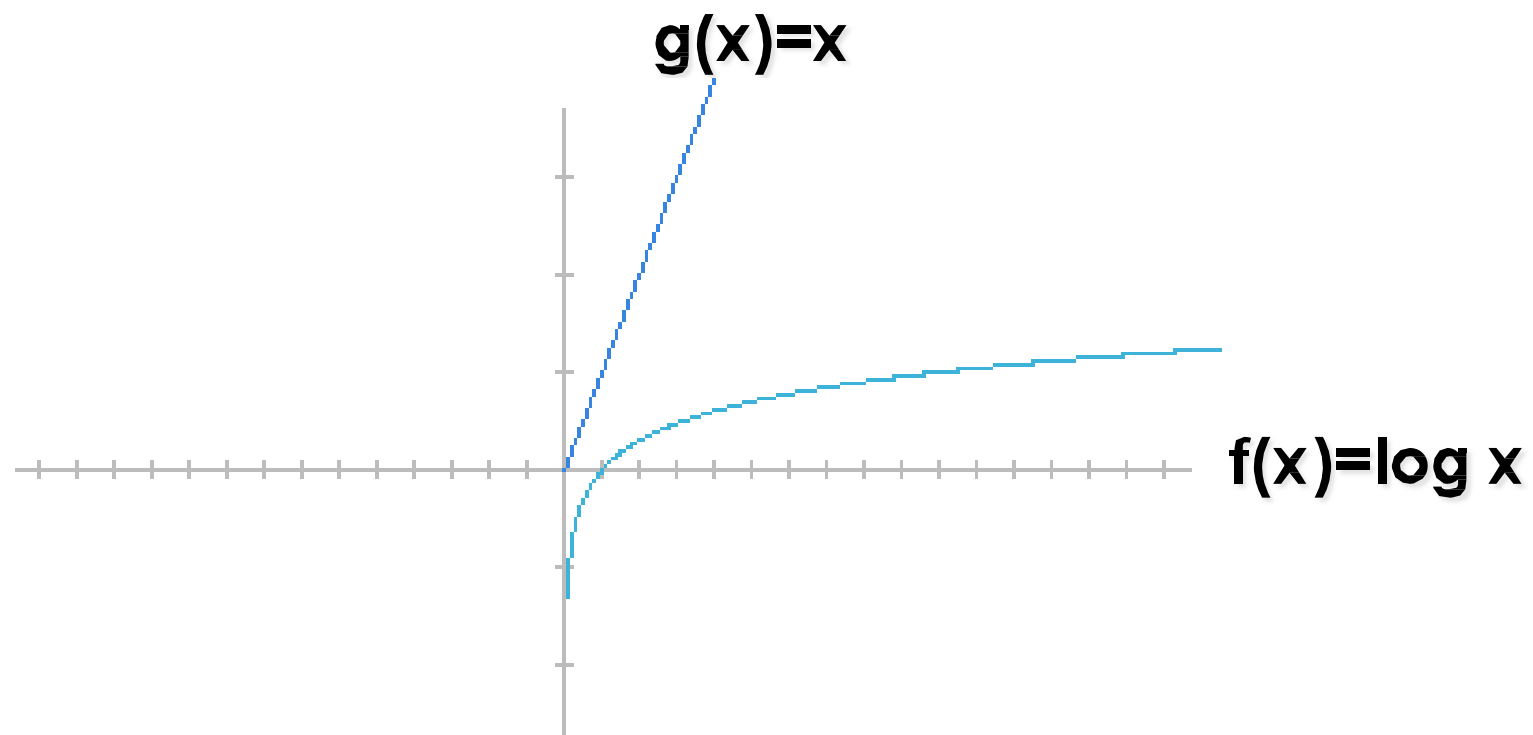


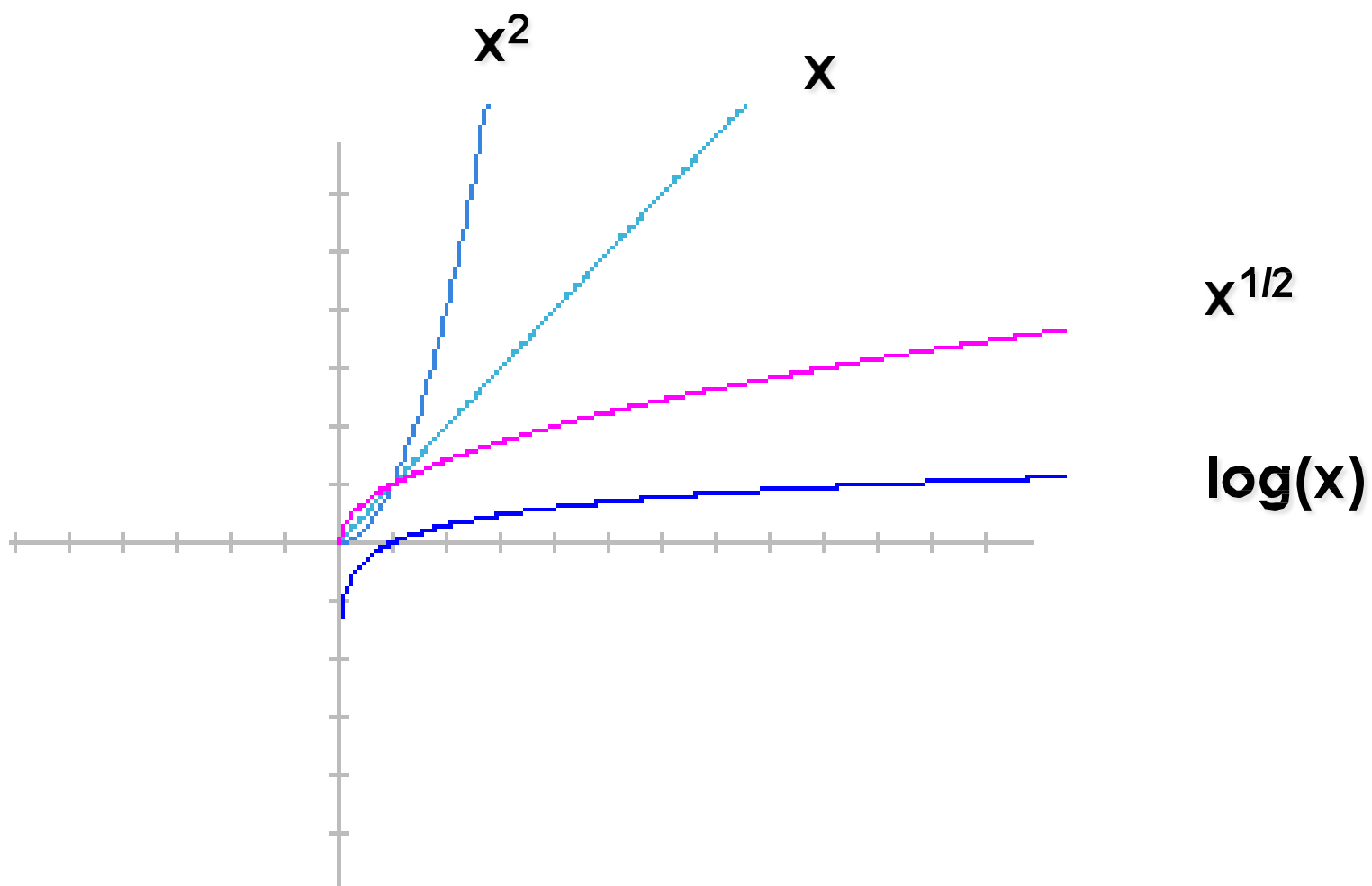


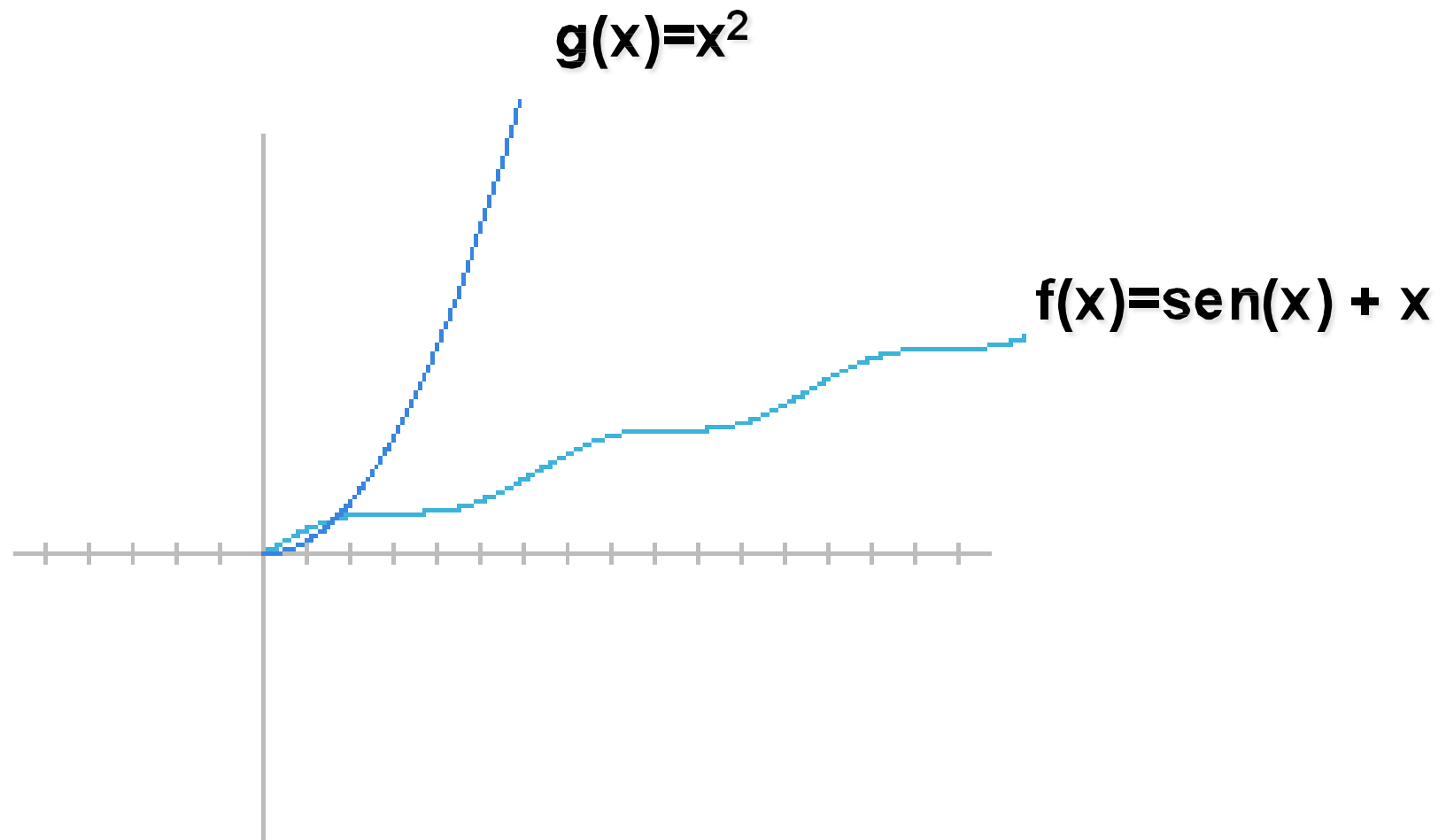


¿Está ($\text{sen } x$) por
debajo de x^2 ?

Solamente se tendrá en cuenta el primer cuadrante, donde tanto X (n o tamaño de entrada) y Y ($f(x)$, $g(x)$, $t(n)$ o costo del algoritmo) toman **valores positivos**.

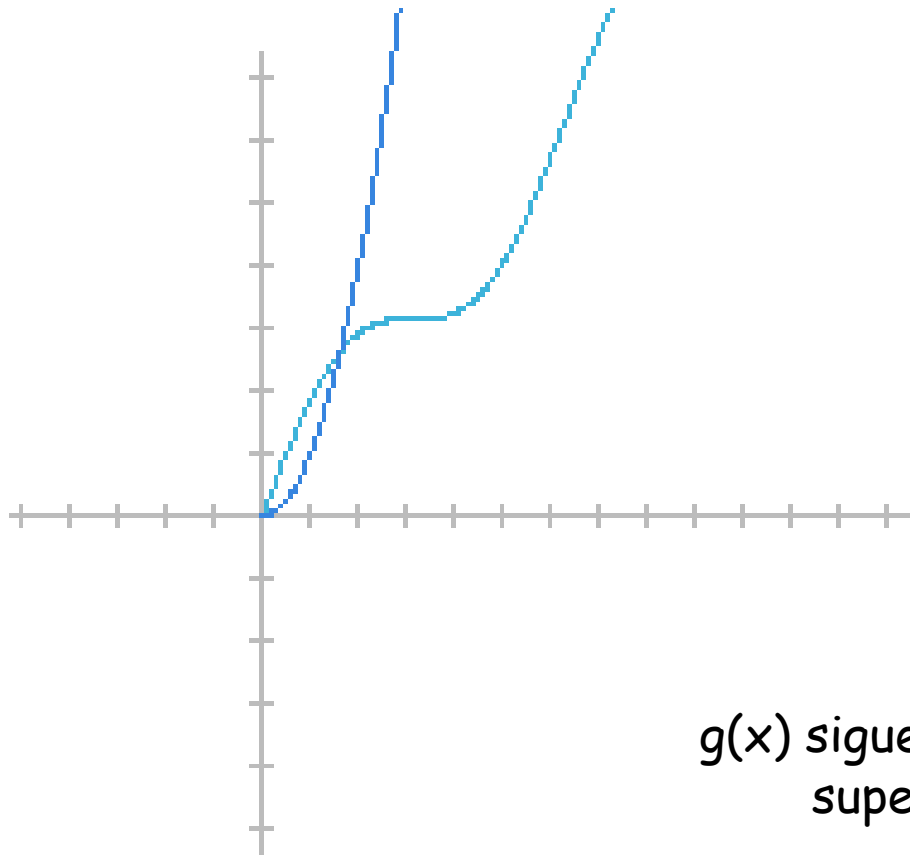






$$g(x)=x^2$$

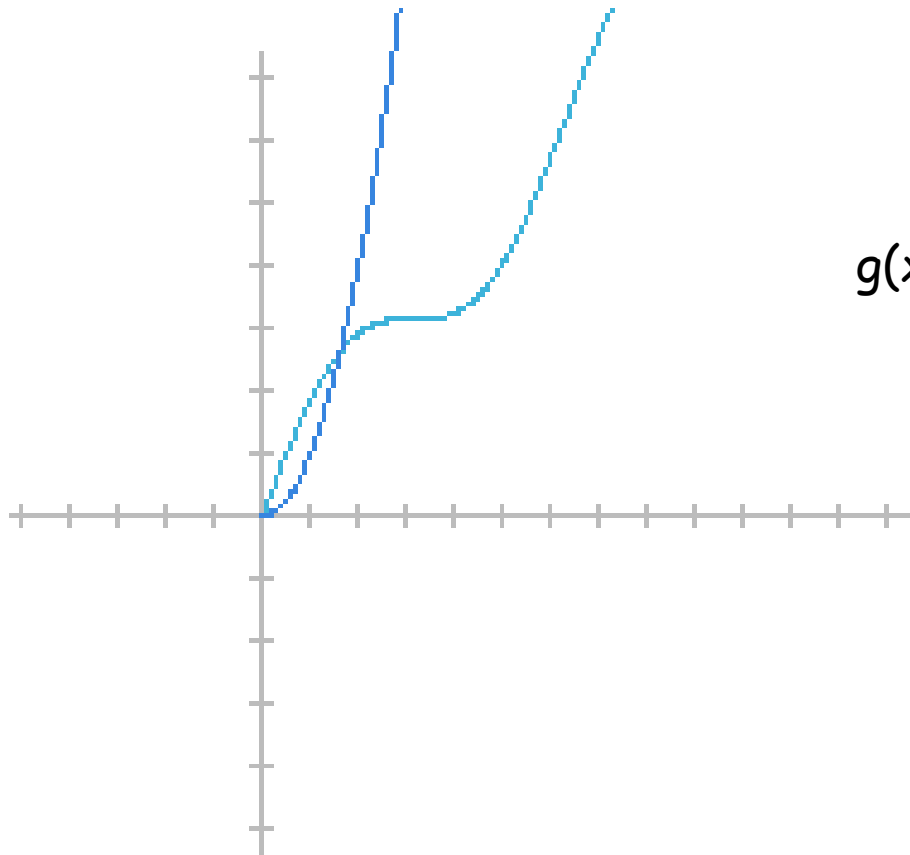
$$f(x)=\text{sen}(x) + x$$



$g(x)$ sigue siendo una cota superior de $f(x)$

$$g(x)=x^2$$

$$f(x)=\text{sen}(x) + x$$

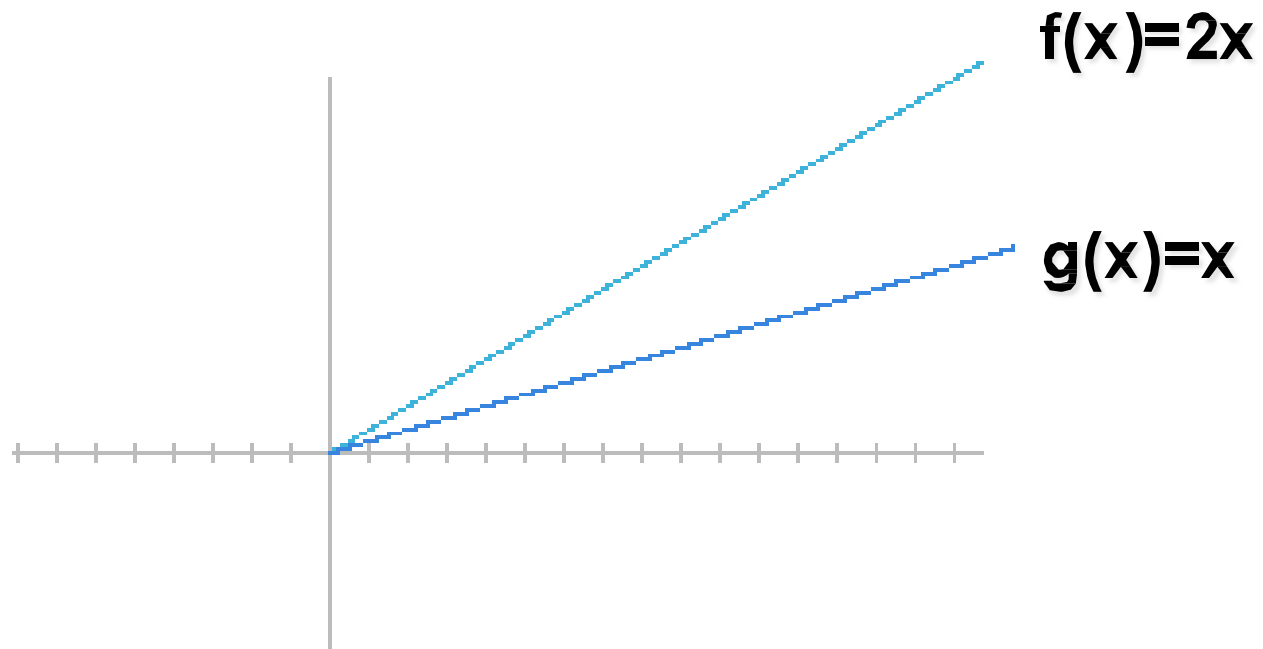


$g(x)$ sigue siendo una cota superior de $f(x)$

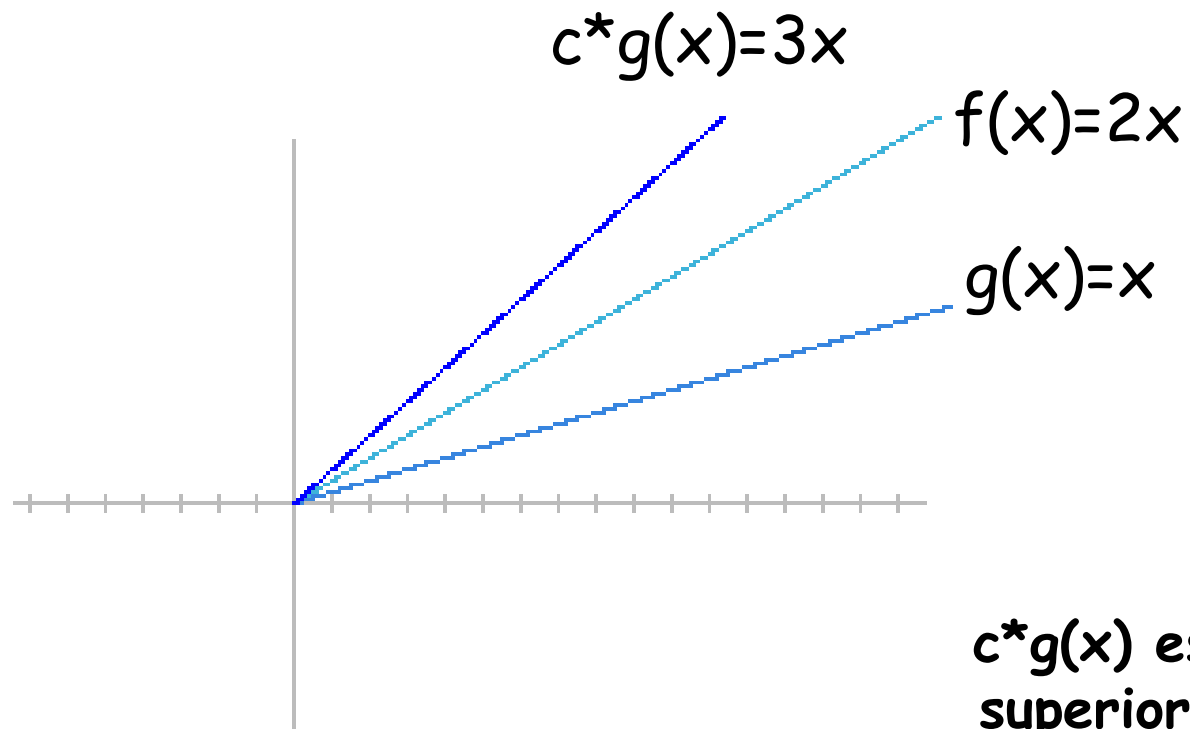
Para que una función sea cota superior de otra, debe serlo para todos los valores de $x \geq k$

Pueden existir valores de $x < k$ para los cuales no sea cota superior

Buscar una cota superior para $f(x)$ en términos de $g(x)$

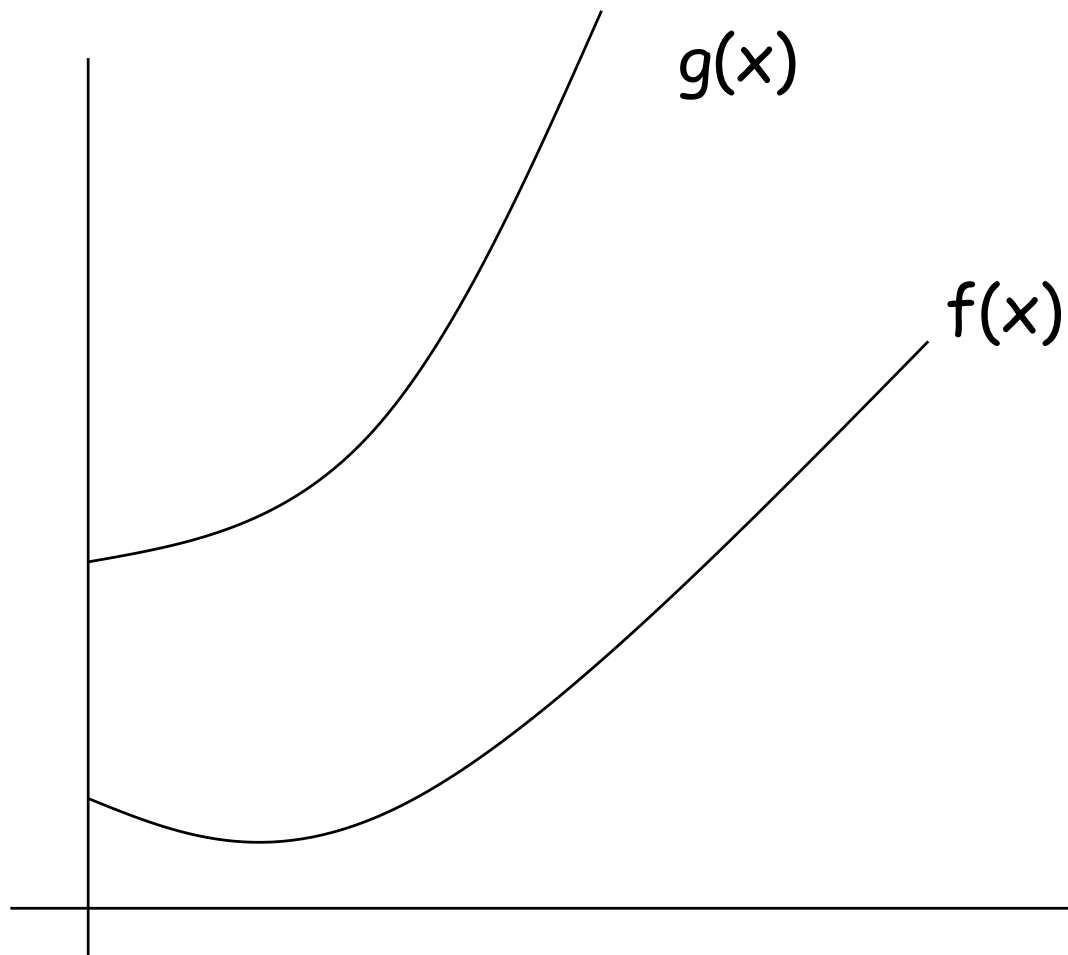


Crecimiento de funciones



$c \cdot g(x)$ es una cota superior para $f(x)$

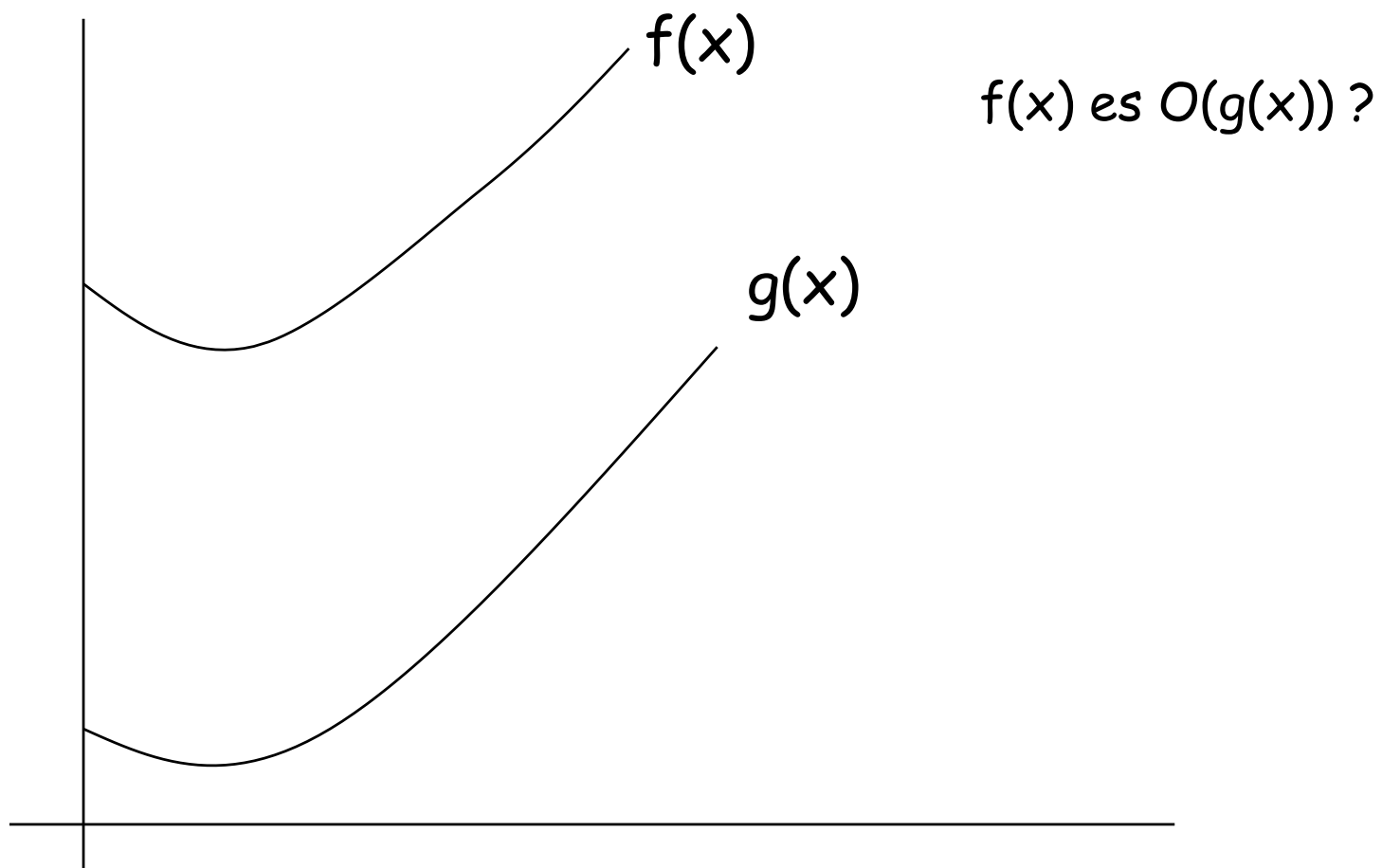
Crecimiento de funciones



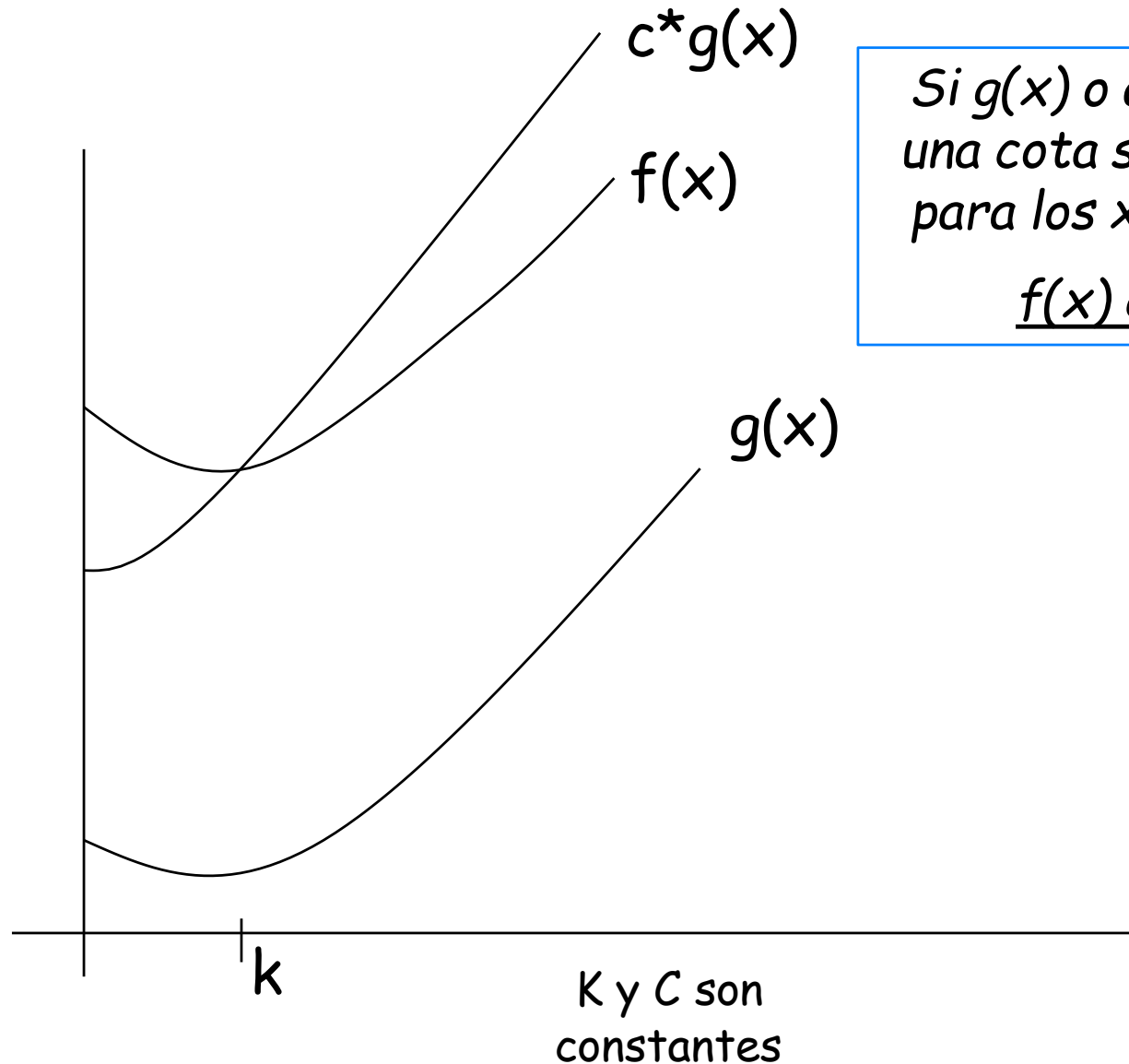
Si $g(x)$ o algún $c \cdot g(x)$ es una cota superior de $f(x)$, para los $x \geq k$, se dice que

$f(x)$ es $O(g(x))$

Crecimiento de funciones



Crecimiento de funciones



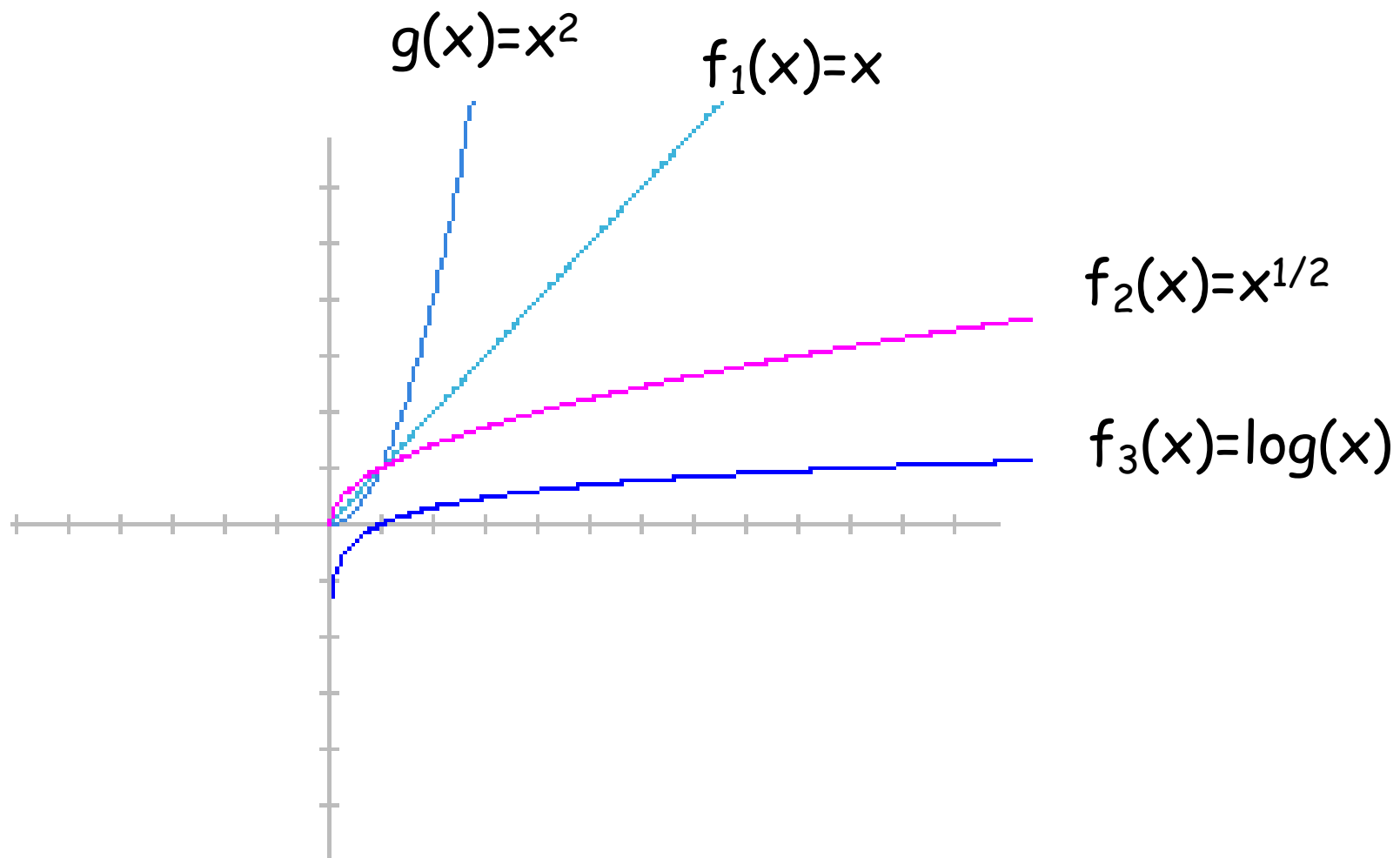
Si $g(x)$ o algún $c \cdot g(x)$ es una cota superior de $f(x)$, para los $x \geq k$, se dice que $f(x)$ es $O(g(x))$

Crecimiento de funciones

Dadas dos funciones f y g , se dice que $f(x)$ es $O(g(x))$ si existen constantes c y k tales que:

$$f(x) \leq c * g(x)$$

se cumple para todos los $x \geq k$



$g(x)$ es una cota superior de $f_1(x)$, $f_2(x)$, $f_3(x)$

Crecimiento de funciones

Formalmente la notación O representa un conjunto de funciones

$O(g(x)) = \{ f(x) \mid \text{existen constantes positivas } c \text{ y } k, \text{ tales que } 0 \leq f(x) \leq c * g(x), \text{ para todos los } x \geq k \}$

Crecimiento de funciones

$$O(x^2) = \{ x, x^{1/2}, \log(x), \dots \}$$

Crecimiento de funciones

Muestre que $7x^2 = O(x^3)$

Crecimiento de funciones

Muestre que $7x^2 = O(x^3)$

$$7x^2 \leq x^3 \quad \text{para } x \geq 7$$

Por lo tanto se cumple que

$$f(x) \leq 1 \cdot g(x)$$

$$\text{para } x \geq 7$$

$$C=1, k=7$$

Crecimiento de funciones

Es x^3 , $O(7x^2)$

Crecimiento de funciones

Es x^3 , $O(7x^2)$

$$x^3 \leq c \cdot 7x^2$$

$$x \leq 7c$$

Ya que no se cumple para todos los $x \geq k$, no es cierto que x^3 sea $O(7x^2)$

Crecimiento de funciones

Ejercicios

Demstrar que:

- x^2+2x+1 es $O(x^2)$
- $\log n$ es $O(n)$, parta sabiendo que $n < 2^n$
- $x^2+4x+17$ es $O(x^3)$ pero que x^3 no es $O(x^2+4x+17)$
- $x \log x$ es $O(x^2)$

Explique qué significa que una función sea $O(1)$

Explique qué significa que una función sea $O(1)$

Crecimiento de funciones

Cota superior asintótica

Cuando se determina que el tiempo de computo de un algoritmo es $O(g(x))$ se establece una cota superior

Dentro del análisis para obtener la cota superior, se debe considerar el peor caso, de esta forma se consigue tener una cota que no puede resultar peor

Crecimiento de funciones

Cota superior asintótica

Suponga que para un algoritmo se encontró que $T_1(n) = O(n^2)$ y para otro que $T_2(n) = O(n \lg n)$, qué representan estos resultados?

¿Qué se puede esperar en cuanto al tiempo de ejecución de los algoritmos?

¿Se puede asegurar que siempre es mejor el algoritmo 2 que el 1?

Crecimiento de funciones

Cota superior asintótica

Es una generalización decir que el tiempo de ejecución del insertion sort es $O(n^2)$, esto se puede asegurar, para el peor caso y caso promedio. Para el mejor caso se podría decir que es $O(n)$ (Aunque $O(n)$ es $O(n^2)$).

Crecimiento de funciones

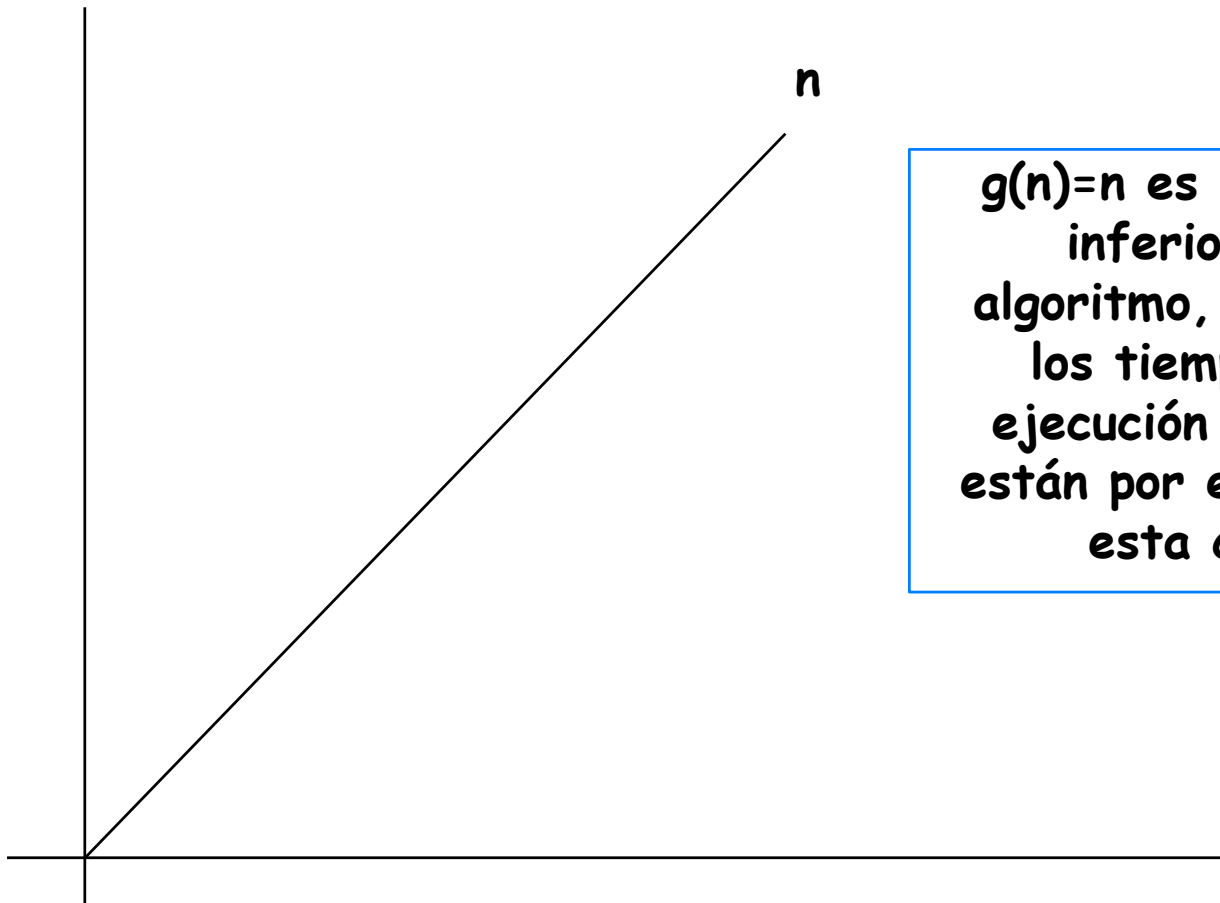
Cota inferior

$\Omega(g(x)) = \{ f(x) \mid \text{existen constantes positivas } c \text{ y } k, \text{ tales que } 0 \leq c * g(x) \leq f(x), \text{ para todos los } x \geq k \}$

Crecimiento de funciones

Cota inferior

Suponga que para un algoritmo se encontró que $T_1(n) = \Omega(n)$



$g(n)=n$ es una cota inferior del algoritmo, esto es, los tiempos de ejecución siempre están por encima de esta cota

Crecimiento de funciones

Cota inferior

Suponga que para un algoritmo se encontró que $T_1(n) = \Omega(n^2)$ y para otro que $T_2(n) = \Omega(n \lg n)$, qué representan estos resultados?

¿Qué se puede esperar en cuanto al tiempo de ejecución de los algoritmos?

Crecimiento de funciones

Notación $\Theta(f(n))$

$f(n)=\Theta(g(n))$ sii $f(n)=O(g(n))$ y $f(n)=\Omega(g(n))$

Crecimiento de funciones

Notación o

$o(g(x)) = \{ f(x) \mid \text{para toda constante positiva } c, \text{ existe una constante positiva } k, \text{ tal que } 0 \leq f(x) < c * g(x), \text{ para todos los } x \geq k \}$

Crecimiento de funciones

Notación o

La cota $2n^2 = O(n^2)$ es asintóticamente ajustada pero $2n^2 = o(n^2)$ no es cierto.

Se utiliza la notación o para denotar cota superiores que no son asintóticamente ajustadas

$2n = o(n^2)$ pero $2n^2 = o(n^2)$ no es cierto.

Crecimiento de funciones

Notación ω

$\omega(g(x)) = \{ f(x) \mid \text{para toda constante positiva } c, \text{ existe una constante positiva } k, \text{ tal que } 0 \leq c * g(x) < f(x), \text{ para todos los } x \geq k \}$

Crecimiento de funciones

Complejidades de los algoritmos

Complejidad	Terminología
$O(1)$	Complejidad constante
$O(\log n)$	Complejidad logarítmica
$O(n)$	Complejidad lineal
$O(n \log n)$	Complejidad $n \log n$
$O(n^b)$	Complejidad polinomial
$O(b^n)$	Complejidad exponencial
$O(n!)$	Complejidad factorial

Crecimiento de funciones

Complejidades de los algoritmos

Un problema que se puede resolver utilizando un algoritmo con complejidad polinómica en el peor caso se llama **tratable**. De no ser así se llama **intratable**

Un problema para el cual no existe un programa que lo pueda solucionar se llama **irresoluble**. De no ser así se llaman **resolubles**. (Halting problem - Turing)

Crecimiento de funciones

Complejidades de los algoritmos

Existen problemas que no se pueden resolver, en el peor caso en tiempo polinomial, pero que dada alguna solución, se puede comprobar que es efectivamente una solución. Estos problemas se llaman **NP** (Polinómico - No determinista)

Existe un conjunto de problemas NP que además, si se llegase a encontrar una solución en tiempo polinómico, daría solución a un conjunto de problemas relacionados. Este tipo de problema se conoce como **NP-completos**

Crecimiento de funciones

Complejidades de los algoritmos

El problema de la **satisfactibilidad** es un problema NP-completo.

Dada una asignación de valores de verdad, se puede verificar en tiempo polinómico si tal asignación satisface, o no, una fórmula proposicional. Sin embargo, no existe un algoritmo que en tiempo polinómico pueda encontrar la asignación de valores de verdad para que una fórmula cualquiera se satisfaga.

Crecimiento de funciones

Complejidades de los algoritmos

Además, si se encontrara un programa que lograra hallar la solución en tiempo polinómico, se podría solucionar un conjunto de problemas relacionados con la lógica proposicional