

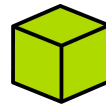
Curso de **Complejidad Algorítmica con JavaScript**

Marcelo Arias @360macky



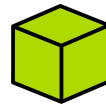
**Es momento de dar
el siguiente paso**





¿Qué aprenderás en este curso?

- Dominar el concepto de complejidad algorítmica.
- Evaluar qué tan eficiente es un algoritmo.
- Aprender a seleccionar algoritmos basados en el consumo de recursos.



¿Por qué deberías aprender análisis de algoritmos?

- Puedes crear software más eficiente a través de la selección de algoritmos.
- Es un *skill* necesario para las entrevistas de trabajo.

Proyecto del curso





Repositorio de algoritmos

Una colección de algoritmos, donde cada algoritmo cuente con su propio análisis.

Algunos algoritmos los trabajaremos en clase, otros los podrás realizar tú.

Determinaremos qué tan eficientes pueden ser estos algoritmos.

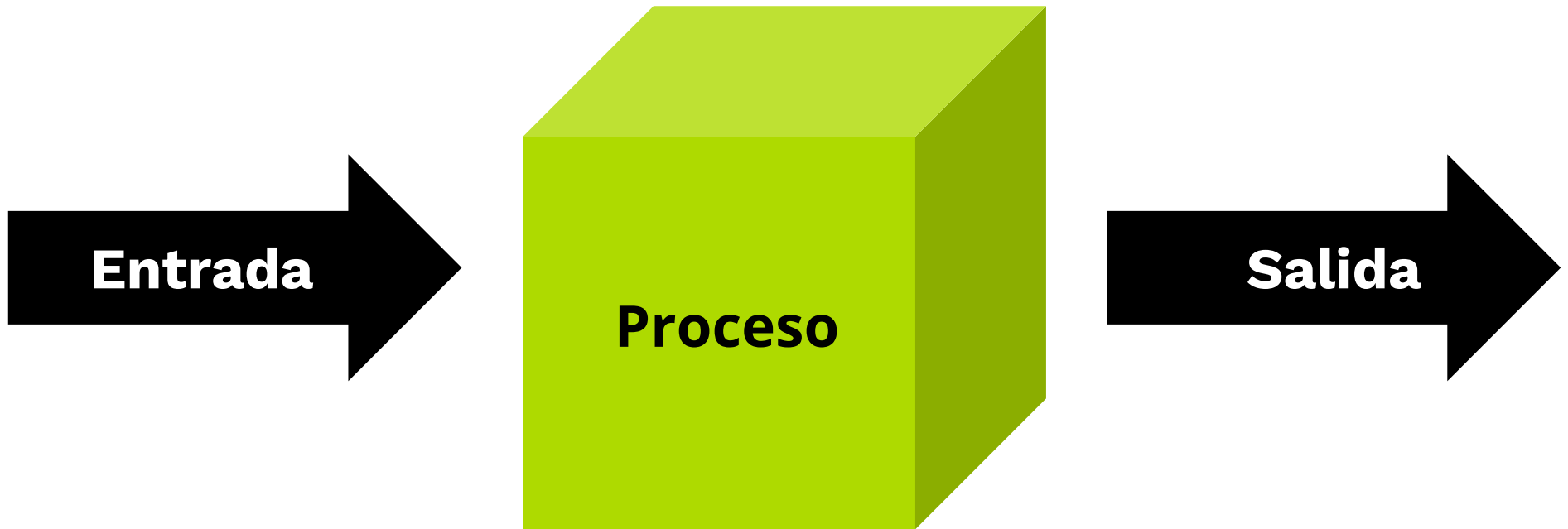
The background is a dark gray grid. In the top-left corner, there are three yellow cubes of different sizes, some partially cut off by the edge. At the bottom, there is a yellow floor with a grid pattern that recedes into the distance.

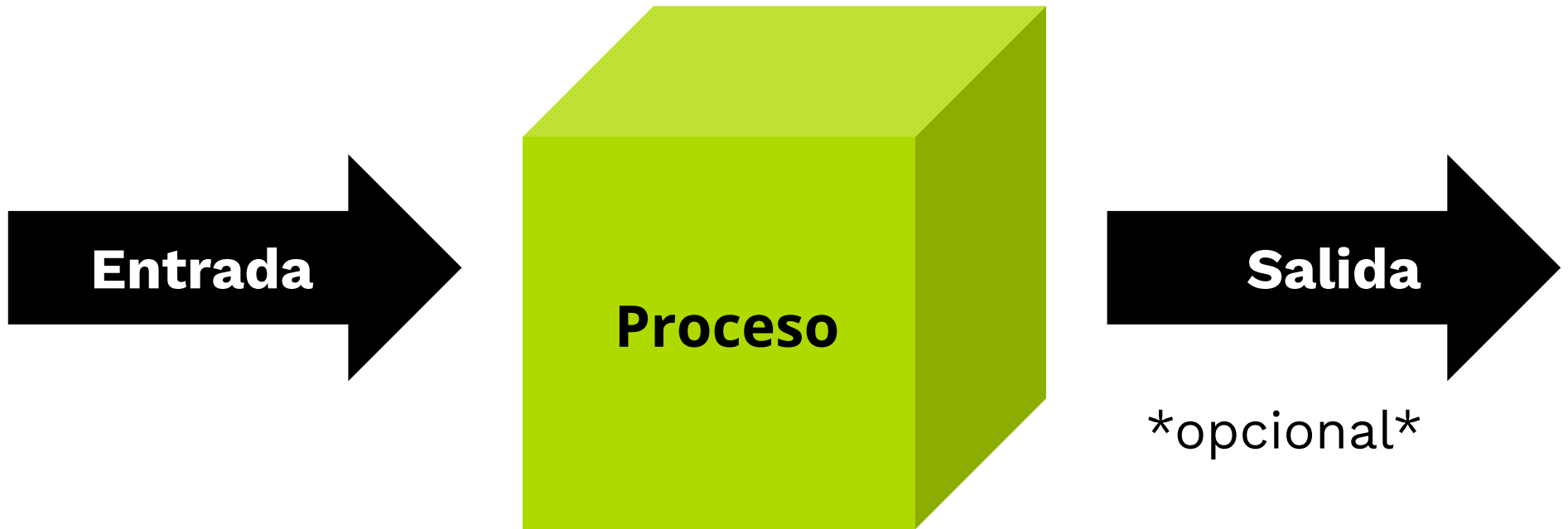
Estructura de un algoritmo

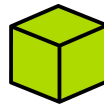
Modelo de solución de un
problema

**¿Qué es un
algoritmo?**









Algoritmo

Un algoritmo es una secuencia de instrucciones.

Los algoritmos que se analizarán tienen que contar al menos con datos de entrada (**input**).

```
function algoritmo(entrada) {  
    // Proceso  
    return salida;  
}
```

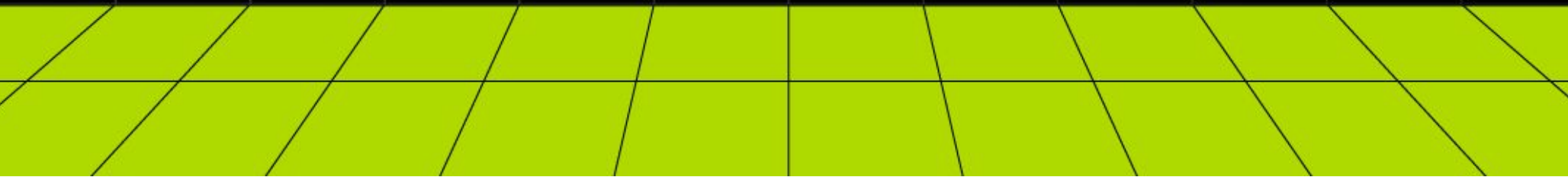
```
// Usando el algoritmo  
algoritmo();
```

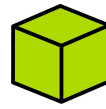
```
function algoritmo(entrada) {  
    let salida = {};  
  
    if () {  
    }  
  
    for () {  
    }  
  
    funcion_a();  
  
    return salida;  
}
```



¿Cómo elegir un buen algoritmo?

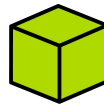
Un mismo problema,
la mejor solución



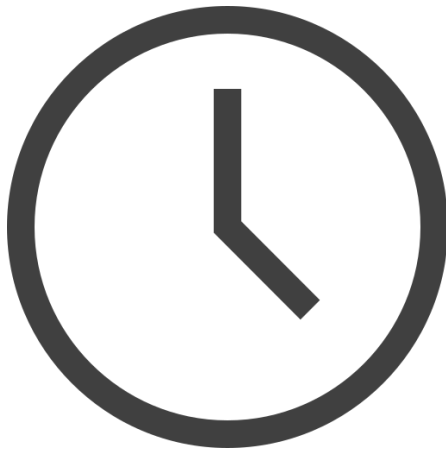


Un problema, muchas soluciones

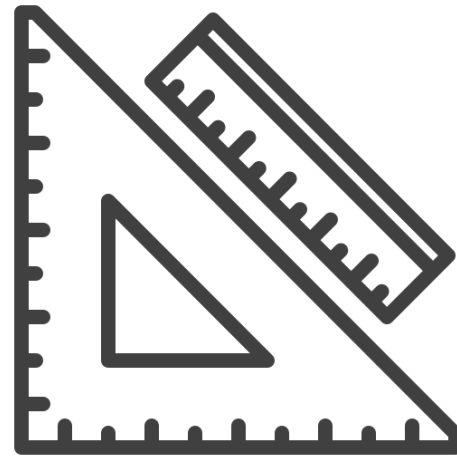
Existen muchos algoritmos que resuelven el mismo problema.



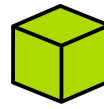
¿Qué evaluamos de un algoritmo?



Tiempo



Espacio



Tiempo de algoritmo

¿Cuánto tarda el algoritmo?

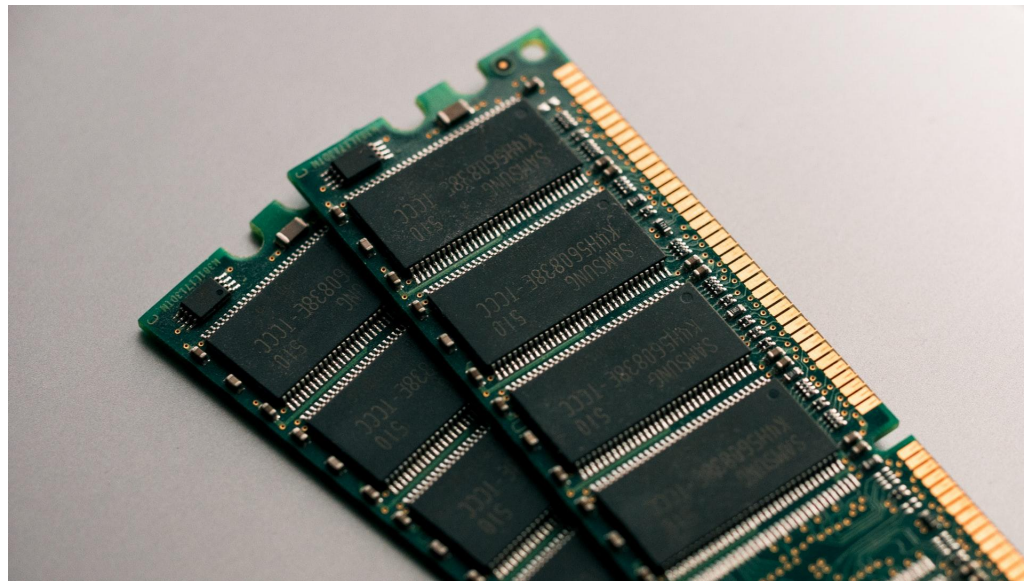


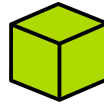
LOADING



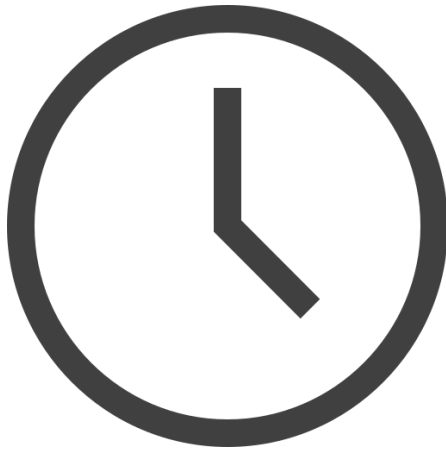
Espacio de algoritmo

¿Cuánto espacio en memoria ocupa el algoritmo?

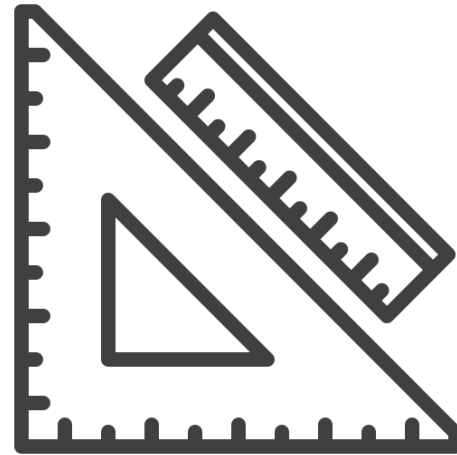




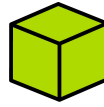
¿Existe un mejor aspecto a analizar?



Tiempo

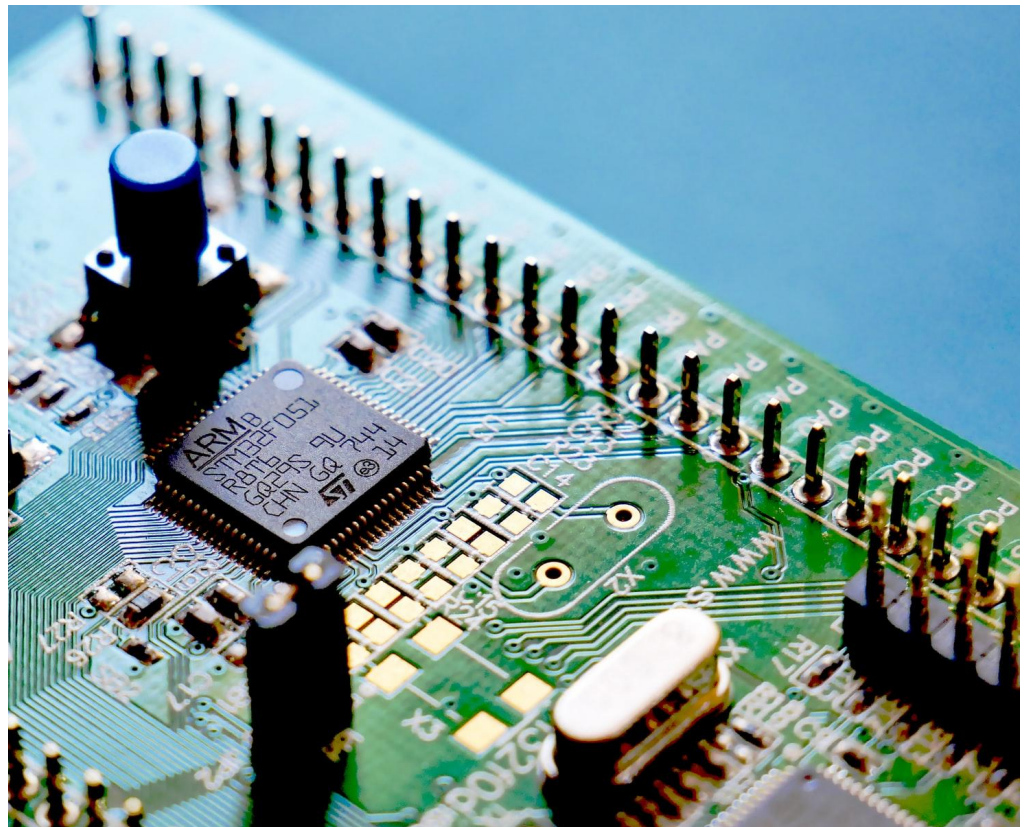


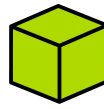
Espacio



En espacio...

En dispositivos embebidos, el buen manejo de la memoria es crítico.



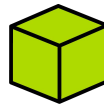


Rendimiento en JavaScript

El **tiempo** es un factor más importante en JavaScript.

The background features a dark gray grid pattern. In the top-left corner, there are three yellow cubes of varying sizes, some with black outlines. The bottom of the image shows a perspective view of a yellow floor with black grid lines.

Introducción a la complejidad algorítmica



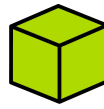
Recursos

Tiempo

Representados en milisegundos, segundos, minutos, etc.

Espacio

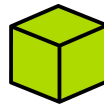
Representado en bytes, kilobytes, megabytes, etc.



Medir los recursos

Para obtener cuántos recursos utiliza un algoritmo, **solo** medimos los recursos.

Vamos a tomar cuánto tiempo y cuánto espacio ocupa un algoritmo.



Teoría de la complejidad

Estudia el consumo de recursos que un algoritmo ocupa.



La complejidad se basa en el crecimiento

En la complejidad queremos entender el crecimiento de recursos, no su tamaño.

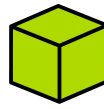
El **crecimiento de recursos** importa.

**La complejidad es
el crecimiento de
recursos**





Complejidad temporal



Situación

Necesitamos un **algoritmo rápido** que obtenga el promedio de puntos PlatzRank de una lista de X estudiantes.

Tenemos cuatro algoritmos: **Experto, Astronauta, Nova y Visión.**

¿Cuál elegimos?

**¡Empecemos
a medir!**



El algoritmo «**Astronauta**» obtiene el promedio de PlatzRank de 10 estudiantes, en 10 segundos.

El algoritmo «**Astronauta**» obtiene el promedio de PlatzRank de 20 estudiantes, en 20 segundos.

El algoritmo «**Astronauta**» obtiene el promedio de PlatzRank de 30 estudiantes, en 30 segundos.

El algoritmo «**Experto**» obtiene el promedio de PlatzRank de 10 estudiantes, en 20 segundos.

El algoritmo «**Experto**» obtiene el promedio de PlatzRank de 20 estudiantes, en 20 segundos.

El algoritmo «**Experto**» obtiene el promedio de PlatzRank de 30 estudiantes, en 20 segundos.

Algoritmos

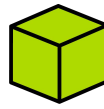
	Astronauta	Experto
10 estudiantes	10 segundos	20 segundos
20 estudiantes	20 segundos	20 segundos
30 estudiantes	30 segundos	20 segundos
40 estudiantes	40 segundos	20 segundos

Datos de entrada

Algoritmos

	Astronauta	Experto
10 estudiantes	10 segundos	20 segundos
20 estudiantes	El tiempo aumenta dependiendo de los datos de entrada.	El tiempo es constante sin depender de los datos de entrada.
30 estudiantes		
40 estudiantes		


Datos de entrada



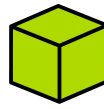
Complejidad temporal

No es sobre cuántos segundos más o menos un algoritmo se demore en ejecutarse.

Si no **cómo aumenta** el tiempo.



Complejidad temporal en práctica




Interfaz performance

El método **performance.now()** nos ayuda a **medir el tiempo** entre dos líneas de código en **milisegundos**.

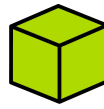
Existe otro método: **console.time()**, pero no es tan preciso como **performance.now()**.

Visualización de complejidad temporal





Complejidad espacial



Situación

Necesitamos un **algoritmo, que ocupe poco espacio**, que obtenga el promedio de puntos Platzirank de una lista de X estudiantes.

Tenemos cuatro algoritmos: **Experto, Astronauta, Nova y Visión.**

¿Cuál elegimos?

El algoritmo «**Visión**» obtiene el promedio de PlatziRank de 10 estudiantes, ocupando 10 kilobytes.

El algoritmo «**Visión**» obtiene el promedio de PlatziRank de 20 estudiantes, ocupando 20 kilobytes.

El algoritmo «**Visión**» obtiene el promedio de PlatziRank de 30 estudiantes, ocupando 30 kilobytes.

El algoritmo «**Nova**» obtiene el promedio de PlatziRank de 10 estudiantes, ocupando 10 kilobytes.

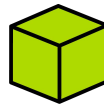
El algoritmo «**Nova**» obtiene el promedio de PlatziRank de 20 estudiantes, ocupando 100 kilobytes.

El algoritmo «**Nova**» obtiene el promedio de PlatziRank de 30 estudiantes, ocupando 1000 kilobytes.

Algoritmos

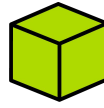
	Visión	Nova
10 estudiantes	10 kb	10 kb
20 estudiantes	20 kb	100 kb
30 estudiantes	30 kb	1000 kb
40 estudiantes	40 kb	10000 kb

Datos de entrada



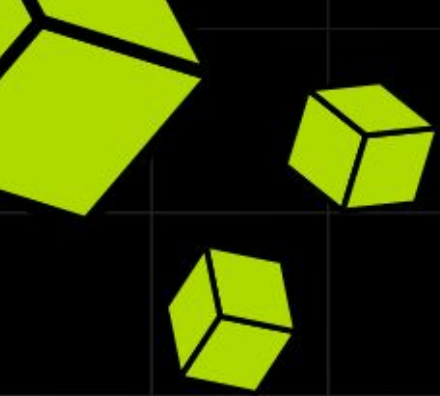
Memoria

Cuando trabajamos con grandes cantidades de datos tendremos que almacenar información en otras partes.



Espacio auxiliar

La complejidad espacial incluye el **espacio auxiliar** y el espacio ocupado por los datos de entrada.



Complejidad espacial en práctica

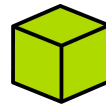


El estado de la complejidad



**¿La complejidad
solo mide espacio
y tiempo?**





Otras complejidades existentes

- Accesos a memoria.
- Procesos paralelos.
- Comparaciones.
- Entre otras más.



Complejidad en el futuro

Si descubrimos interesante optimizar el uso de un recurso en computación, allí tendremos un nuevo campo de estudio de complejidad.

**¿Cómo simplificamos
la complejidad?**



The background features a dark gray grid pattern. In the top-left corner, there are three yellow cubes of varying sizes, some with black outlines. The bottom of the image shows a perspective view of a yellow floor with black grid lines.

Introducción al análisis asintótico

Complejidad

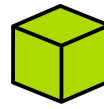
10	estudiantes	=	10	segundos
20	estudiantes	=	20	segundos
30	estudiantes	=	30	segundos
40	estudiantes	=	40	segundos

**Datos de
entrada
(input)**

Complejidad

10	cursos =	20	kilobytes
20	cursos =	40	kilobytes
30	cursos =	60	kilobytes
40	cursos =	80	kilobytes

**Datos de
entrada
(input)**



En la vida real...

Complejidad

10	estudiantes	=	14.143 milisegundos
20	estudiantes	=	25.951 milisegundos
30	estudiantes	=	32.457 milisegundos
40	estudiantes	=	41.245 milisegundos

**Datos de
entrada
(input)**

**Necesitamos
simplificar**





Análisis asintótico

Es un método para describir el comportamiento limitante de una función.

GeoGebra

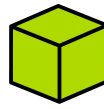


Notación Big-O

Cómo podemos medir la eficiencia de un algoritmo.

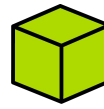
**¿Por qué necesitamos
una notación?**





¿Qué buscamos con Big-O?

Buscamos descubrir una **función** (constante, lineal, polinomial, logarítmica, y exponencial) que sea **mayor o igual** que la complejidad de un algoritmo.

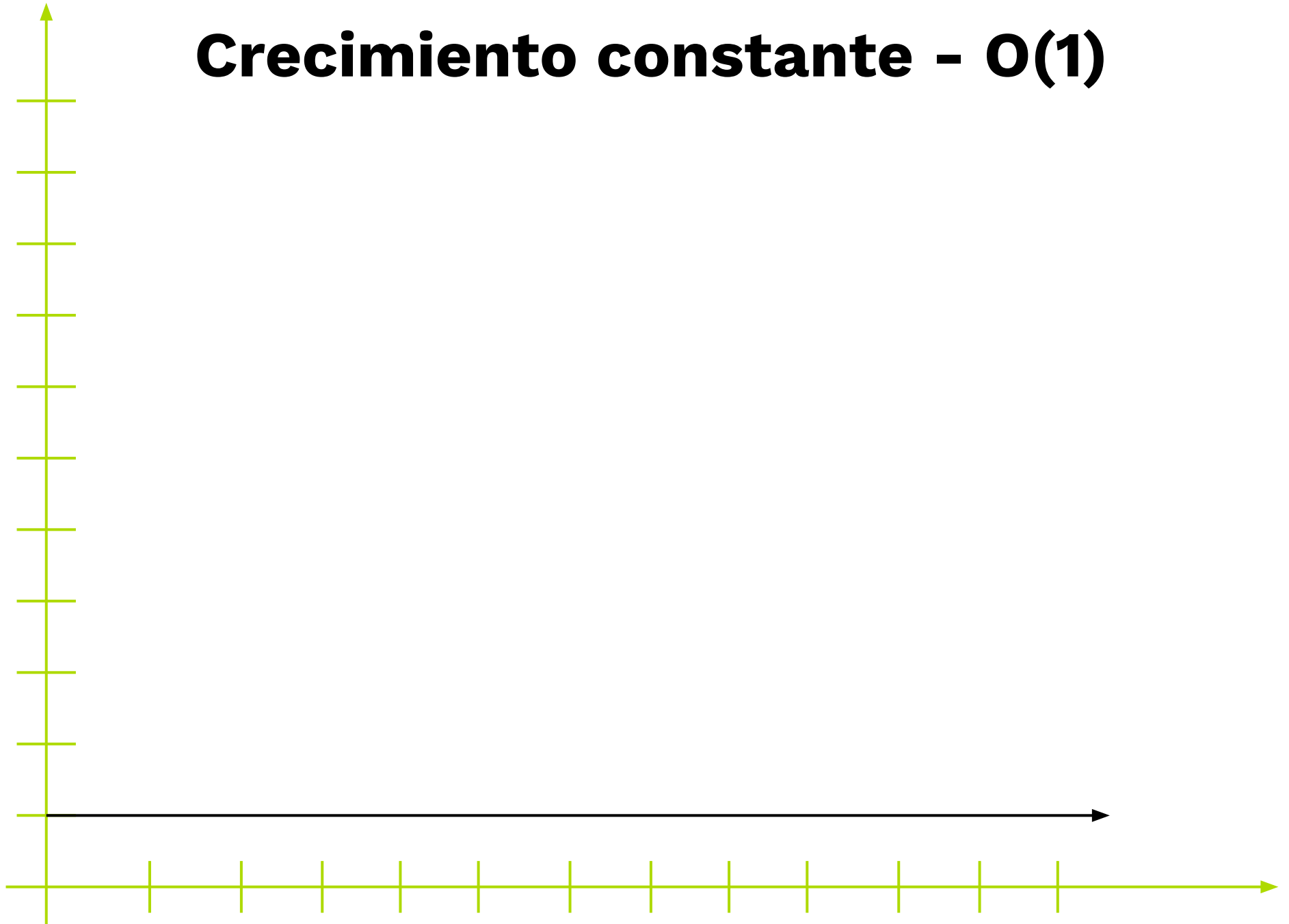


Clases de Big-O

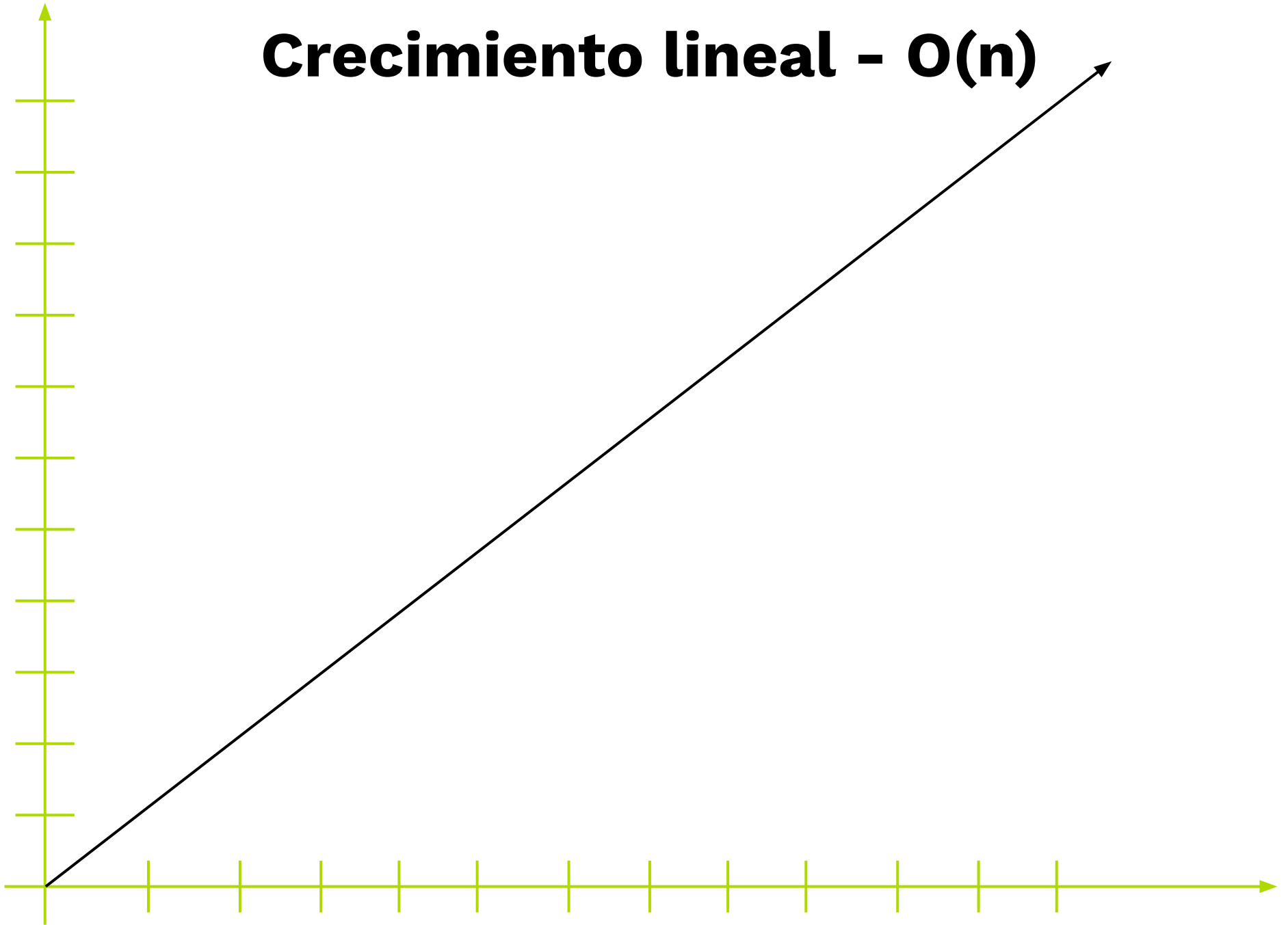
Clase	Crecimiento
$O(1)$	Constante
$O(\log n)$	Logarítmico
$O(n)$	Lineal
$O(n^2)$	Cuadrático
$O(2^n)$	Exponencial
$O(n!)$	Factorial

GeoGebra

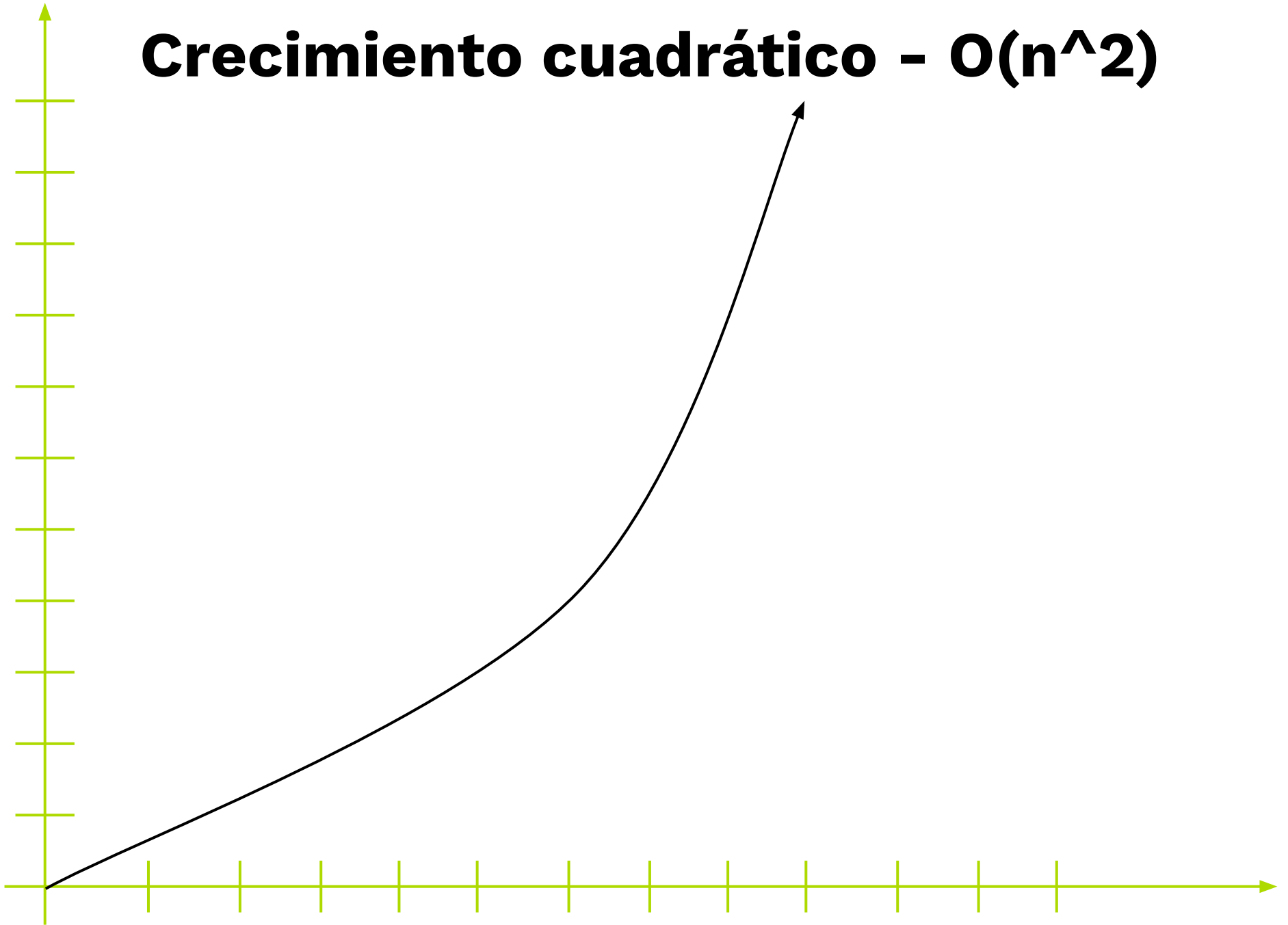
Crecimiento constante - $O(1)$



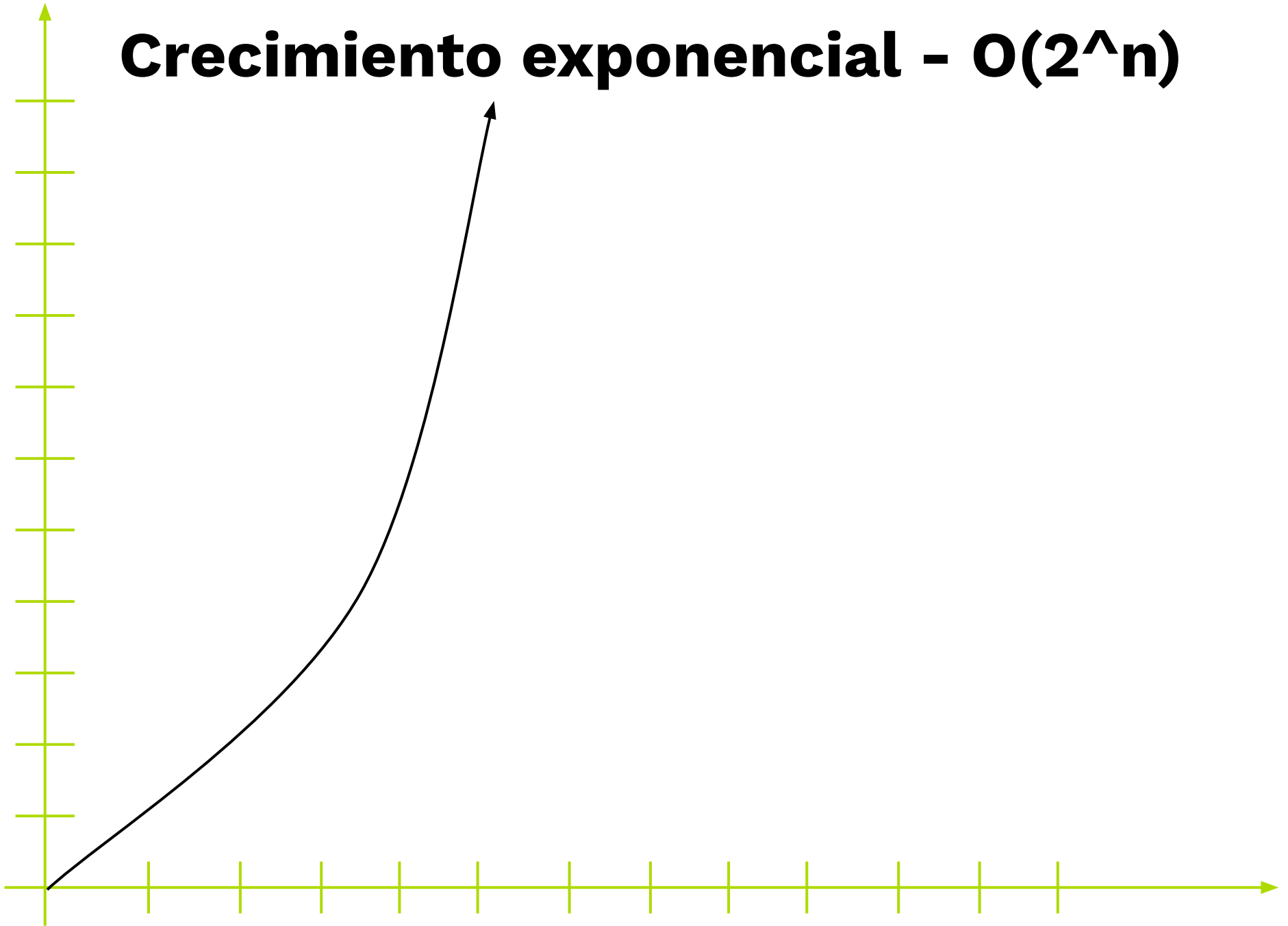
Crecimiento lineal - $O(n)$



Crecimiento cuadrático - $O(n^2)$



Crecimiento exponencial - $O(2^n)$





Mayank Joshi

@dermayank



Big-O notation summarised using emojis

$O(1) = O(\text{😄})$

$O(\log n) = O(\text{😊})$

$O(n) = O(\text{😐})$

$O(n^2) = O(\text{😞})$

$O(2^n) = O(\text{😓})$

$O(n!) = O(\text{😭})$

5:56 AM · May 24, 2021 · Twitter for Android

621 Retweets **37** Quote Tweets **2,559** Likes



The background features a dark gray grid pattern. In the top-left corner, there are three yellow cubes of varying sizes, some with black outlines. The bottom of the image shows a perspective view of a yellow floor with black grid lines.

Cálculo de la notación Big-O

Notación Big-O en complejidad temporal




```
let bar = 'test' //  $O(1)$ 
```

```
if () {} //  $O(1)$ 
```

```
for () {} //  $O(n)$ 
```

```
while () {} //  $O(n)$ 
```

```
for () { for () {} } //  $O(n^2)$ 
```

Notación Big-O en complejidad espacial



```
let bar = 'test' // 0(1)
```

```
if () {} // 0(1)
```

```
for () {} // 0(1)
```

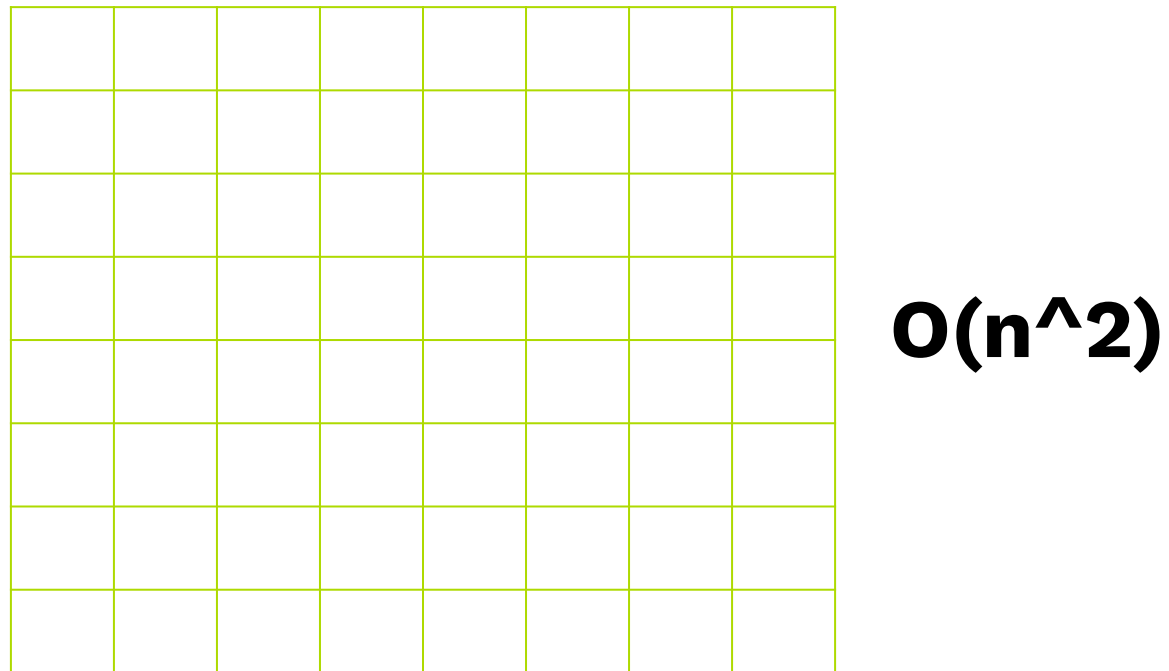
```
let resultado = [1, 2, ..., n] // 0(n)
```

```
let dimensional = ...  
[[2, 4], [6, 8], [10, 12]] // 0(n^2)
```

Si necesitamos crear un arreglo de **N** elementos:



Si necesitamos crear un arreglo de **N x N** elementos:



Simplificar la notación



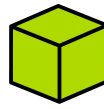


Simplificando la notación

$$O(2n) \longrightarrow O(n)$$

$$O(50) \longrightarrow O(1)$$

$$O(n^2 + 50) \longrightarrow O(n^2)$$



El crecimiento importa

La complejidad de un algoritmo nace de cuántos recursos utiliza el algoritmo al ejecutarse.

La notación Big-O solo se enfoca en el **crecimiento**.



Evaluación de complejidad temporal con Big-O

Búsqueda lineal y algoritmos
de ordenamiento





Evaluación de complejidad espacial con Big-O

Búsqueda lineal y algoritmos
de ordenamiento





Evaluación de complejidad temporal con Big-O

Algoritmos de ordenamiento





Evaluación de complejidad temporal con Big-O

Problema de payloads





Evaluación de complejidad espacial con Big-O

Algoritmos de ordenamiento





Evaluación de complejidad espacial con Big-O

Problema de payloads



The background features a dark gray grid pattern. In the top-left corner, there are three yellow cubes of varying sizes, some of which are partially cut off by the edge of the frame. At the bottom of the image, there is a horizontal band of yellow squares, also in a grid pattern, which appears to represent a floor or a base.

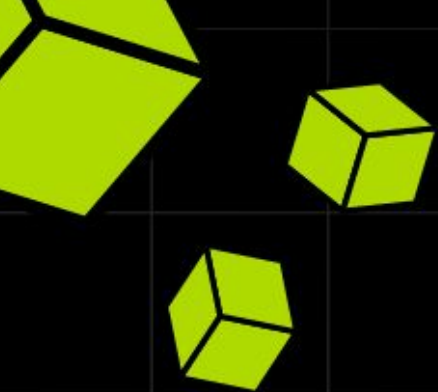
Recomendaciones para la evaluación de algoritmos

**¿El crecimiento
siempre importa?**



**¿Cómo usar
correctamente el
análisis asintótico?**





¡Felicitaciones!

Curso de **Complejidad Algorítmica con JavaScript**

Marcelo Arias @360macky

