**Análisis de Complejidad Algorítmica**

Objetivo:  
 Poder medir la complejidad algorítmica de un programa existente

Poder realizar un algoritmo en función a una complejidad determinada

Es el proceso de identificar el **tiempo**, **espacio** u **otro recurso** que es necesario para ejecutar el algoritmo

**OBS**: En computación no solamente manejamos el tiempo (a través del CPU) y el espacio (a través de la memoria), también hay otros recursos como la energía y la comunicación.

Entonces cuando hacemos un análisis de algoritmo necesitamos ver cual de estos factores es el mas importante en nuestro algoritmo.

Nos centraremos en este caso para la medición del tiempo

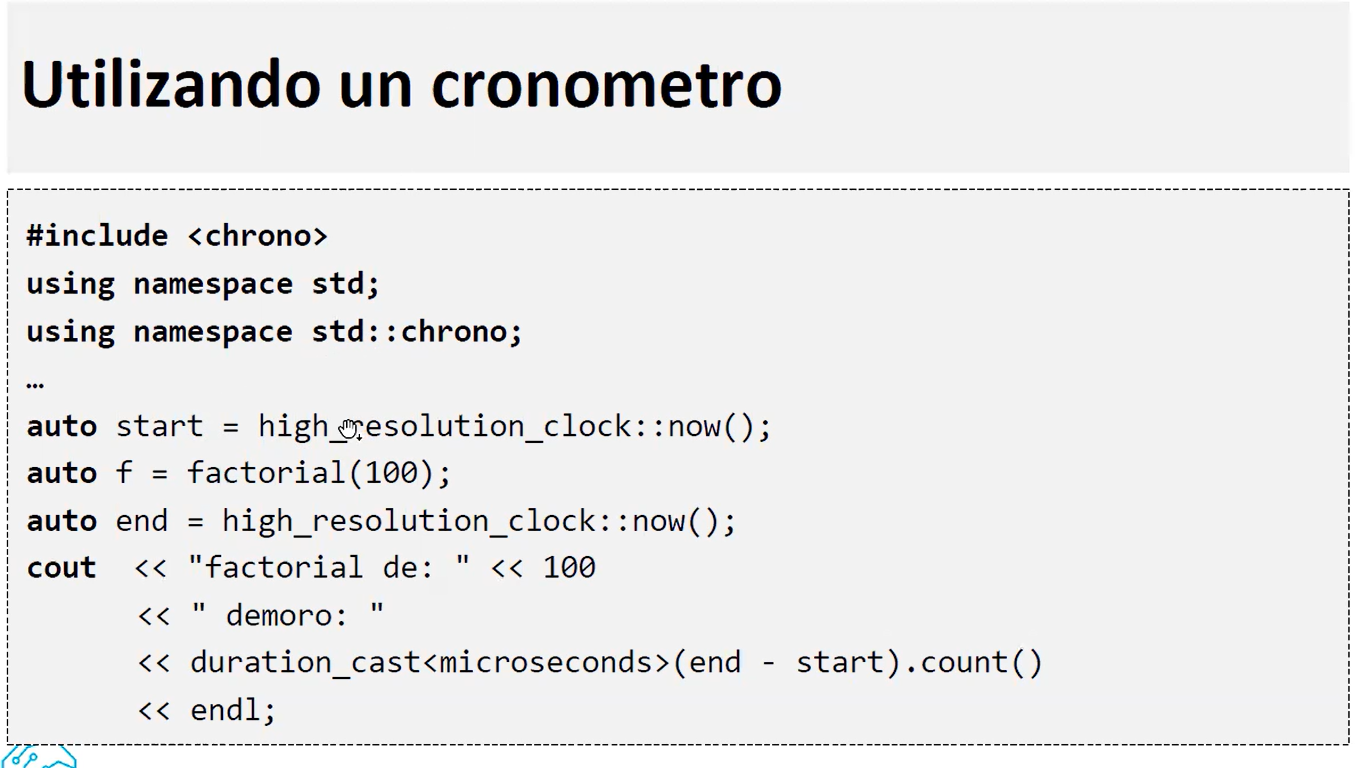
**COMPLEJIDAD DE TIEMPO:**

**¿Como se identificar el tiempo necesario?**

**Existen tres sistemas de medición:**

* Medir el tiempo, utilizando un cronometro.
* Contando el número de instrucciones. (Cada instrucción tiene un tiempo diferente)
* Evaluando el orden de crecimiento.

**Medir el tiempo, utilizando un cronometro:**

Si queremos evaluar esta función **factorial**, colocamos un cronometro al inicio y al final  
Y vamos almacenando los tiempos, y lo que se busca es automatizar este proceso donde probamos bajo diferentes parámetros el uso de esta función. Y basado en eso comienzo a armar una grafica de tiempo, y esa grafica nos va dar una línea constante, creciente, exponencial, etc., y basado en eso sacare su nivel de complejidad.

**¿De qué depende?**

* Implementación (lenguaje, instrucccion)
* Computadora (Hardware y Software)
* Instrucción
* Aplicaciones de ejecución
* Algoritmo

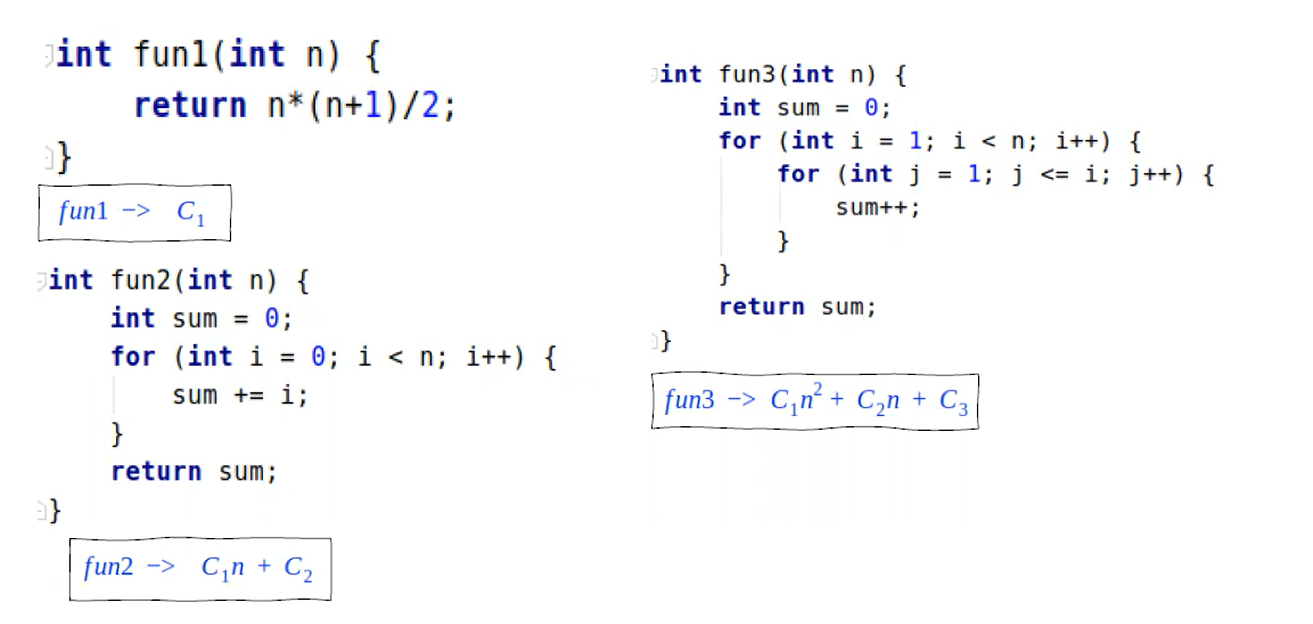
OBS: Este sistema de modelo de evaluación manual es muy bueno, pero si lo complementas con **estadística**

**Ventajas:**

* Es la forma mas sencilla para sistemas complejos.
* Se ajusta a modelos probabilísticos o estadísticos.

**Desventajas**:

* Hay muchos factores que afectan la medición, ejm. La PC
* Mediciones bajo el mismo set de datos pueden variar significativamente. No es muy precisa si lo haces con poca información, debes estar seteada por muchos valores

**Contando el número de instrucciones:**Es el uso de la función de la función de tiempo

En el primer caso, solamente se está evaluando una línea de instrucción, y es constante. Por lo tanto, esta función es constante. 🡪 fn1 -> C1 (Ecuación constante)

En el segundo caso, primero se ve que la instrucción C1 al estar en un bucle “for” se va ejecutar “n” veces es por eso que se tiene C1\*n, mientras que el C2 solamente se ejecuta una vez es constante. 🡪 fn2 -> C1\*n + C2 (Ecuación lineal)

En el tercer caso, el C1 es la suma del bucle for anidado que se ejecuta n\*n veces, y el C2 se ejecuta “n veces” que es el bucle for que esta afectado por el de arriba. Y las instrucciones de afuera son C constantes. 🡪 fn3 -> C1\*n^2 + C2\*n + C3 (Ecuación cuadrática)

Se asume que todas las operaciones en un programa **consumen el mismo tiempo** y que **ese tiempo es constante,** por ejemplo, las siguientes operaciones tienen una duración similar:

* Operaciones matemáticas
* Comparaciones
* Asignaciones
* Accesos a memoria

Se debe desarrollar una **función matemática** que evalúa la eficiencia basada en el **numero de instrucciones.**

**Ventajas:**

* Es **independiente** de **otros factores extremos**.
* Se obtiene un **único valor** de eficiencia para un mismo set de datos.

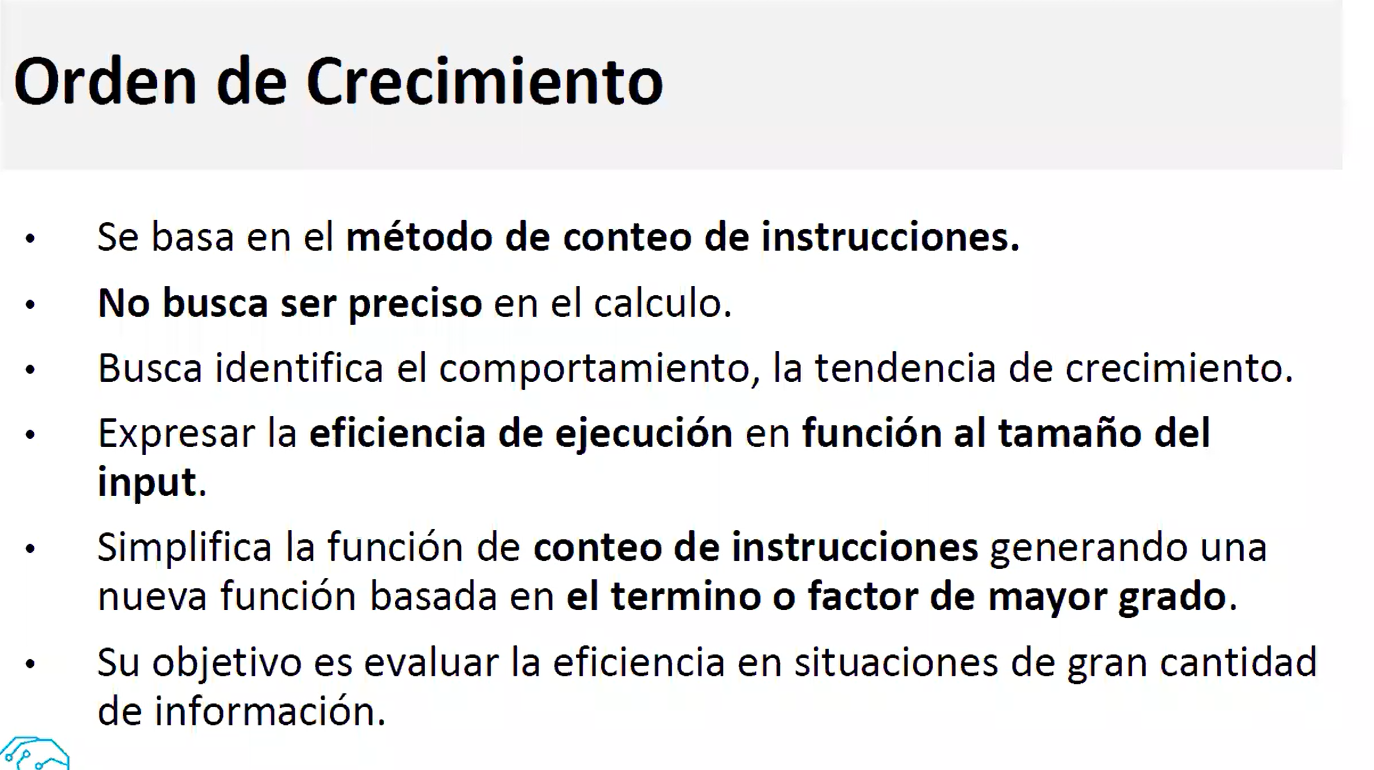
Desventajas:

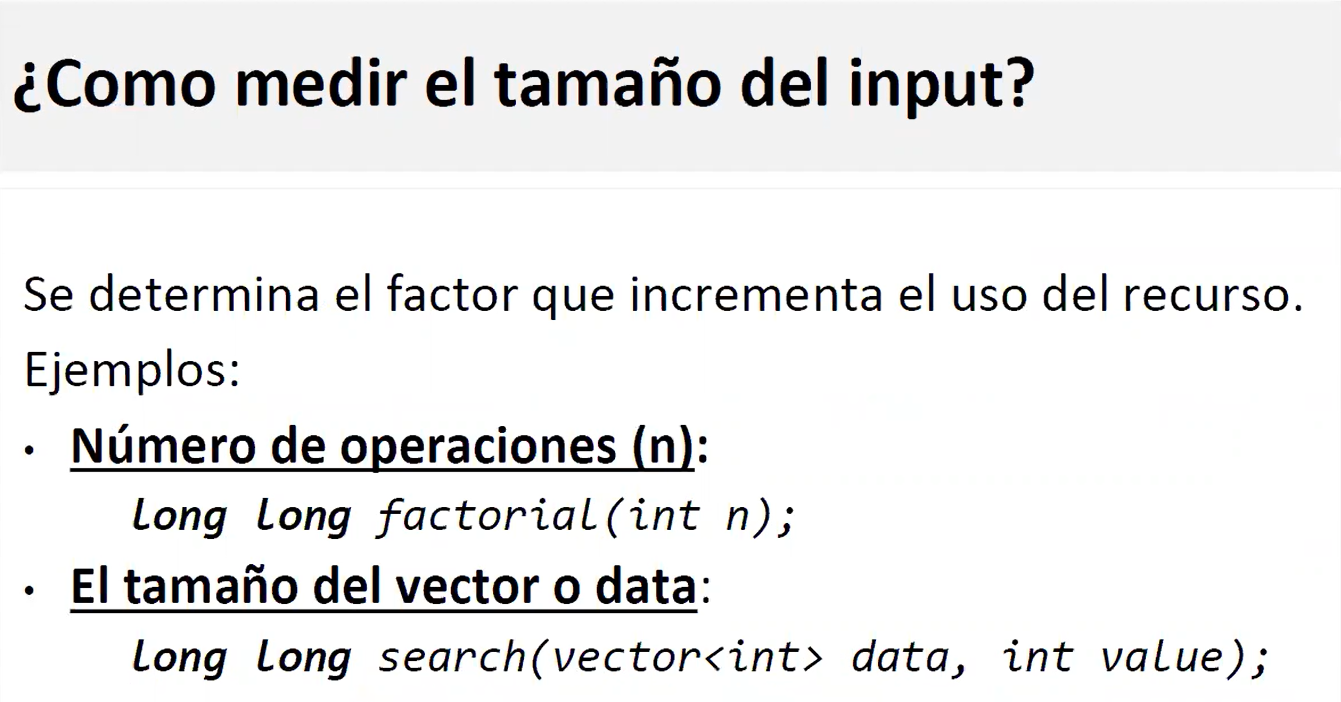
* La **ecuación** que se genera podría ser **singular**
* Puede **dificultar** la **comparación de eficiencias**.

**Orden de Crecimiento:**

Este método no es preciso, pero lo que busca es la tendencia, hacia donde se va mover nuestro algoritmo

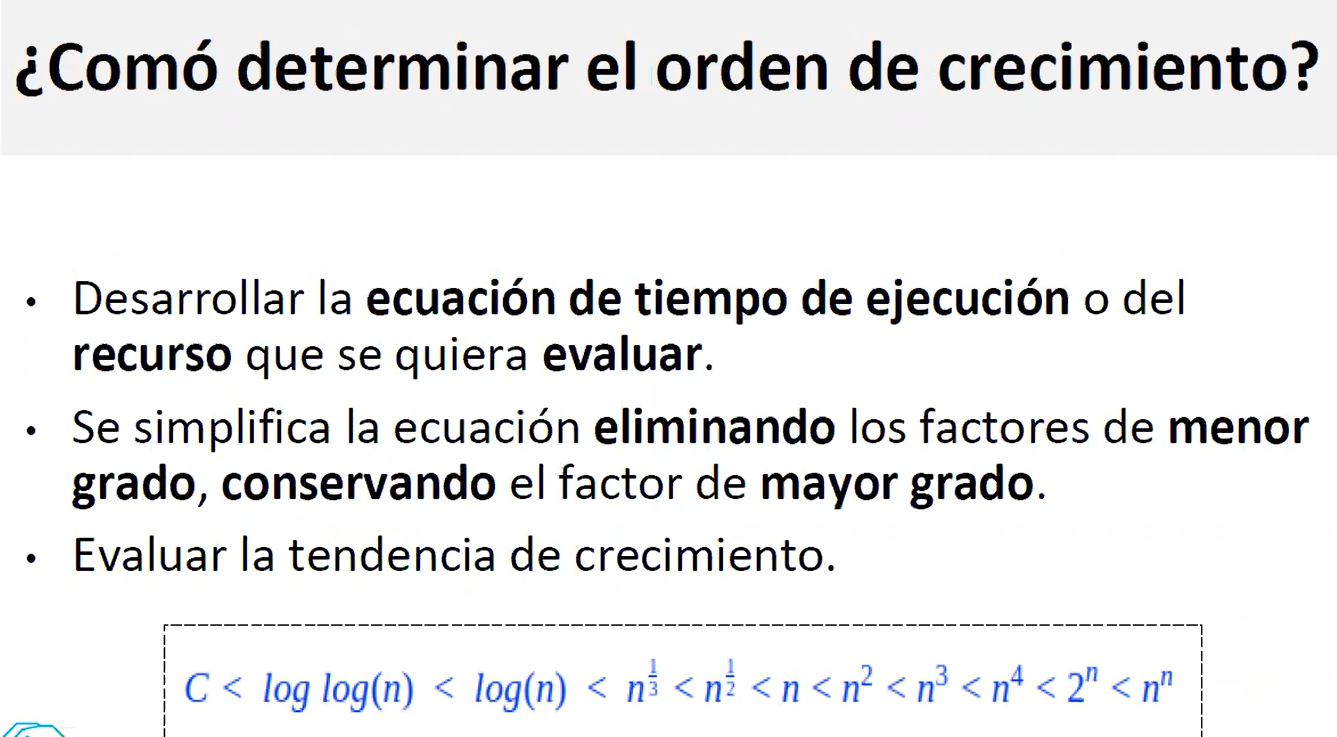
Simplifica la ecuación hallada de igual manera en el anterior método, agarrando el de mayor grado. Aquí utiliza el análisis asintótico, lo que hace es trabajar con el termino de mayor grado para evaluarla. Su objetivo es evaluar la eficiencia para gran cantidad de información, en poca información no es muy eficiente.

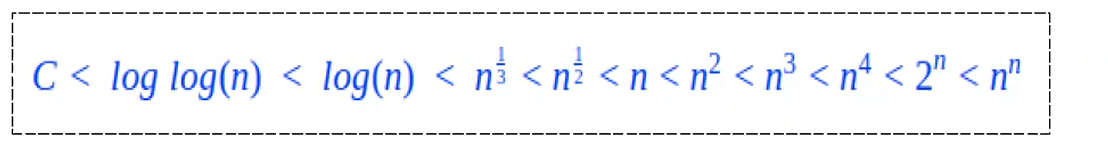




Se determina desarrollando la ecuación, y se simplifica conservando el factor de mayor grado, y solamente se toma el factor, SIN LA CONSTANTE. Ejm.

fun3 -> C1\*n^2 + C2\*n + C3 🡪 Agarramos el de mayor grado C1\*n^2

Y le quito la constante, quedándome: “n^2” 🡪 El orden de crecimiento de este es n^2

Ya una vez teniendo mi orden de crecimiento n^2, hago la comparación:

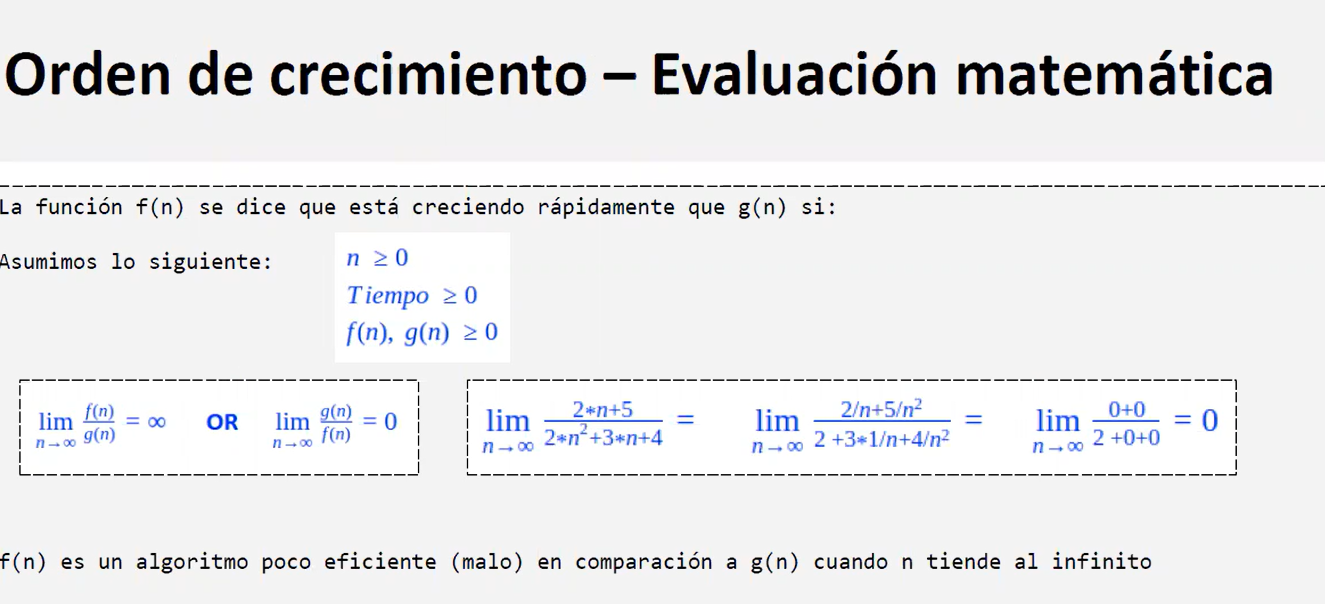
Si mi algoritmo me salió constante C es recontra eficiente, y si me salió n^n es totalmente ineficiente.

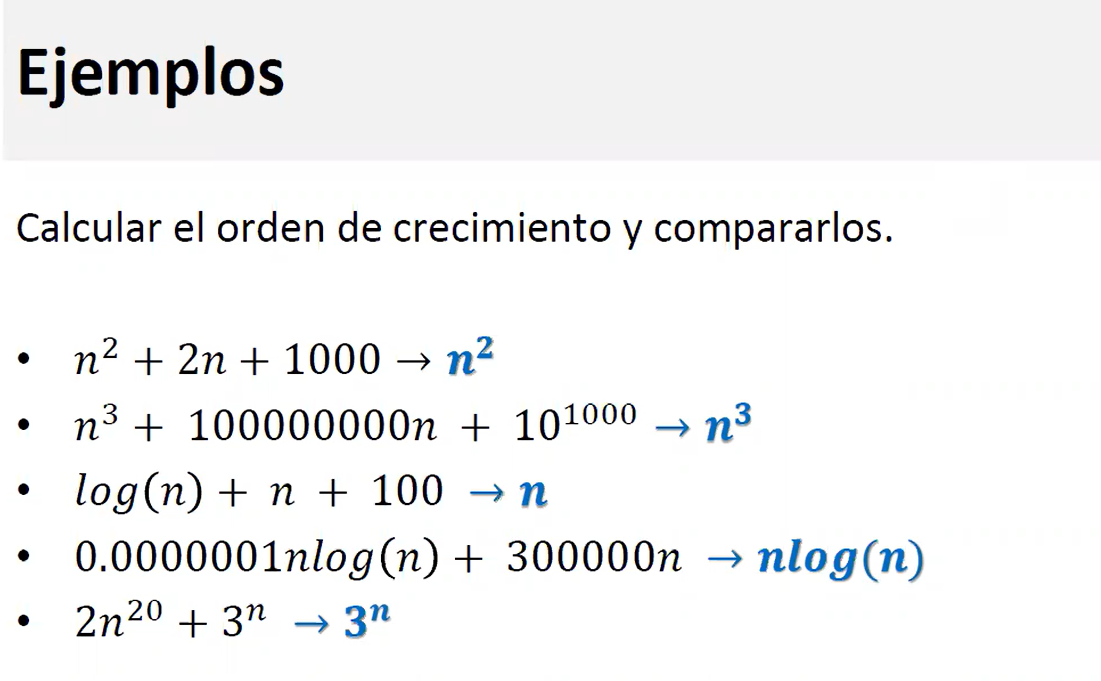
Nosotros trabajaremos solamente los polinómicos, los no polinómicos 2^n o n^n, no lo trabajaremos

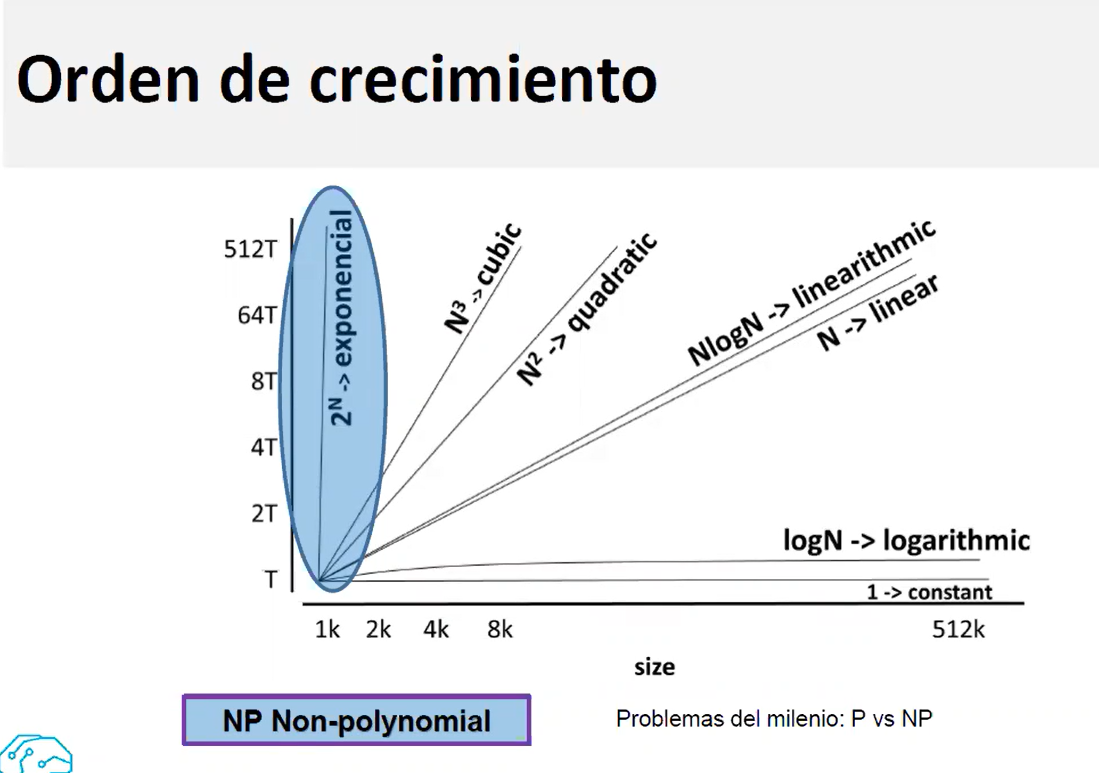
Ventajas:

* Facilita la comparación entre algoritmos.
* Se obtiene un único valor para un mismo set de datos

Desventajas:

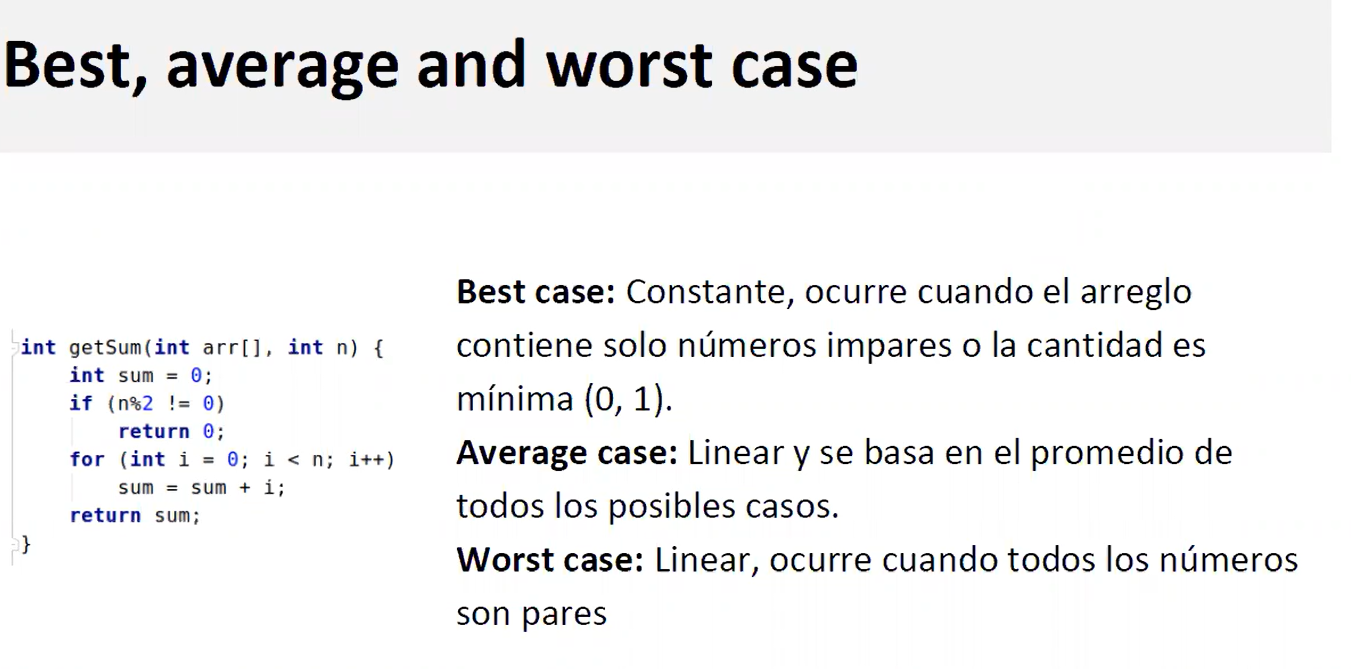
* Se **pierde precisión** por la simplificación que en casos prácticos podría ser muy útiles.





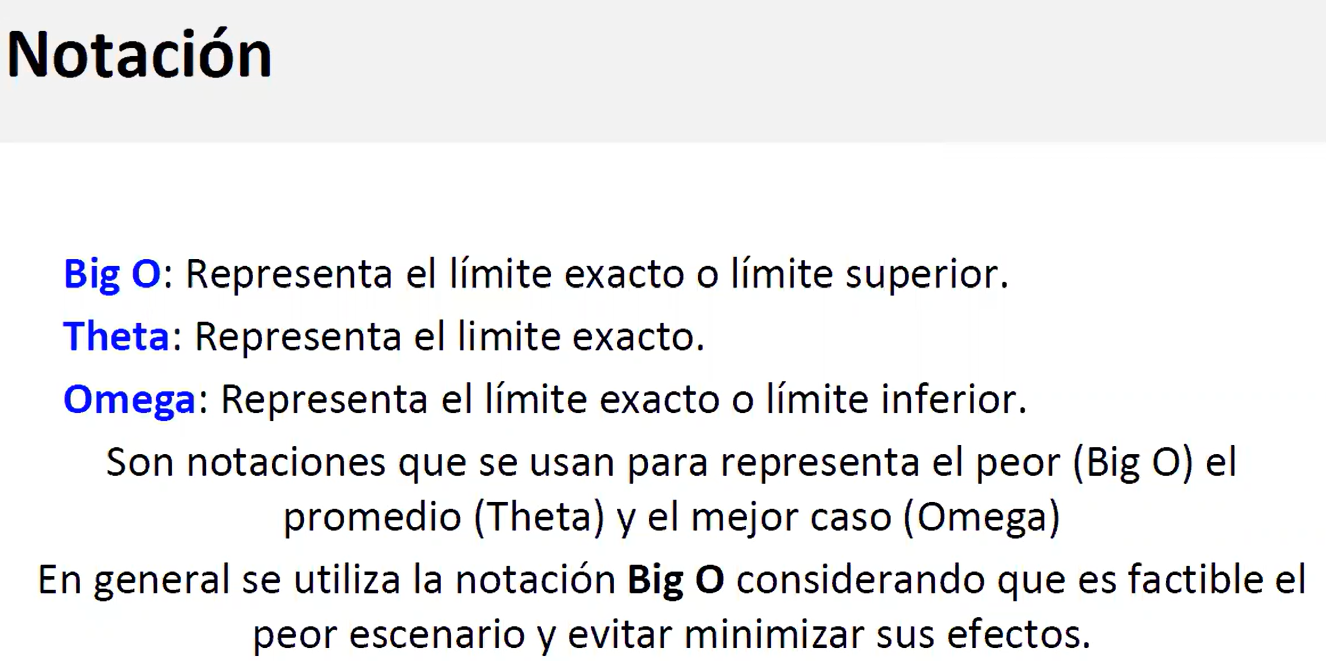
Problemas del milenio: Se busca ver si es posible convertir de un NP a P, porque si yo tuviera la posibilidad de convertir un NP a P, se simplificaría mucho el programa.

**Esa evalucion de complejidad algorítmica a traves del orden de crecimiento lo puedo evaluar en tres casos:**

****

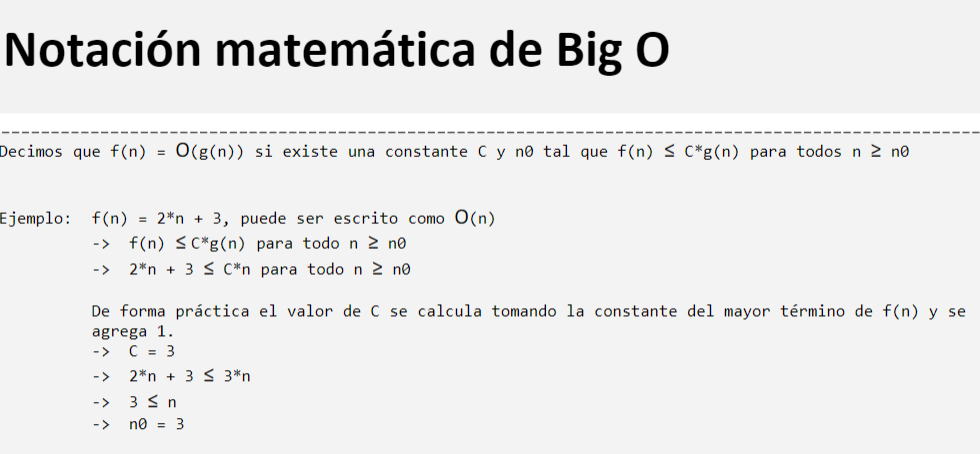
Nosotros trabajaremos el PEOR CASO:

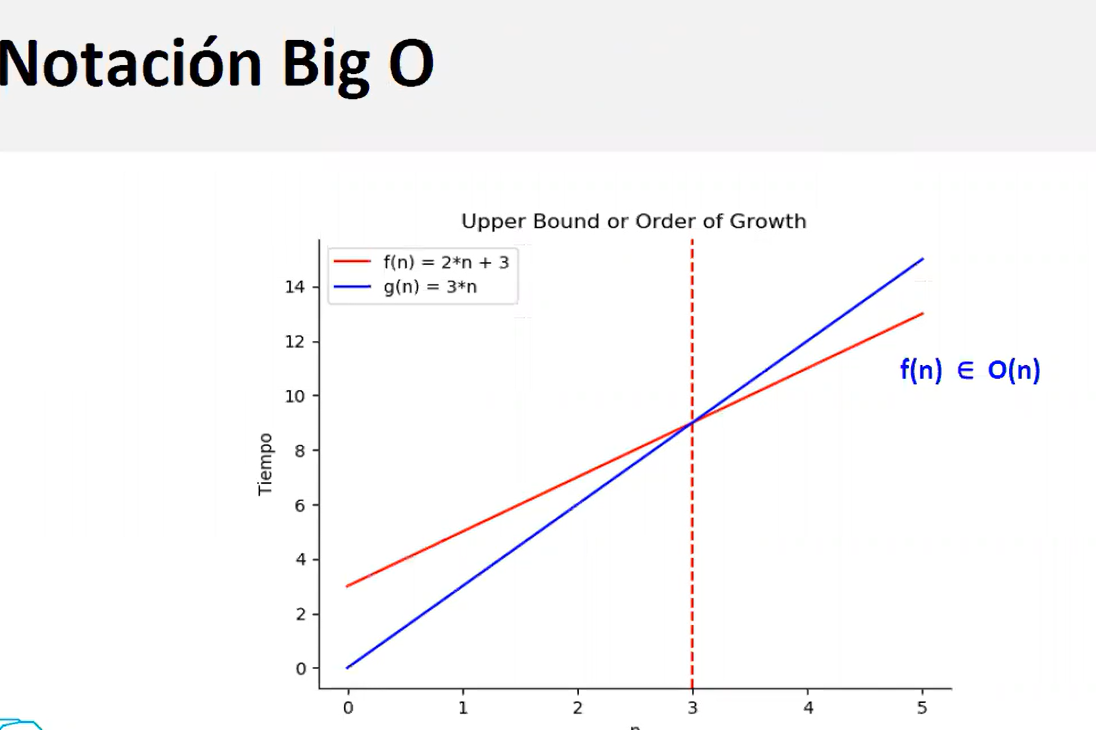
Y para cada caso existe algún tipo de Notación, para el peor caso NOSOTROS TRABAJAREMOS CON EL BIG O

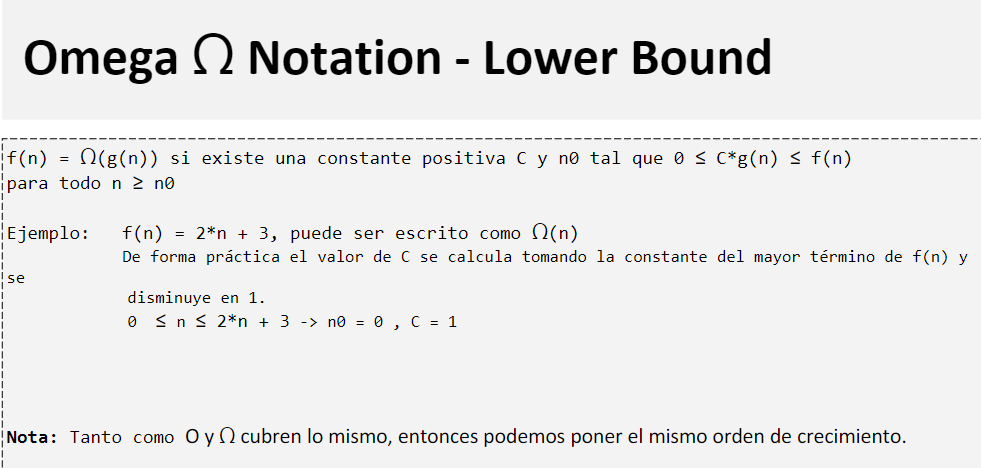


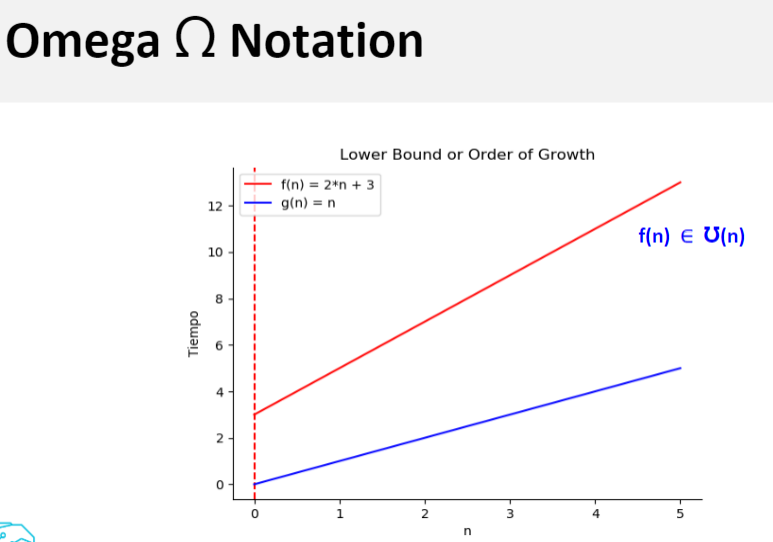
**PARA ESTE ANALISIS DE COMPLEJIDAD USAREMOS EL BIG O**

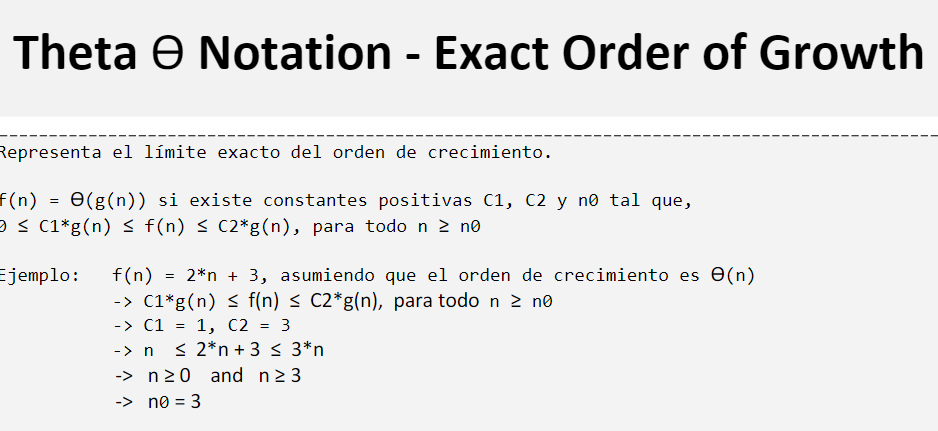
El Big O es el factor pesimista, vamos a buscar el peor caso. Pero acuérdate el Big O no siempre será la situación real en la que trabajamos.

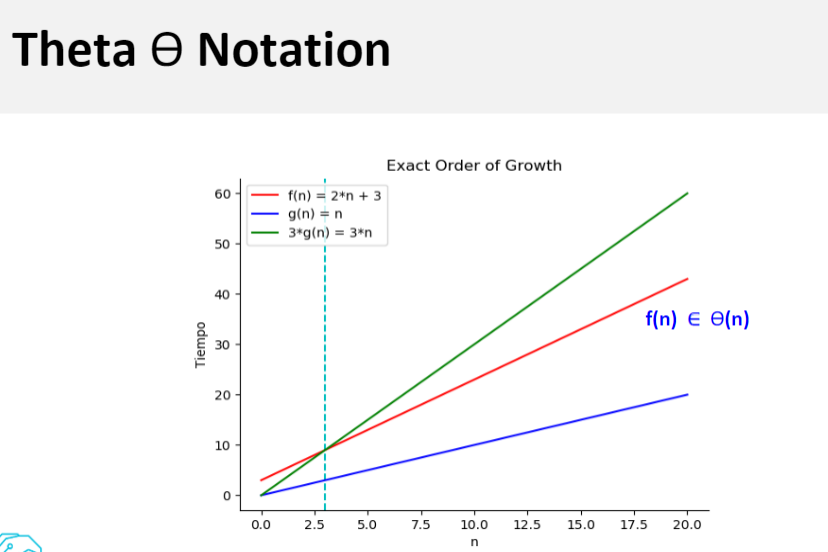




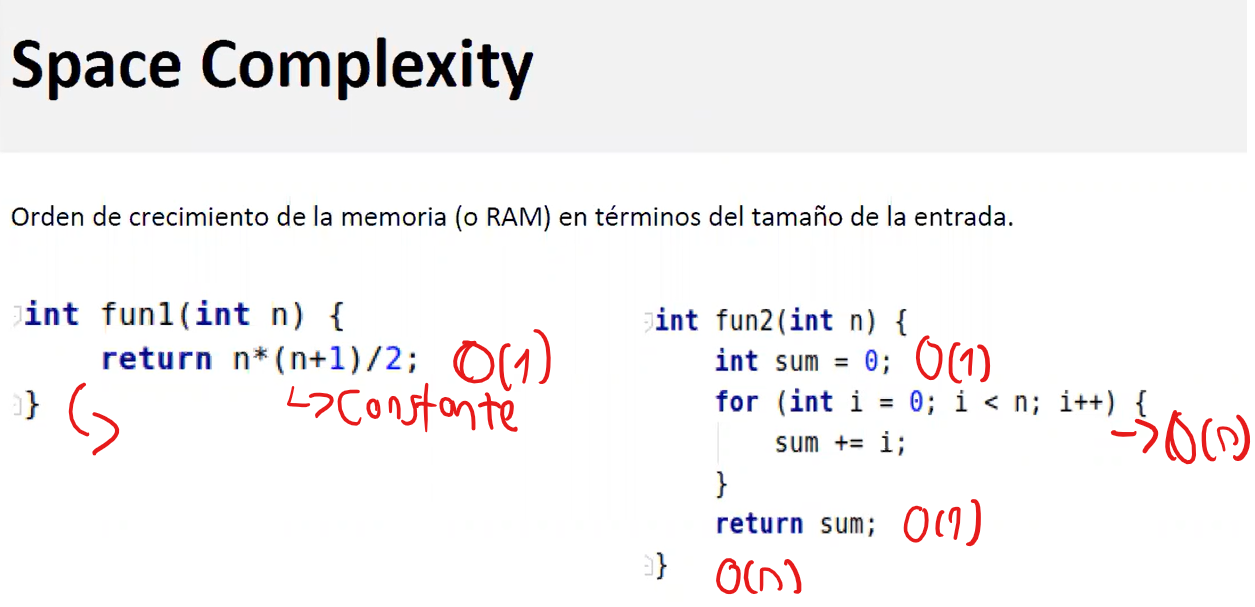








**COMPLEJIDAD ESPACIAL**

****

En la complejidad de tiempo para el primer caso saldría O(1) y para el segundo saldría O(n)

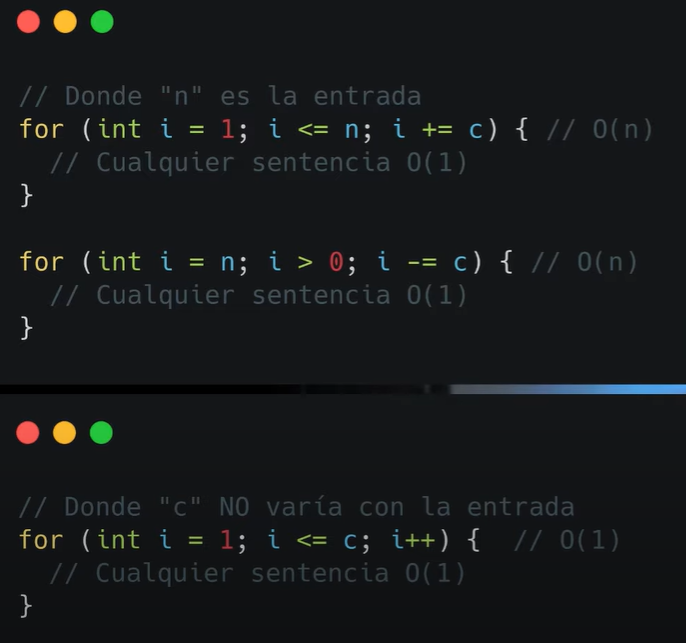
Pero en la complejidad de espacio no es la misma que tiempo, porque en el espacio se mide la cantidad de variables que se esta usando.   
En el primer caso su complejidad de espacio es 1, porque solamente utilizo una variable.

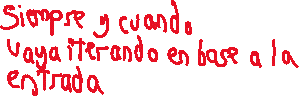
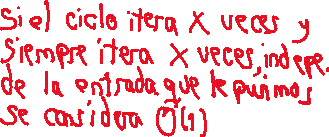
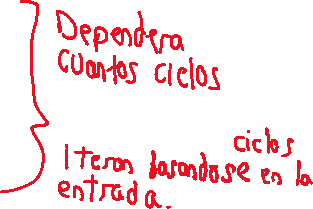
En el segundo caso su complejidad de espacio es 3, porque se esta utilizando 3 variables.

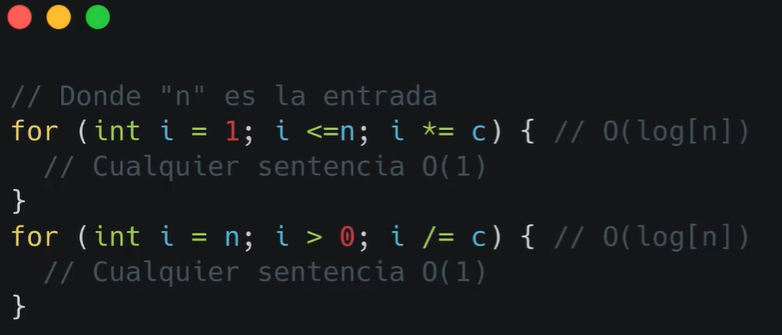
Pero en notación Big O para ambos casos noma seria O(1) – Big O de 1 o constante en ambos casos, que es diferente a cuando medimos el tiempo en notación Big O.

Así que no necesariamente la complejidad de tiempo será paralela a la de espacio.

**ANALISIS DE CICLOS:**

****



****



|

