Notación Asintótica 2

mat-151

Éxamen Rápido (6 minutos)

- Cada operación fundamental usa *c* milisegundos, ¿cuánto tiempo toma contar hasta 1,000,000? ¿Cuál es el valor de N?¿Cuál es el órden de complejidad en función de N?

```
int i,k,count=0;
for(i=0; i<N; i++)
  for(k=0; k<N; k++)
      count ++;
```

Notación asintótica

- **Comportamiento asintótico:** comportamiento cuando $n \to \infty$, donde *n* es el tamaño del problema.

- Tres notaciones básicas:

$f \sim g$ ("f y g son asintóticamente equivalentes")

$f \preceq g$ ("f está asintóticamente dominada por g ")

$f \asymp g$ ("f y g están respectivamente asintóticamente acotadas")

Notación asintótica (equivalentes)

$$\boxed{f \sim g \quad \text{significa} \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1}$$

- Ejemplo: $3x^2 + 2x + 1 \sim 3x^2$

Para *n* grande son muy parecidas

- $\sim$ es una **relación de equivalencia:**

➡Transitiva: $(x \sim y) \wedge (y \sim z) \Rightarrow (x \sim z)$

➡Reflexiva: $x \sim x$

➡Simétrica: $(x \sim y) \Rightarrow (y \sim x)$

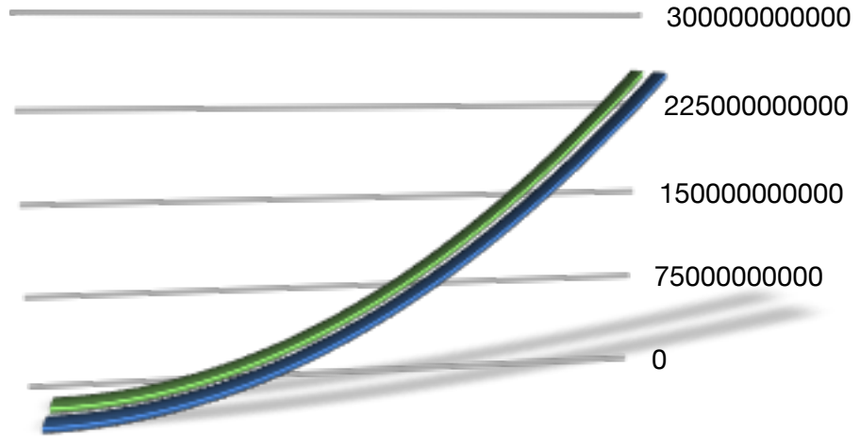
- Idea básica: tomar en cuenta solo los ***términos principales***, ignorando aquellos que crecen más lento.

Notación asintótica (equivalentes)

$$f \sim g \quad \text{significa} \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

- Ejemplo: $3x^2 + 2x + 1 \sim 3x^2$
- $\sim$ es una **relación de equivalencia:**
  - ➡ Transitiva: $(x \sim y) \wedge (y \sim z) \Rightarrow (x \sim z)$
  - ➡ Reflexiva: $x \sim x$
  - ➡ Simétrica: $(x \sim y) \Rightarrow (y \sim x)$

Para *n* grande son muy parecidas



- Idea básica: tomar en cuenta solo los ***términos principales***, ignorando aquellos que crecen más lento.

Notación asintótica (dominada)

$$f \preceq g \quad \text{significa} \quad \limsup_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

- i.e., $\frac{f(n)}{g(n)}$ está eventualmente acotado por un valor finito.

- Idea básica: ***f*** crece más lento que ***g***, o tan rápido como ***g***.

$\preceq$

- Transitivo: $(f \preceq g) \wedge (g \preceq h) \Rightarrow (f \preceq h)$
- Reflexivo: $f \preceq f$

Notación asintótica (respectivamente acotadas)

Escribir $f \asymp g$ cuando hay constantes positivas c_1, c_2 tales que,

$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

para una n suficientemente grande.

- Ejemplos:
  - $n \asymp 2n$
  - $n \asymp (2 + sin\pi n)n$
- $\asymp$ es una relación de equivalencia.

Notación O grande

$$T(n) = O(f(n)) \quad \text{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \leq cf(n)$$

Notación O grande

$$T(n) = O(f(n)) \quad \text{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \leq cf(n)$$

- Se lee: “T(n) es de orden f(n)”

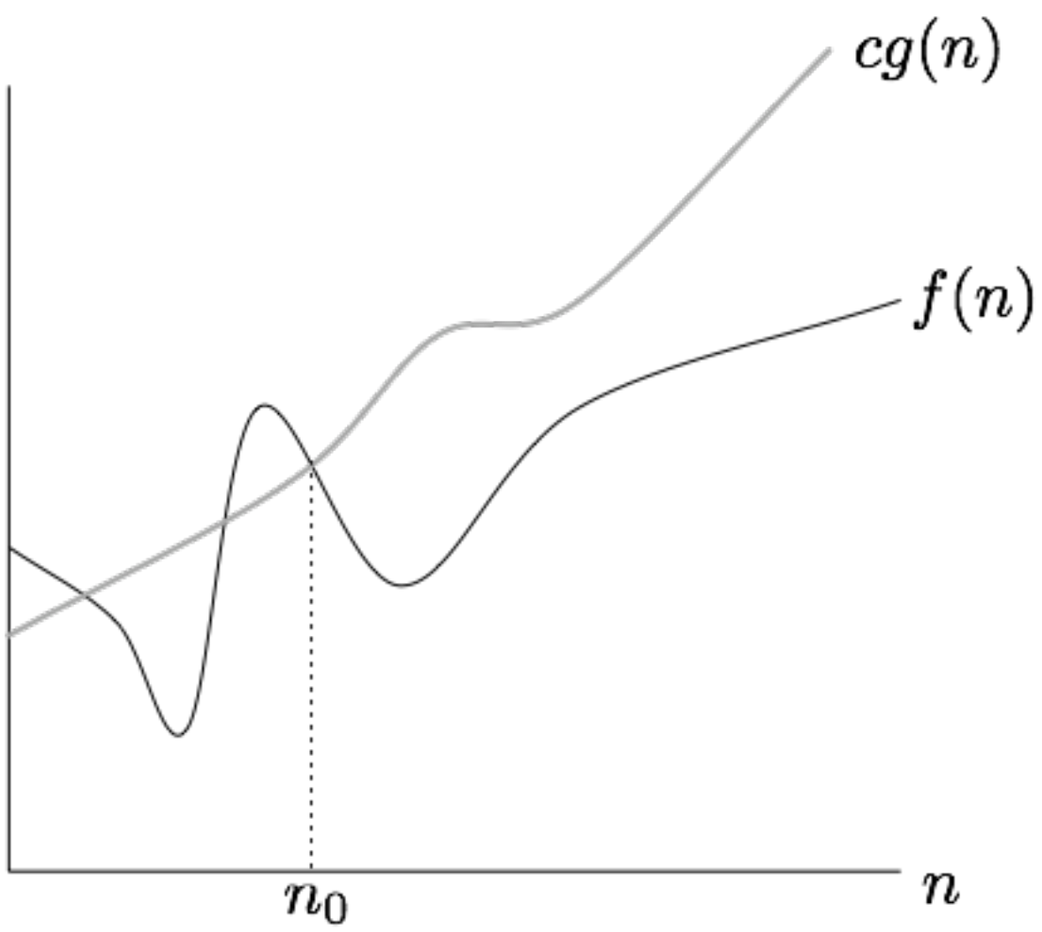
Notación O grande

$$T(n) = O(f(n)) \quad \text{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \leq cf(n)$$

- Se lee: “T(n) es de orden f(n)”
- T(n) no crece más rápido que f(n)

Notación O grande

$$T(n) = O(f(n)) \quad \text{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \leq cf(n)$$

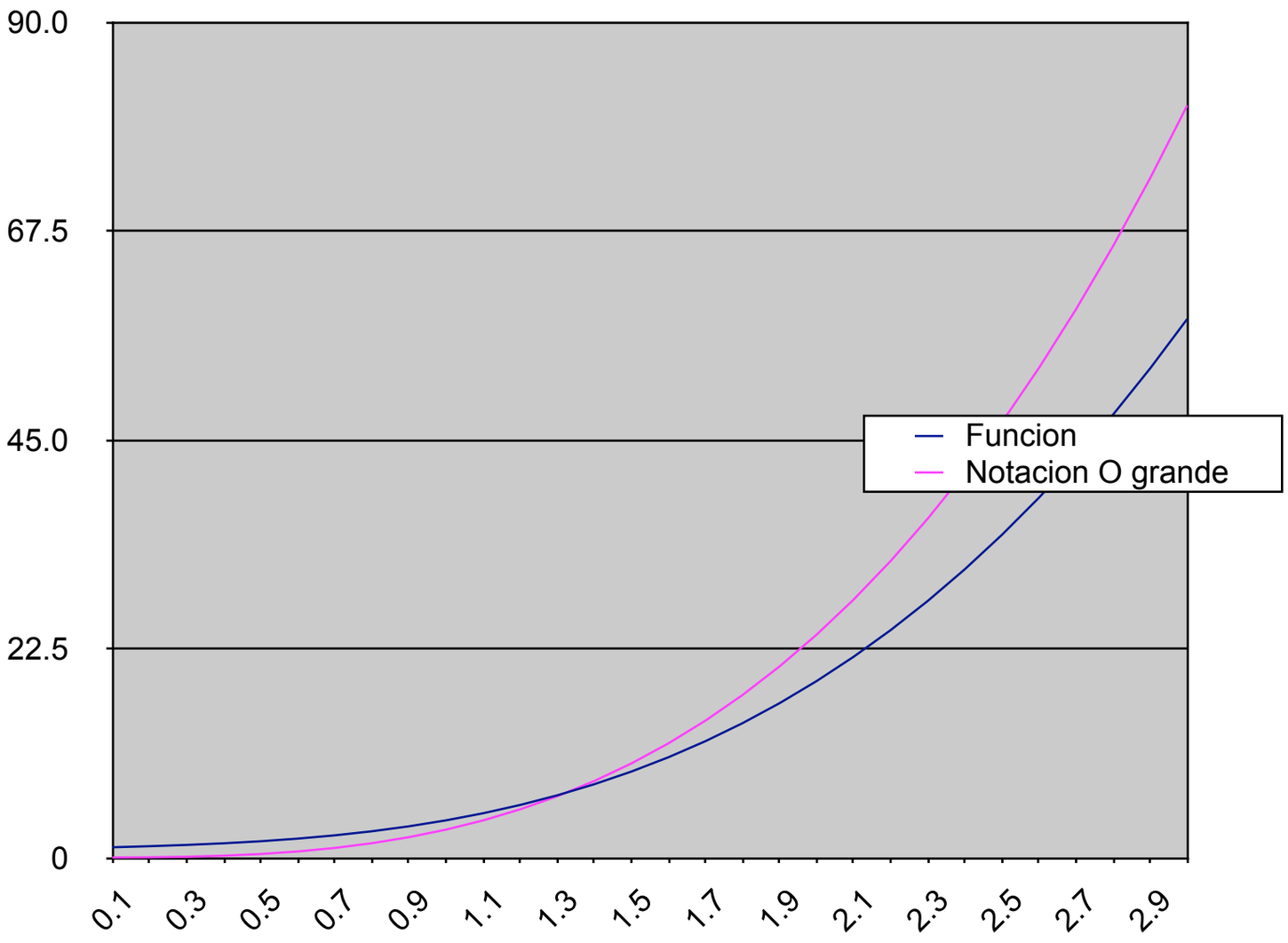


$$f(n) = O(g(n))$$

- Se lee: “T(n) es de orden f(n)”
- T(n) no crece más rápido que f(n)

Notación O grande

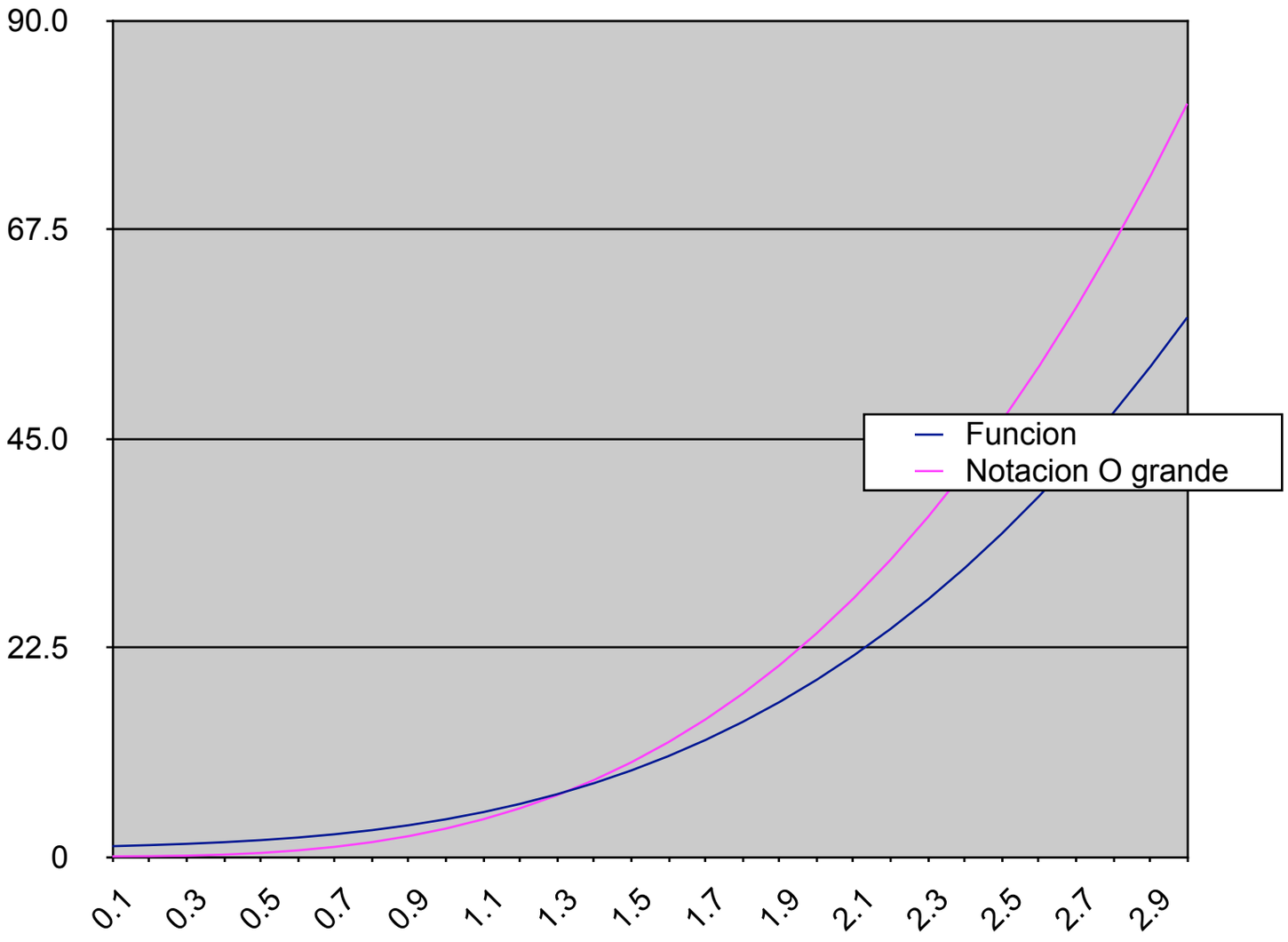
$$T(n) = 2n^3 + n + 1 = O(n^3)$$



Notación O grande

$$T(n) = 2n^3 + n + 1 = O(n^3)$$

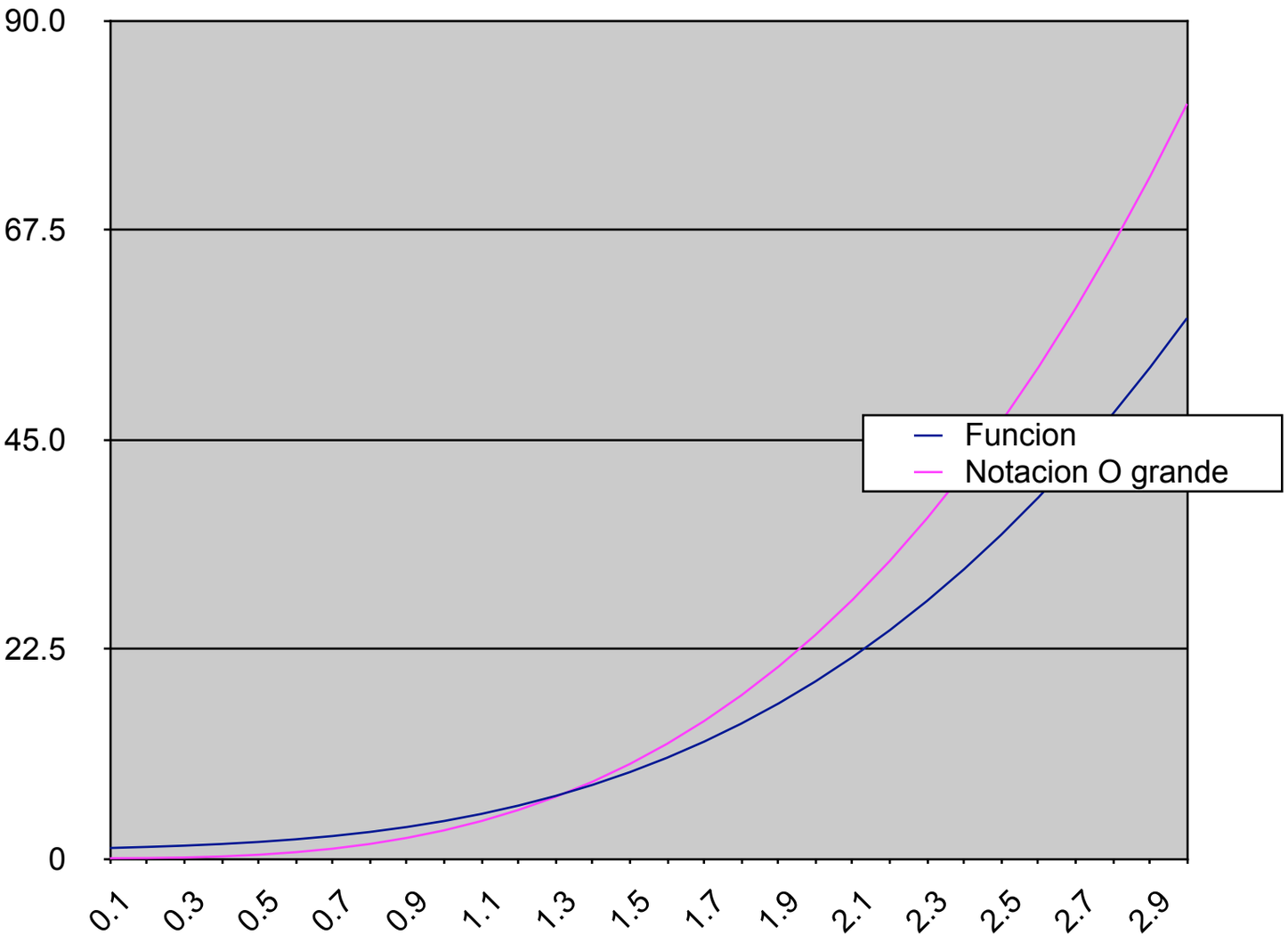
- porque haciendo (c,n_0) = (3,2) verificamos que para n≥2



Notación O grande

$$T(n) = 2n^3 + n + 1 = O(n^3)$$

- porque haciendo *(c,n_0)* = (3,2) verificamos que para n≥2

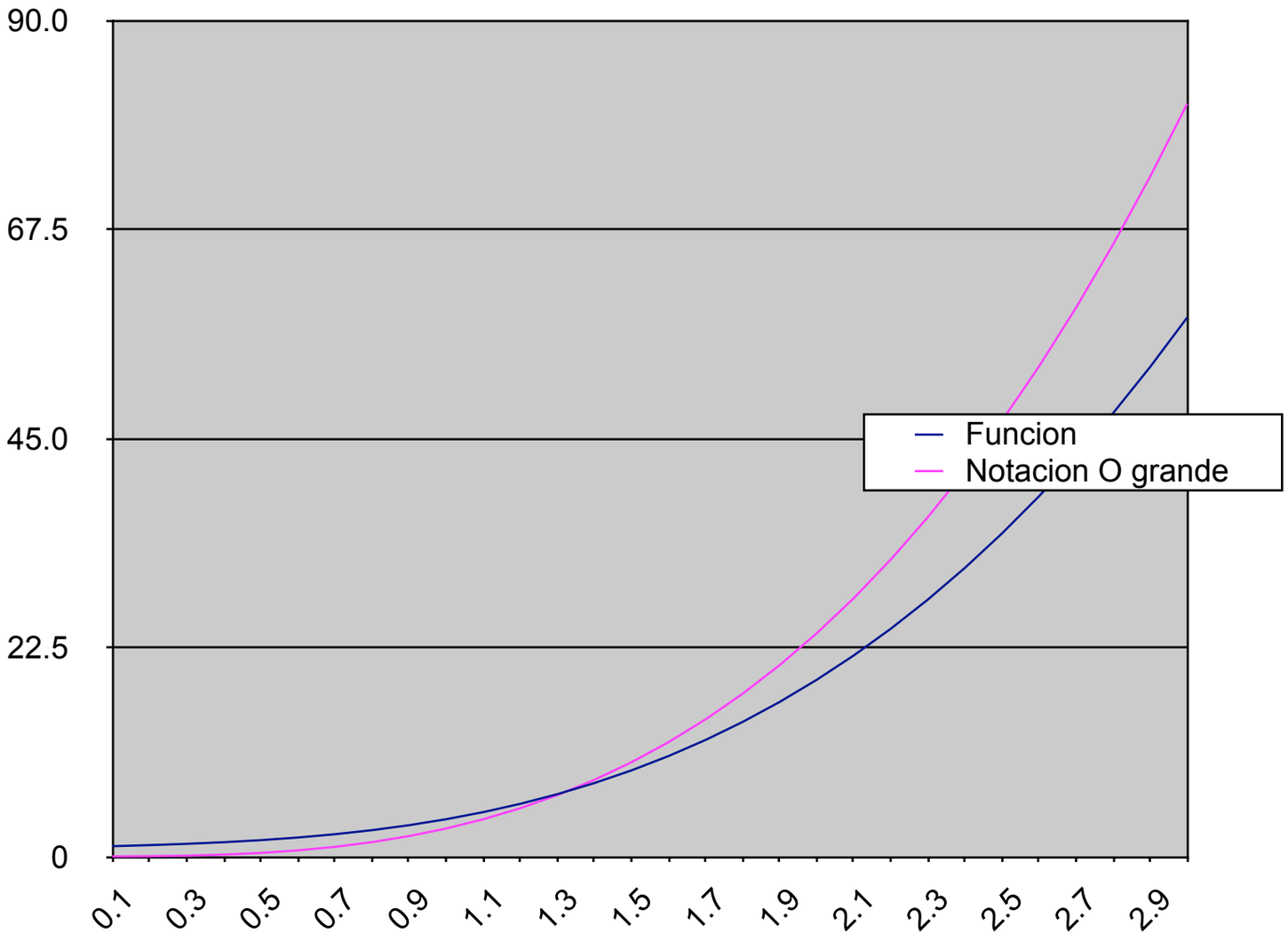


$$T(n) = 2n^3 + n + 1$$

Notación O grande

$$T(n) = 2n^3 + n + 1 = O(n^3)$$

- porque haciendo *(c,n_0)* = (3,2) verificamos que para n≥2



$$T(n) = 2n^3 + n + 1$$

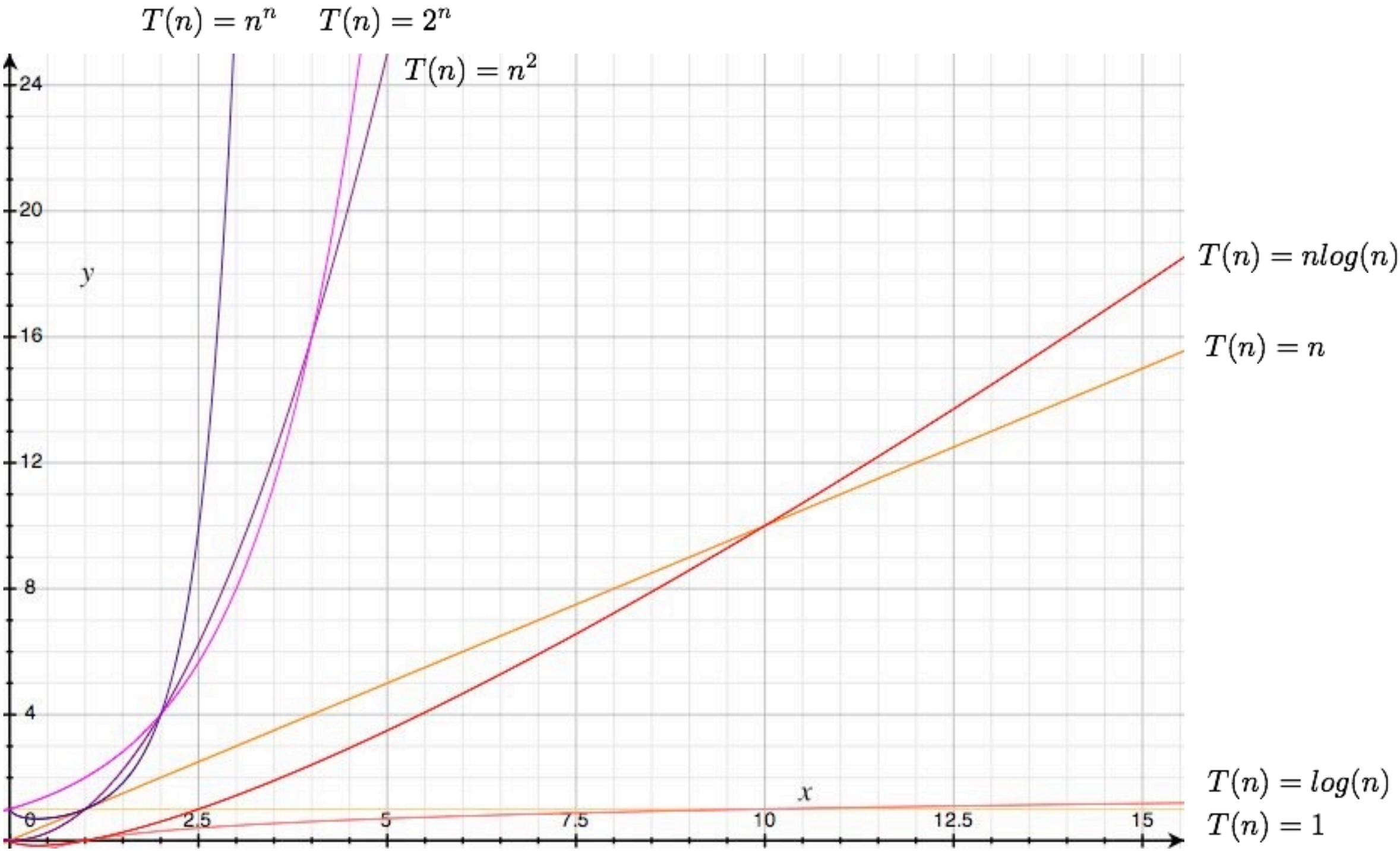
$$\leq 3n^3$$

Notación O grande

- La notación O grande permite **definir un límite superior al crecimiento de una función**.

- Establece un **"orden" entre funciones:**

$$1 \leq \log n \leq n \leq n \log n \leq n^2 \leq 2^n \leq n^n$$


$T(n) = n^n$
$T(n) = 2^n$
$T(n) = n^2$
$T(n) = nlog(n)$
$T(n) = n$
$T(n) = log(n)$
$T(n) = 1$
y
x
24
20
16
12
8
4
0
2.5
5
7.5
10
12.5
15

Notación O grande

- Considerar funciones que sean lo **más pequeñas posibles**:

Notación O grande

- Considerar funciones que sean lo **más pequeñas posibles**:

$$
\begin{aligned}
T(n) &= 2n^3 + n + 1 \\
T(n) &= O(n^4) \\
T(n) &= O(2^n) \\
T(n) &= O(n!)
\end{aligned}
$$

Notación O grande

- Considerar funciones que sean lo **más pequeñas posibles**:

$$
\begin{aligned}
T(n) &= 2n^3 + n + 1 \\
T(n) &= O(n^4) \\
T(n) &= O(2^n) \\
T(n) &= O(n!)
\end{aligned}
$$

- Si ***T(n)*** es un polinomio de grado ***g***, entonces ***T(n) = O(n^g)***: se pueden “olvidar” los términos de grado inferior y las constantes.

Notación O grande

- Considerar funciones que sean lo **más pequeñas posibles**:

$$
\begin{aligned}
T(n) &= 2n^3 + n + 1 \\
T(n) &= O(n^4) \\
T(n) &= O(2^n) \\
T(n) &= O(n!)
\end{aligned}
$$

- Si ***T(n)*** es un polinomio de grado ***g***, entonces ***T(n) = O(n^g)***: se pueden “olvidar” los términos de grado inferior y las constantes.

- Usar la **clase de funciones más chica**: *2n* es *O(n)*, aunque sea también O(n^2)

Notación O grande

- Considerar funciones que sean lo **más pequeñas posibles**:

$$
\begin{aligned}
T(n) &= 2n^3 + n + 1 \\
T(n) &= O(n^4) \\
T(n) &= O(2^n) \\
T(n) &= O(n!)
\end{aligned}
$$

- Si ***T(n)*** es un polinomio de grado ***g***, entonces ***T(n) = O(n^g)***: se pueden “olvidar” los términos de grado inferior y las constantes.

- Usar la **clase de funciones más chica**: *2n* es *O(n)*, aunque sea también O(n^2)

- Usar la **expresión más simple**

Notación Ω

$$T(n) = \Omega(f(n)) \quad \textbf{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \geq cf(n)$$

Notación Ω

- Presenta la **cota asintótica inferior** de una función.

$$T(n) = \Omega(f(n)) \quad \textbf{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \geq cf(n)$$

Notación Ω

- Presenta la **cota asintótica inferior** de una función.

$$T(n) = \Omega(f(n)) \quad \textbf{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \geq cf(n)$$

- Se usa generalmente para acotar el tiempo de ejecución en el **mejor caso**.

Notación Ω

- Presenta la **cota asintótica inferior** de una función.

$$T(n) = \Omega(f(n)) \quad \textbf{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \geq cf(n)$$

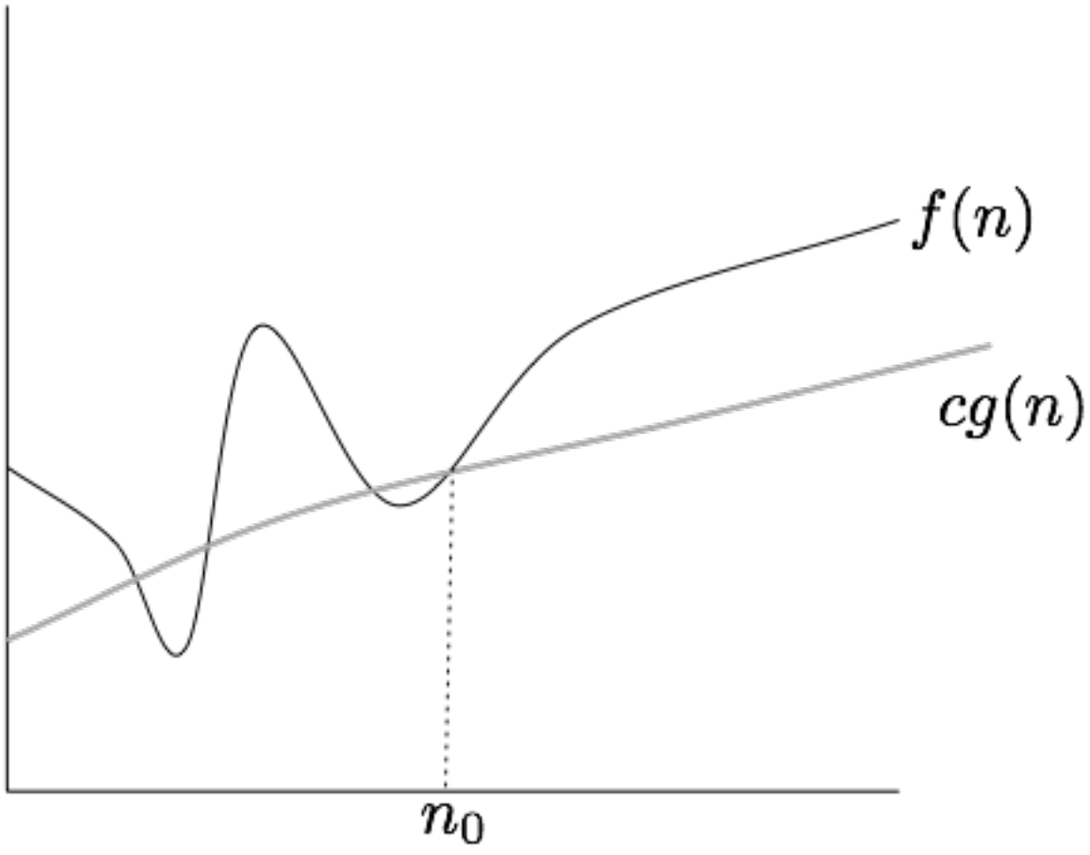
- Se usa generalmente para acotar el tiempo de ejecución en el **mejor caso**.

- Cuando decimos que el *tiempo de ejecución* de un algoritmo es *Ω(g(n))*, decimos que **sin importar la entrada de tamaño *n*, el tiempo de cálculo para esa entrada es al menos *una constante multiplicativa de g(n)*,** para una *n* suficientemente grande.

Notación Ω

- Presenta la **cota asintótica inferior** de una función.

$$T(n) = \Omega(f(n)) \quad \textbf{ssi} \quad \exists (c, n_0) \in \mathbb{R}^+ \times \mathbb{N} \mid \forall\, n \geq n_0 \quad T(n) \geq cf(n)$$



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

- Se usa generalmente para acotar el tiempo de ejecución en el **mejor caso**.

- Cuando decimos que el *tiempo de ejecución* de un algoritmo es *Ω(g(n))*, decimos que **sin importar la entrada de tamaño *n*, el tiempo de cálculo para esa entrada es al menos *una constante multiplicativa de g(n)*,** para una *n* suficientemente grande.

Notación Θ

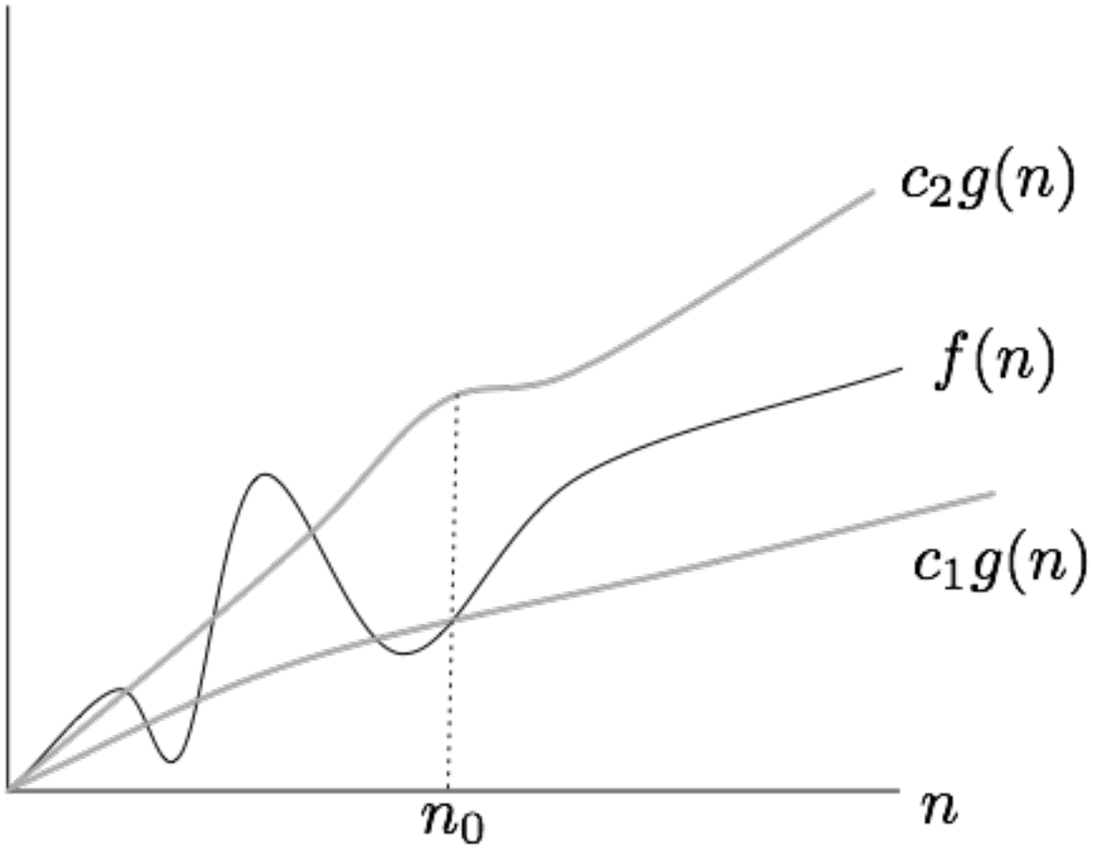
$$T(n) = \Theta(f(n)) \quad \text{ssi} \quad \exists (c_1, c_2, n_0) \in \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{N}$$

$$\text{tal que} \quad \forall\, n \geq n_0 \quad 0 \leq c_1 f(n) \leq T(n) \leq c_2 f(n)$$

Notación Θ

$$T(n) = \Theta(f(n)) \quad \text{ssi} \quad \exists (c_1, c_2, n_0) \in \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{N}$$

$$\text{tal que} \quad \forall\, n \geq n_0 \quad 0 \leq c_1 f(n) \leq T(n) \leq c_2 f(n)$$



$$f(n) = \Theta(g(n))$$

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Determinar las constantes positivas c_1, c_2, y n_0 tales que

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Determinar las constantes positivas c_1, c_2, y n_0 tales que

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Determinar las constantes positivas c_1, c_2, y n_0 tales que

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

Dividiendo por n^2 tenemos:

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Determinar las constantes positivas c_1, c_2, y n_0 tales que

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

Dividiendo por n^2 tenemos:

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Determinar las constantes positivas c_1, c_2, y n_0 tales que

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

Dividiendo por n^2 tenemos:

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$\forall n \geq 1, \frac{1}{2} - \frac{3}{n} \leq \frac{1}{2}$$

Notación Θ

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Determinar las constantes positivas c_1, c_2, y n_0 tales que

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

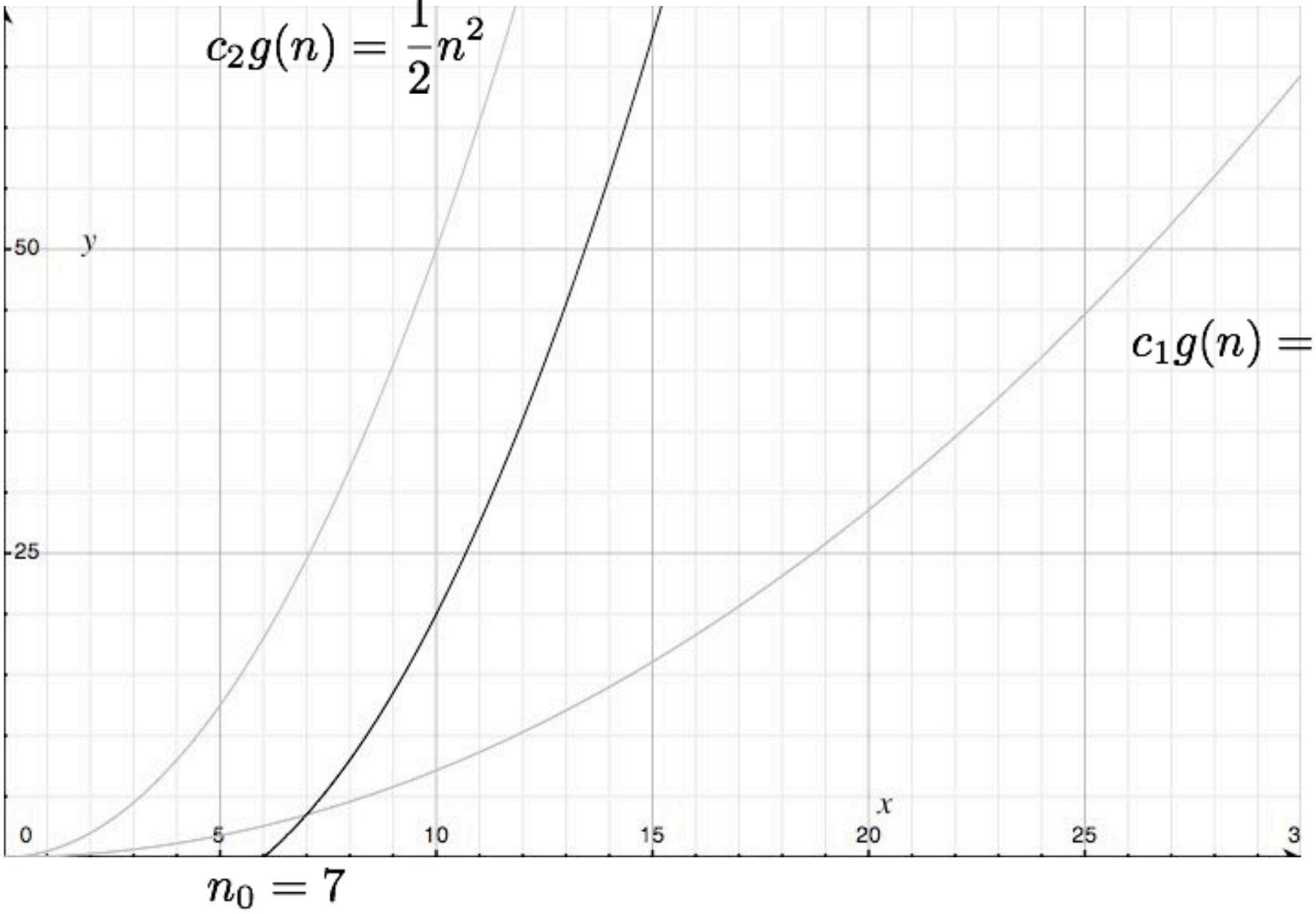
Dividiendo por n^2 tenemos:

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$\forall n \geq 1, \frac{1}{2} - \frac{3}{n} \leq \frac{1}{2} \qquad \forall n \geq 7, \frac{1}{2} - \frac{3}{n} \geq \frac{1}{14}$$

Notación Θ

$$f(n) = \frac{1}{2}n^2 - 3n$$



$$f(n) = \Theta(n^2)$$

Notación asintótica (propiedades)

- **Transitividad**

$$\text{Si } f(n) = O(g(n)) \quad \text{y} \quad g(n) = O(h(n)), \text{ entonces } f(n) = O(h(n))$$

Notación asintótica (propiedades)

- **Transitividad**

$$\text{Si } f(n) = O(g(n)) \quad \text{y} \quad g(n) = O(h(n)), \text{ entonces } f(n) = O(h(n))$$

Prueba:

$$\exists (c > 0, n_0) \quad | \quad \forall n \geq n_0, f(n) \leq cg(n)$$

$$\exists (c' > 0, n'_0) \quad | \quad \forall n \geq n'_0, g(n) \leq c'h(n)$$

Si consideramos un número n, al menos tan grande como ambos, n_0 y n'_0

$$f(n) \leq cg(n) \leq cc'h(n), \text{ entonces } f(n) \leq cc'h(n) \quad \forall \quad n \geq \max(n_0, n'_0)$$ ■

Notación asintótica (propiedades)

- **Suma**

$$\text{Si} f(n) = O(h(n)) \text{ y } g(n) = O(h'(n)), \text{ entonces}$$

$$f(n) + g(n) = O(\max(h(n), h'(n)))$$

- **Prueba:**

Notación asintótica (propiedades)

- **Suma**

$$\text{Si} f(n) = O(h(n)) \text{ y } g(n) = O(h'(n)), \text{ entonces}$$

$$f(n) + g(n) = O(\max(h(n), h'(n)))$$

- **Prueba:**

Sea $n_0 = \max(n_1, n_2)$ y $c_0 = 2\max(c_1, c_2)$. Considere la suma $f_1(n) + f_2(n)$ para $n \geq n_0$:

$$\begin{aligned} f_1(n) + f_2(n) &\leq c_1 g_1(n) + c_2 g_2(n), \quad n \geq n_0 \\ &\leq c_0 (g_1(n) + g_2(n))/2 \\ &\leq c_0 \max(g_1(n), g_2(n)). \end{aligned}$$

Notación asintótica (propiedades)

- **Producto**

$$\text{Si } f(n) = O(h(n)) \quad \text{y} \quad g(n) = O(h'(n)), \text{ entonces}$$

$$f(n)g(n) = O(h(n)h'(n))$$

Notación asintótica (propiedades)

- **Producto**

$$\text{Si } f(n) = O(h(n)) \quad \text{y} \quad g(n) = O(h'(n)), \text{ entonces}$$

$$f(n)g(n) = O(h(n)h'(n))$$

Prueba

$$\exists (c > 0, n_0) \quad | \quad \forall\ n \geq n_0, f(n) \leq ch(n)$$

$$\exists (c' > 0, n'_0) \quad | \quad \forall\ n \geq n'_0, g(n) \leq c'h'(n)$$

$$\text{entonces } \forall\ n \geq \max(n_0, n'_0), \quad f(n)g(n) \leq cc'h(n)h'(n) \ \blacksquare$$

Notación asintótica (propiedades)

$$(N + O(1))(N + O(logN) + O(1))$$

$$= \ N^2 + O(N) + O(NlogN) + O(logN) + O(N) + O(1)$$

$$= \ N^2 + O(NlogN)$$

Ejemplos de análisis de algortimos

Ejemplos de análisis de algortimos

- Análisis del algoritmo de *búsqueda secuencial* y *búsqueda binaria*.

Ejemplos de análisis de algortimos

- Análisis del algoritmo de *búsqueda secuencial* y *búsqueda binaria*.
    - Suponemos que la entrada son números enteros

Ejemplos de análisis de algortimos

- Análisis del algoritmo de *búsqueda secuencial* y *búsqueda binaria*.
  - Suponemos que la entrada son números enteros
  - Compañía de tarjetas de crédito, N tarjetas robadas, M transacciones a verificar.

Ejemplos de análisis de algortimos

- Análisis del algoritmo de *búsqueda secuencial* y *búsqueda binaria*.
  - Suponemos que la entrada son números enteros
  - Compañía de tarjetas de crédito, N tarjetas robadas, M transacciones a verificar.
    - N del orden de (10^3 a 10^6)

Ejemplos de análisis de algortimos

- Análisis del algoritmo de *búsqueda secuencial* y *búsqueda binaria*.
  - Suponemos que la entrada son números enteros
  - Compañía de tarjetas de crédito, N tarjetas robadas, M transacciones a verificar.
    - N del orden de (10^3 a 10^6)
    - M del orden de (10^6 a 10^9)

Ejemplos de análisis de algortimos

- Análisis del algoritmo de *búsqueda secuencial* y *búsqueda binaria*.
  - Suponemos que la entrada son números enteros
  - Compañía de tarjetas de crédito, N tarjetas robadas, M transacciones a verificar.
    - N del orden de (10^3 a 10^6)
    - M del orden de (10^6 a 10^9)
- Estimar el tiempo de cálculo cuando los parámetros caen en estos rangos.

Búsqueda secuencial

```
int search ( int a[], int v, int l, int r )
{
  for ( int i=l; i<=r; i++ )
    if( v == a[i]) return i;
  return -1;
}
```

Búsqueda secuencial

```
int search ( int a[], int v, int l, int r )
{
  for ( int i=l; i<=r; i++ )
    if( v == a[i]) return i;
  return -1;
}
```

- El tiempo de ejecución depende si el objeto buscado está en el arreglo o no.

Búsqueda secuencial

```
int search ( int a[], int v, int l, int r )
{
  for ( int i=l; i<=r; i++ )
    if( v == a[i]) return i;
  return -1;
}
```

- El tiempo de ejecución depende si el objeto buscado está en el arreglo o no.

- Podemos decir que no está después de recorrer todo el arreglo, o encontrarlo en la primera, segunda, tercera o cualquiera de las posiciones.

Búsqueda secuencial

```
int search ( int a[], int v, int l, int r )
{
  for ( int i=l; i<=r; i++ )
    if( v == a[i]) return i;
  return -1;
}
```

- El tiempo de ejecución depende si el objeto buscado está en el arreglo o no.
- Podemos decir que no está después de recorrer todo el arreglo, o encontrarlo en la primera, segunda, tercera o cualquiera de las posiciones.
- Para hacer una predicción se necesita una entrada adecuada a la aplicación.

Búsqueda secuencial

Búsqueda secuencial

- Tomamos el caso de número **encontrado** o **no encontrado** y los analizamos separadamente.

Búsqueda secuencial

- Tomamos el caso de número **encontrado** o **no encontrado** y los analizamos separadamente.

Propiedad 1: La búsqueda secuencial examina ***M*** números para cada número *no encontrado* y en promedio ***M/2*** posiciones para cada búsqueda *exitosa*.

Búsqueda secuencial

- Tomamos el caso de número **encontrado** o **no encontrado** y los analizamos separadamente.

Propiedad 1: La búsqueda secuencial examina ***M*** números para cada número *no encontrado* y en promedio ***M/2*** posiciones para cada búsqueda *exitosa*.

Si cada número en el arreglo tiene la misma probabilidad de ser el número buscado, entonces: ***(1+2+ ... + M)/M = (M+1)/2*** es el costo promedio de la búsqueda.

Búsqueda secuencial

- Tomamos el caso de número **encontrado** o **no encontrado** y los analizamos separadamente.

Propiedad 1: La búsqueda secuencial examina ***M*** números para cada número *no encontrado* y en promedio ***M/2*** posiciones para cada búsqueda *exitosa*.

Si cada número en el arreglo tiene la misma probabilidad de ser el número buscado, entonces: ***(1+2+ ... + M)/M = (M+1)/2*** es el costo promedio de la búsqueda.

- Esto implica que cada búsqueda es proporcional a ***M***, suponiendo que la comparación entre dos números es constante: *si duplicamos el número de transacciones, duplicamos el tiempo de cálculo.*

Búsqueda secuencial

Búsqueda secuencial

- ¿Cómo acelerar la búsqueda?

Búsqueda secuencial

- ¿Cómo acelerar la búsqueda?
    - **¡ORDENANDO LA LISTA!**

Búsqueda secuencial

- ¿Cómo acelerar la búsqueda?
  - **¡ORDENANDO LA LISTA!**
  - Veremos algoritmos de ordenamiento de ***O(M logM)***, adecuado dado el tamaño de ***N***.

Búsqueda secuencial

- ¿Cómo acelerar la búsqueda?
    - **¡ORDENANDO LA LISTA!**
    - Veremos algoritmos de ordenamiento de ***O(M logM)***, adecuado dado el tamaño de ***N***.
- Con una lista ordenada se puede terminar la búsqueda al llegar al primer número más grande al que buscamos.

Búsqueda secuencial

- ¿Cómo acelerar la búsqueda?
    - **¡ORDENANDO LA LISTA!**
    - Veremos algoritmos de ordenamiento de ***O(M logM)***, adecuado dado el tamaño de ***N***.
- Con una lista ordenada se puede terminar la búsqueda al llegar al primer número más grande al que buscamos.
- Esto reduce el costo a cerca de ***M/2*** para una búsqueda **no exitosa**, lo mismo que era para la búsqueda **exitosa**.

Búsqueda secuencial

- ¿Cómo acelerar la búsqueda?
    - **¡ORDENANDO LA LISTA!**
    - Veremos algoritmos de ordenamiento de ***O(M logM)***, adecuado dado el tamaño de ***N***.
- Con una lista ordenada se puede terminar la búsqueda al llegar al primer número más grande al que buscamos.
- Esto reduce el costo a cerca de ***M/2*** para una búsqueda **no exitosa**, lo mismo que era para la búsqueda **exitosa**.
- **Propiedad 2:** La *búsqueda secuencial* en un arreglo ordenado examina ***M*** números en el peor caso y cerca de ***M/2*** números para cada búsqueda en promedio.

Búsqueda secuencial

Búsqueda secuencial

- El tiempo de ejecución de la búsqueda secuencial es proporcional a ***MN*** para ***M*** transacciones, en promedio y en el peor caso.

Búsqueda secuencial

- El tiempo de ejecución de la búsqueda secuencial es proporcional a ***MN*** para ***M*** transacciones, en promedio y en el peor caso.
- Esto nos dice que este algoritmo no es utilizable para tablas muy grandes:

Búsqueda secuencial

- El tiempo de ejecución de la búsqueda secuencial es proporcional a ***MN*** para ***M*** transacciones, en promedio y en el peor caso.
- Esto nos dice que este algoritmo no es utilizable para tablas muy grandes:
  - Si toma *c* microsegundos examinar un número y $M=10^9$ *y* $N=10^6$, el tiempo de ejecución para todas las transacciones será de al menos $(c/2)10^9$ segundos, o alrededor de *16c* años.

Búsqueda binaria

```
• int search ( int a[], int v, int l, int r )
• {
   •while (r >= l){
      •int m = (l+r)/2;
      •if (v == a[m]) return m;
      •if (v < a[m]) r=m-1; else l=m+1;
   •}
   •return -1;
• }
```

Búsqueda binaria

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.
- Si es igual, la búsqueda es **exitosa.**

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.
- Si es igual, la búsqueda es **exitosa.**
- Si es **menor**, aplicamos el mismo método a la **mitad izquierda** de la tabla.

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.
- Si es igual, la búsqueda es **exitosa.**
- Si es **menor**, aplicamos el mismo método a la **mitad izquierda** de la tabla.
- Si es **mayor**, aplicamos el mismo método a la **mitad derecha** de la tabla.

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.
- Si es igual, la búsqueda es **exitosa.**
- Si es **menor**, aplicamos el mismo método a la **mitad izquierda** de la tabla.
- Si es **mayor**, aplicamos el mismo método a la **mitad derecha** de la tabla.

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.
- Si es igual, la búsqueda es **exitosa.**
- Si es **menor**, aplicamos el mismo método a la **mitad izquierda** de la tabla.
- Si es **mayor**, aplicamos el mismo método a la **mitad derecha** de la tabla.

- **Propiedad 1:** La búsqueda binaria nunca examina más de $(lg_2N)+1$ números.

Búsqueda binaria

- Si los números están ordenados, se puede **eliminar la mitad** de ellos comparando el que buscamos con el de la **posición media** de la tabla.
- Si es igual, la búsqueda es **exitosa.**
- Si es **menor**, aplicamos el mismo método a la **mitad izquierda** de la tabla.
- Si es **mayor**, aplicamos el mismo método a la **mitad derecha** de la tabla.

- **Propiedad 1:** La búsqueda binaria nunca examina más de $(lg_2N)+1$ números.
- **Lo que significa:** Permite resolver problemas muy grandes, de hasta 1 millón de números con máximo 20 comparaciones por transacción, que puede ser menos de lo que tarda una computadora en imprimir o leer el número. En este caso, con c=1, aprox. $\log_2(M)*N/1e^{-6} < 30$ seg.

Problemas de búsqueda

- Un análisis del comportamiento asintótico de estos algoritmos nos permite elegir el mejor entre ellos para nuestra aplicación.

- El análisis del **peor caso** nos proporciona **garantías** sobre el funcionamiento del programa.

- El análisis del **caso promedio** nos da **predicciones** sobre el funcionamiento promedio, con los datos apropiados, del programa.