

Chapter 1.1

Pensando en Algoritmos

Introducción

- ¿Así que quieres ser un profesional exitoso en la informática?
- Recordando: ¿Cómo multiplicar dos números?

Presentación realizada por: Jeff Edmonds
Universidad de York

¿Así que quieres ser un profesional
exitoso en la informática?



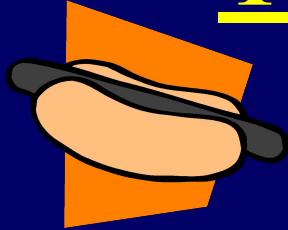
¿Tu objetivo es ser
un profesional común y corriente?



¿O un gran líder y pensador?



Pensamiento Original



Tareas comunes que el Jefe nos asigna:

- Teniendo en cuenta los precios actuales de la carne de cerdo, grano, aceite, etc.
- Conociendo el proceso para la elaboración del “hotdog”.
- ¿De qué forma hacemos más barato el “hotdog”.



Cada día las empresas realizan estas preguntas.

Tu respuesta:

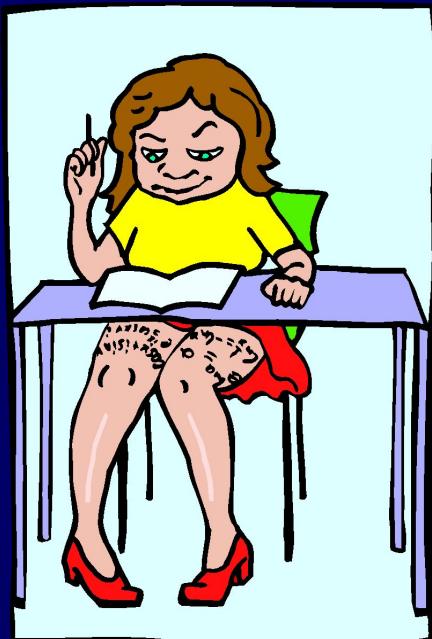
- ¿Um? Dime que codificar.



Excusa: Con la abundancia de Softwares, la demanda de programadores disminuirá.

Tu respuesta:

- Aprendí este gran algoritmo que va a funcionar.



*Pronto todos los algoritmos conocidos
estarán disponibles en las librerías.*

Tu respuesta debería ser :)

- Puedo desarrollar un nuevo algoritmo para usted.



*Los grandes pensadores
siempre serán necesarios.*

El futuro pertenece al profesional que posee:

- **Contenidos:** Hasta la actualidad una comprensión fundamental de los problemas y las soluciones.
- **Métodos:** Principios y técnicas para resolver la amplia gama de problemas desconocidas que surgen en un campo que cambia rápidamente.



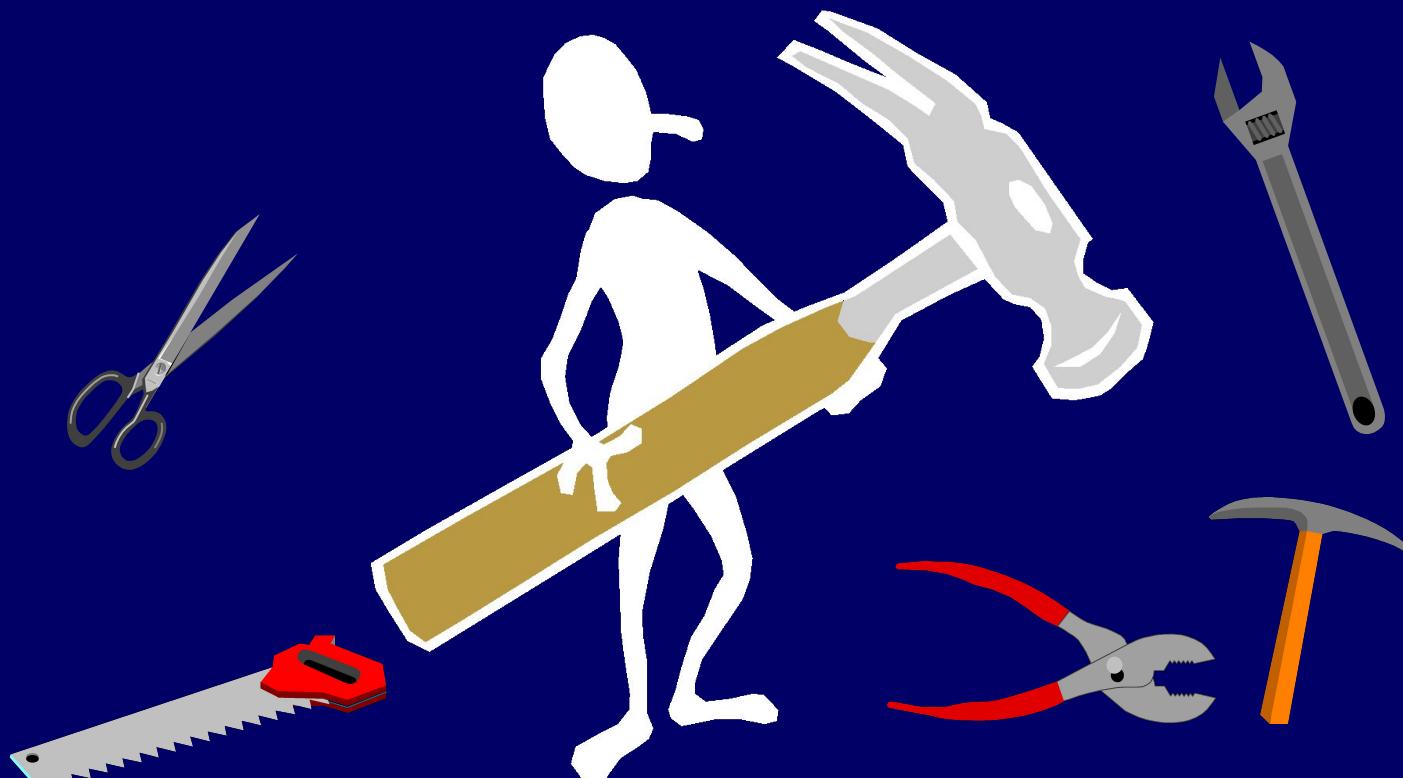
Contenido del curso

- Una lista de algoritmos.
 - Aprender su código.
 - Realizar la traza hasta que se convenzan de que funcionen.
 - Implementarlos.

```
class InsertionSortAlgorithm extends SortAlgorithm
{
    void sort(int a[]) throws Exception {
        for (int i = 1; i < a.length; i++) {
            int j = i;
            int B = a[i];
            while ((j > 0) && (a[j-1] > B)) {
                a[j] = a[j-1];
                j--;
            }
            a[j] = B;
        }
    }
}
```

Contenido del curso

- Estudiar las técnicas de diseño de algoritmos.
- Desarrollar el Pensamiento Abstracto.
- Desarrollar nuevos algoritmos para cualquier problema que pueda surgir.



Caso practico, Analizando:

- Se pidió a muchos programadores expertos codificar la búsqueda binaria.



Caso practico, Analizando:

- Se pidió a muchos experimentados programadores codificar la búsqueda binaria.



80% se equivocó

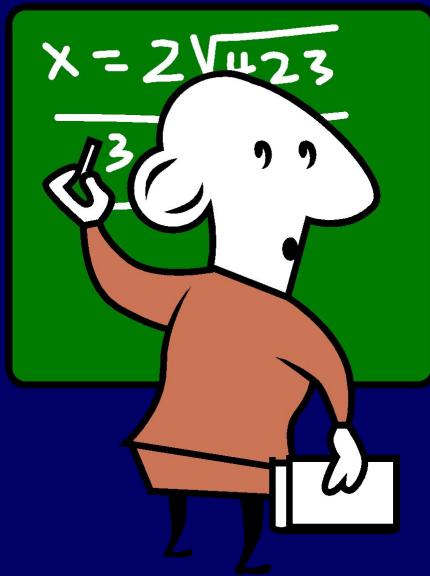
Lo bueno es que no era para una planta de energía nuclear.

¿Qué es lo que les falta?



¿Qué es lo que les falta?

- Tal vez ¿Métodos formales de prueba?



¿Qué es lo que les falta?

- Tal vez ¿Métodos formales de prueba?



Sí, es probable

La industria está empezando a darse cuenta
de que los métodos formales son
importantes.

Pero incluso sin los métodos formales ?

¿Qué es lo que les falta?

- Comprensión fundamental de las técnicas de diseño de algoritmos.
- Pensamiento abstracto.



Contenido del curso

Notaciones, analogías, y abstracciones
para desenvolver,
pensar, y describir algoritmos
para que la corrección sea transparente

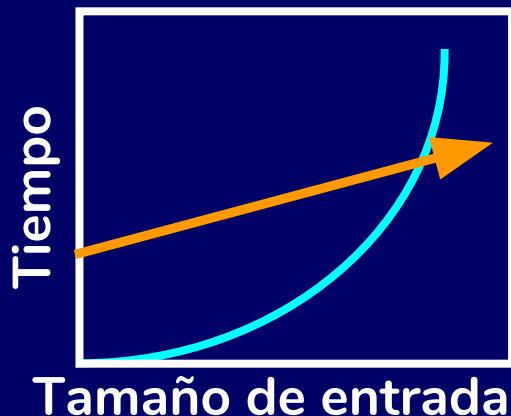
Un estudio de las ideas
fundamentales y técnicas
de diseño de algoritmos

Por ejemplo . . .

Comenzar con un poco de matemáticas

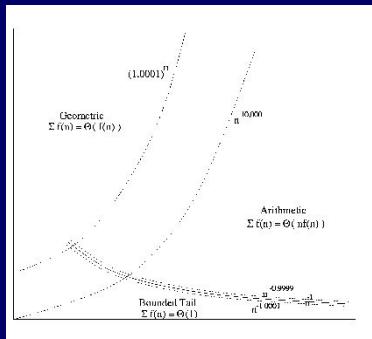
Clasificación de funciones

$$f(i) = n^{\Theta(n)}$$



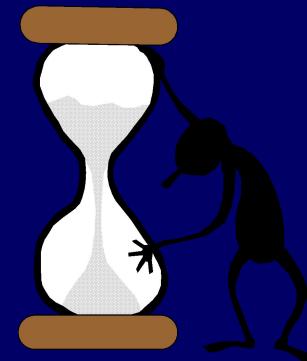
Sumatorias

$$\sum_{i=1} f(i).$$



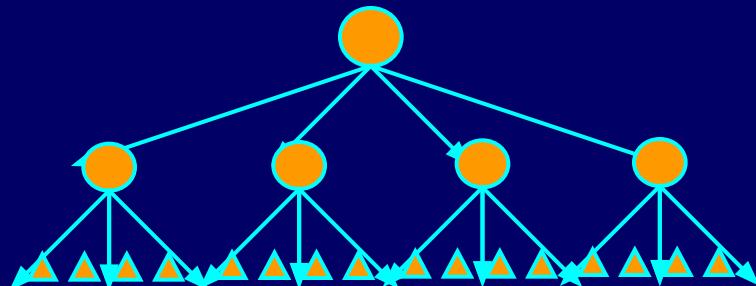
Tiempo y Complejidad

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



Relaciones de recurrencia

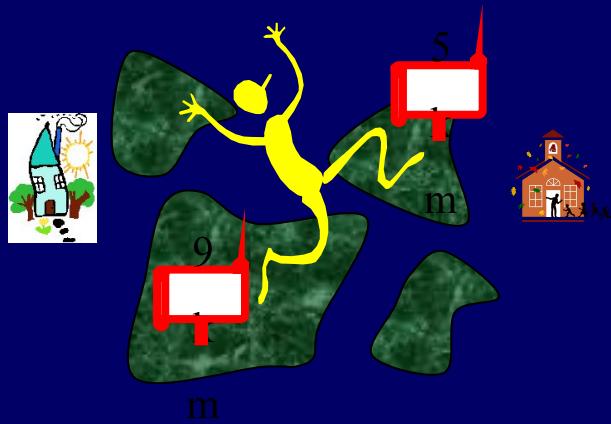
$$T(n) = aT(n/b) + f(n)$$



Los algoritmos iterativos

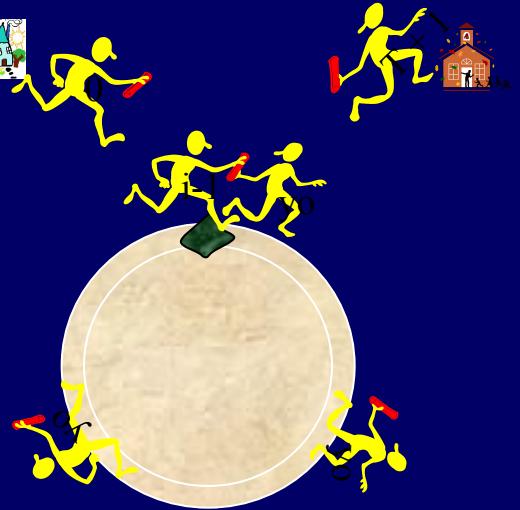
Bucles Invariantes

```
<PreCond>  
CodeA  
loop  
  <Loop-invariante>  
    exit when  
<exitCond>  
  CodeB  
codec  
<PostCond>
```



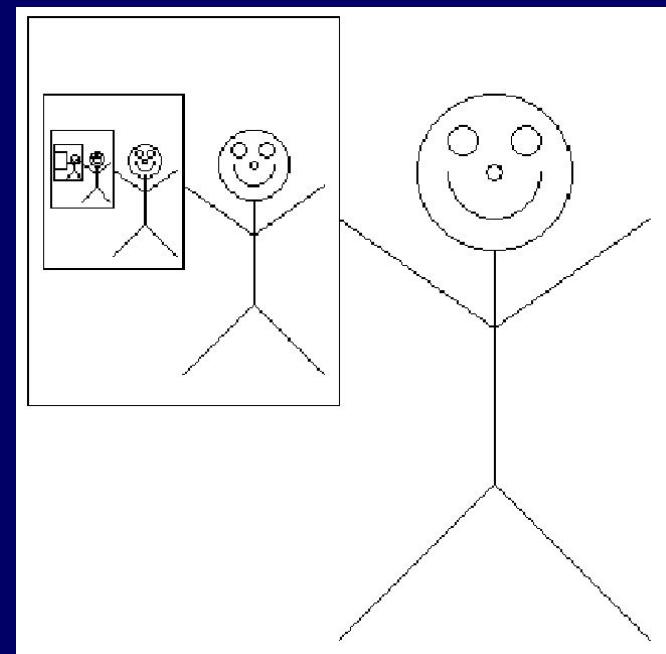
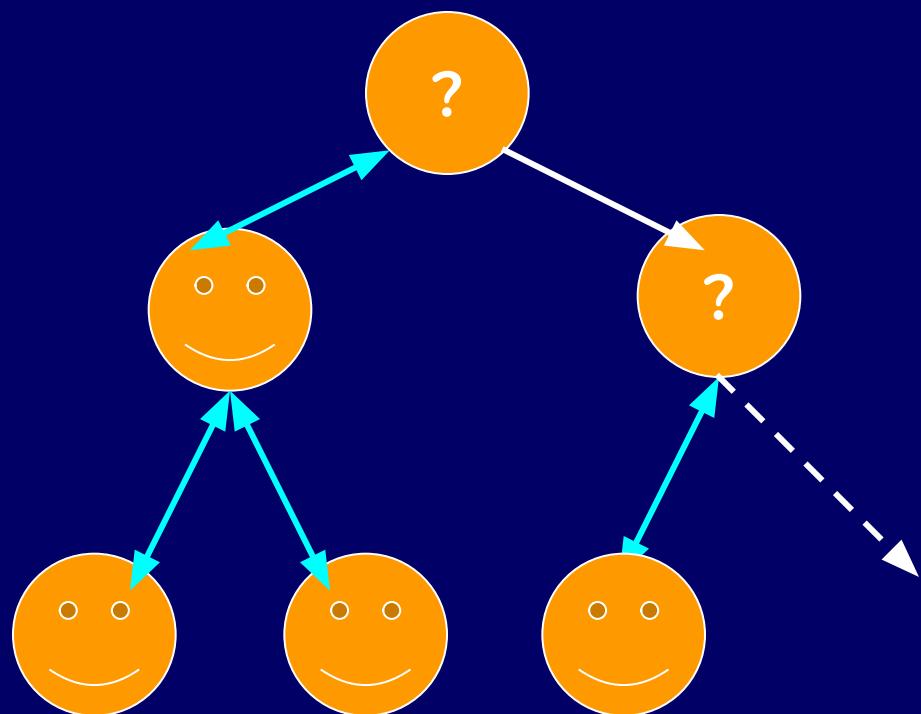
Analizar el
código

División y
Conquista



Caminos
más cortos

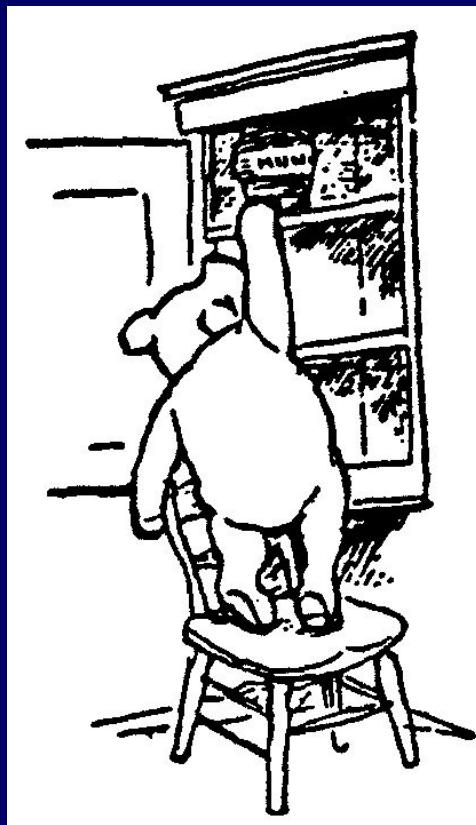
Algoritmos Recursivos



Algoritmos de búsqueda en Grafos



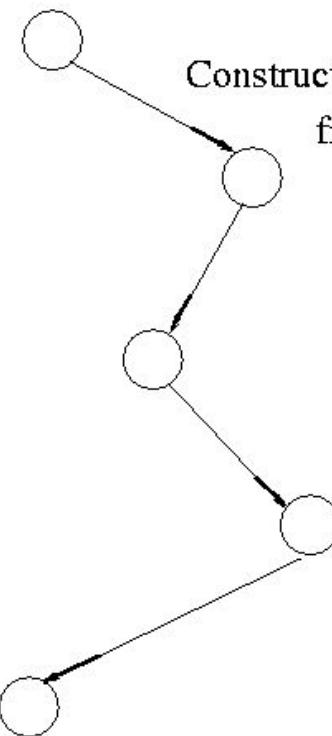
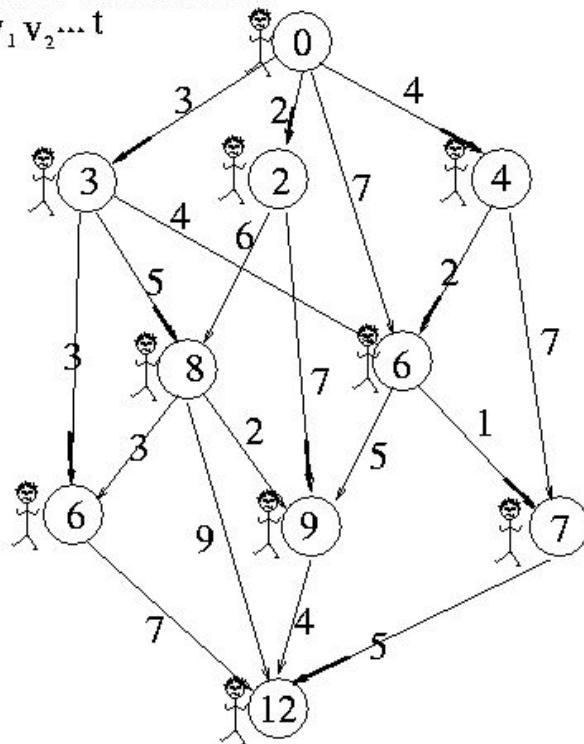
Greedy Algorithms - Algoritmo voraz



Programación dinámica

Solve each subinstance

$s \ v_1 \ v_2 \dots t$



Construct path
from what each
friend stored

j	0	1	2	3	4	5	6	7	8
i	y _j	0	1	0	1	0	1	0	1
0	x _i	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	1	0	1
2	0	0	1	2	1	2	1	2	1
3	0	0	1	2	3	2	3	2	3
4	1	0	1	2	3	2	3	4	3
5	0	0	1	2	3	3	4	4	5
6	1	0	1	2	3	4	4	5	5
7	0	0	1	2	3	4	4	5	6

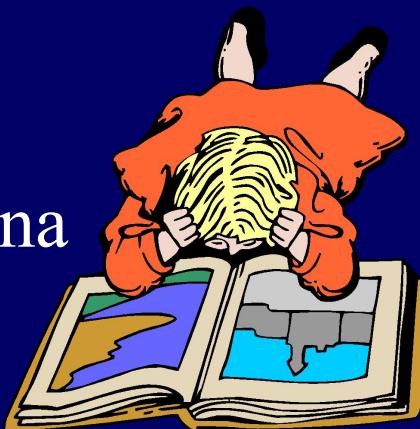
Técnicas de aprendizaje

Leer por adelantado

Se espera que lea los capítulos del curso **antes de** iniciar una aula.

Esto facilitará el entendimiento del tópico y hará más productiva la clase.

Al igual que en una
clase de inglés



Obligatorio!
**desarrollar los ejercicios
que se le encomienda**

Explicando

- Vamos a probar la técnica de aprendizaje “Aprende-Explicando”.
- Por lo tanto, la mejor forma de estudiar es explicar el material una y otra vez en voz alta así mismo, a los demás, y hasta con su oso de peluche.



Ser creativo

- Hacer preguntas.
- Proponer ideas originales.
- ¿Por qué se hace de esta manera y no de esta forma?



Conjeturas y contraejemplos

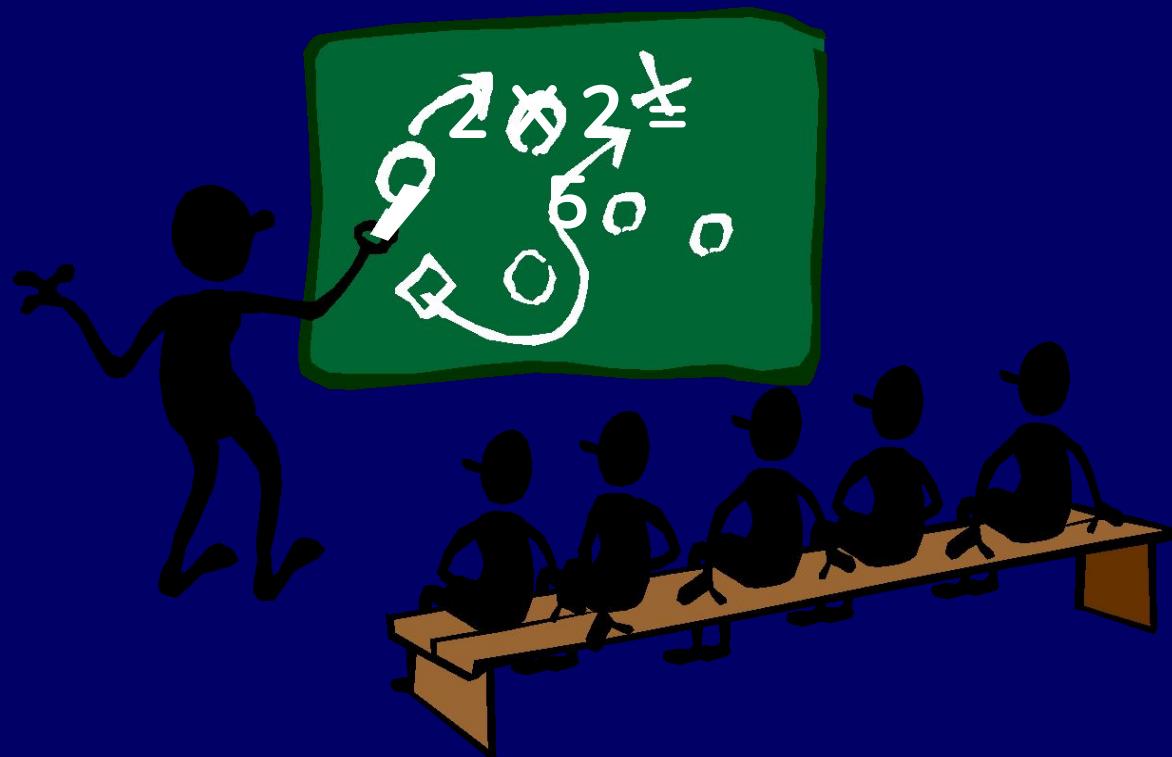
- Proponer algoritmos eficientes para resolver un problema.
- Evaluar todos los casos, es decir, observar para que casos de entrada el algoritmo da respuestas equivocadas.

Refinamiento:
La mejor solución viene de un proceso de
refinamiento de las posibles soluciones.



Algunos ejemplos de Algoritmos

Recordando la primaria ☺
¿Cómo multiplicar dos números?



Números complejos

- Recordando: ¿Cómo multiplicar 2 números complejos?
 - $(a + bi)(c + di) = [ac - bd] + [ad + bc]i$
 - **De entrada:** a, b, c, d **Salida:** ac-bd, ad + bc
- Si una multiplicación cuesta \$1 y una adición un centavo.
- ¿Proponer la forma más barata para obtener el resultado?
- ¿Se puede realizar en menos de \$4.02?

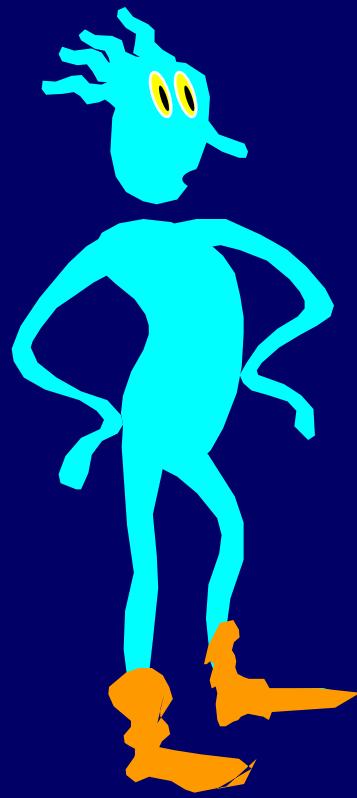
Método de Gauss \$3.05:

De entrada: a, b, c, d Salida: $ac-bd, ad + bc$

- $m_1 = ac$
- $m_2 = bd$
- $A_1 = m_1 - m_2 = ac - bd$
- $m_3 = (a + b)(c + d) = ac + \cancel{ad} + bc + \cancel{bd}$
- $A_2 = m_3 - m_1 - m_2 = ad + bc$

Pregunta:

- El método de Gauss resuelve la operación con un 25% menos de trabajo.
- ¿Podría haber algún método donde la operación se realice con menos costo?



Odette

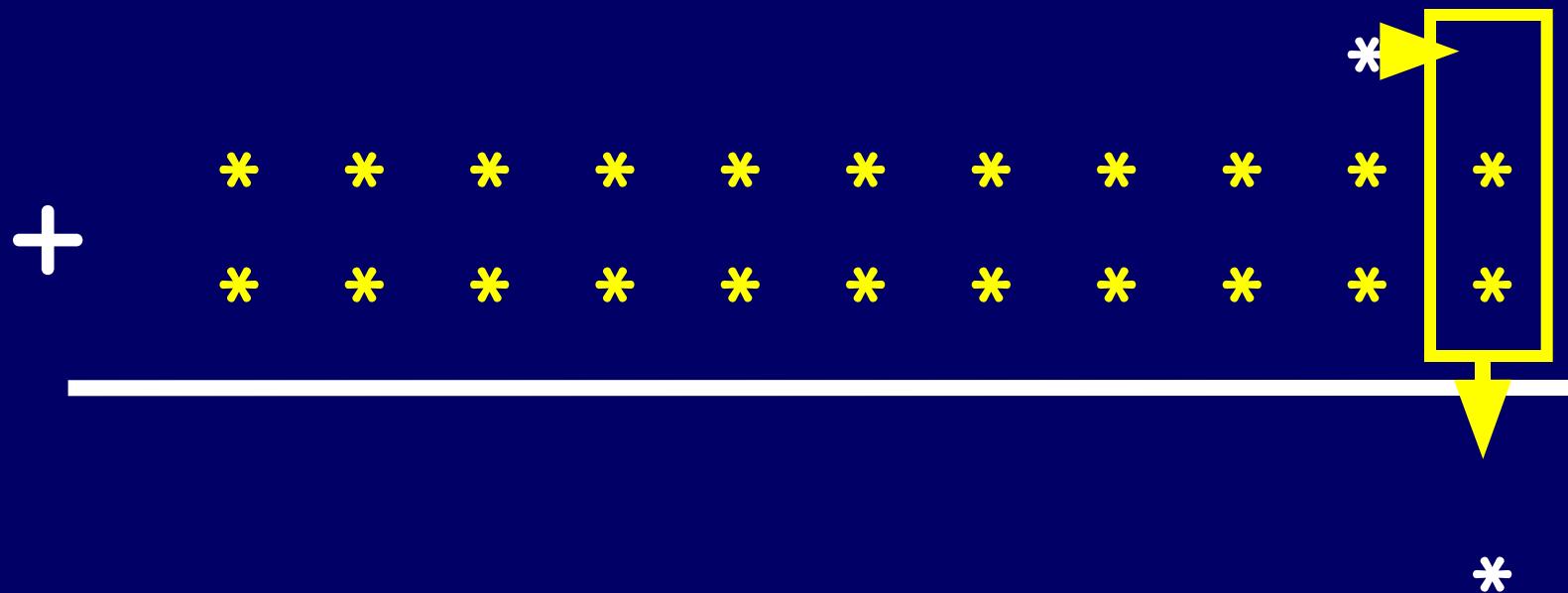


Bonzo

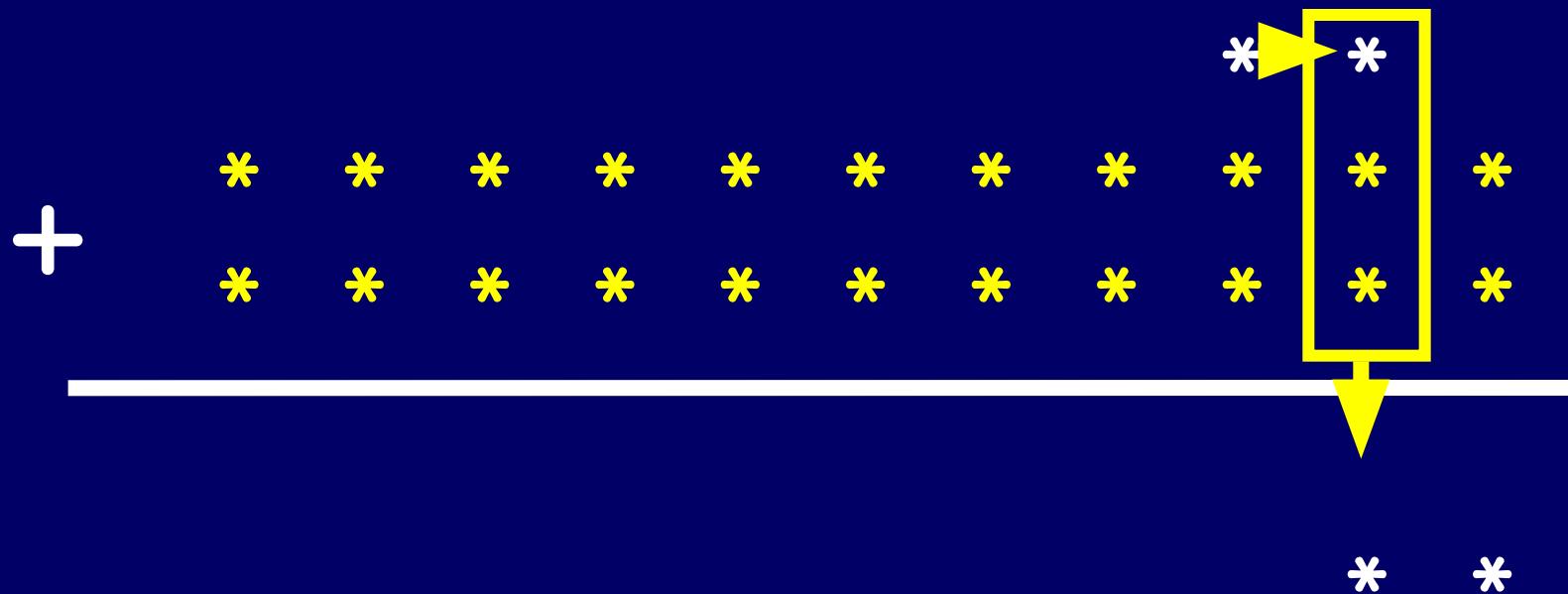
Cómo sumar 2 números de n bits.

$$\begin{array}{r} * * * * * * * * * * \\ + * * * * * * * * * * \\ \hline \end{array}$$

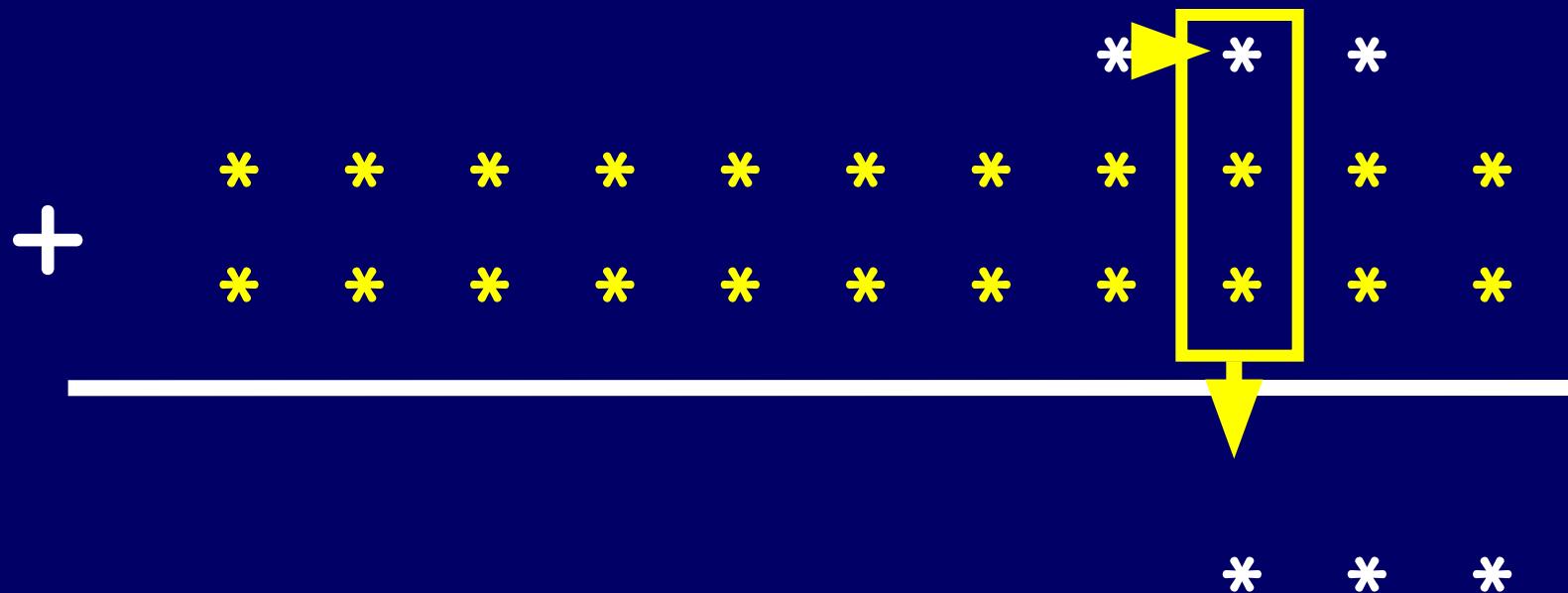
Cómo sumar 2 números de n bits.



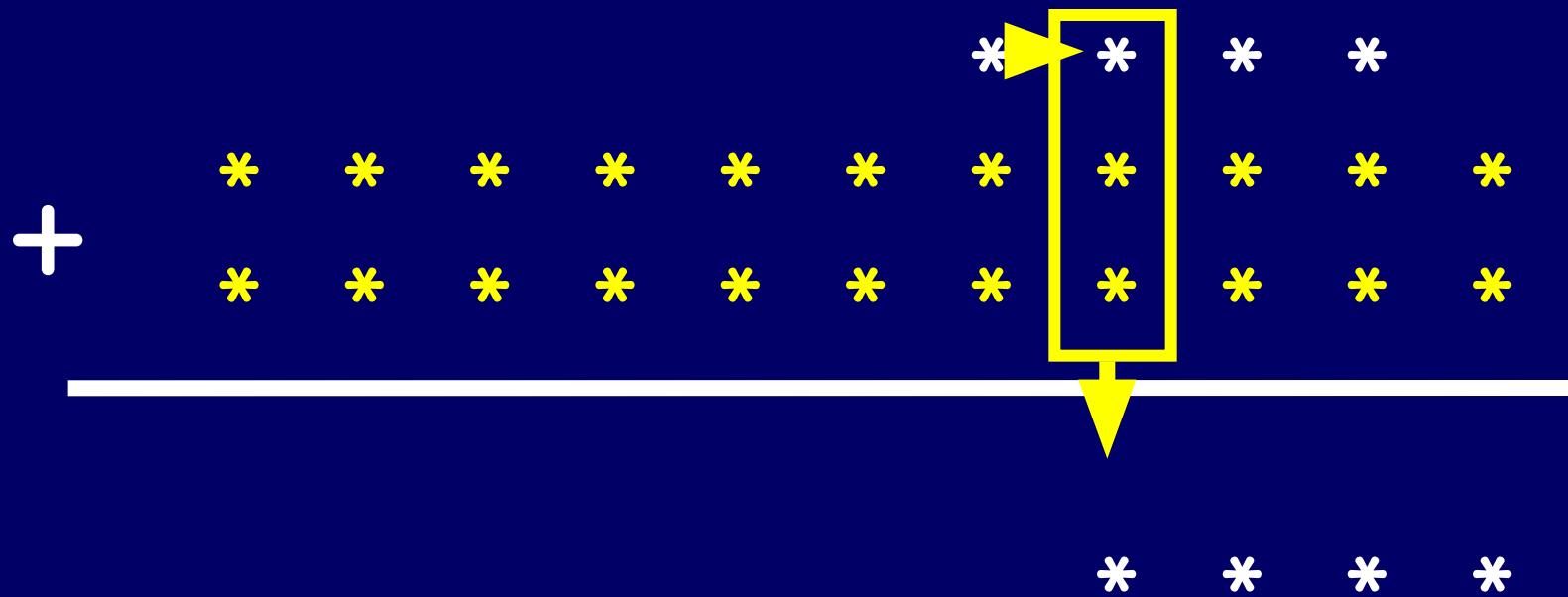
Cómo sumar 2 números de n bits.



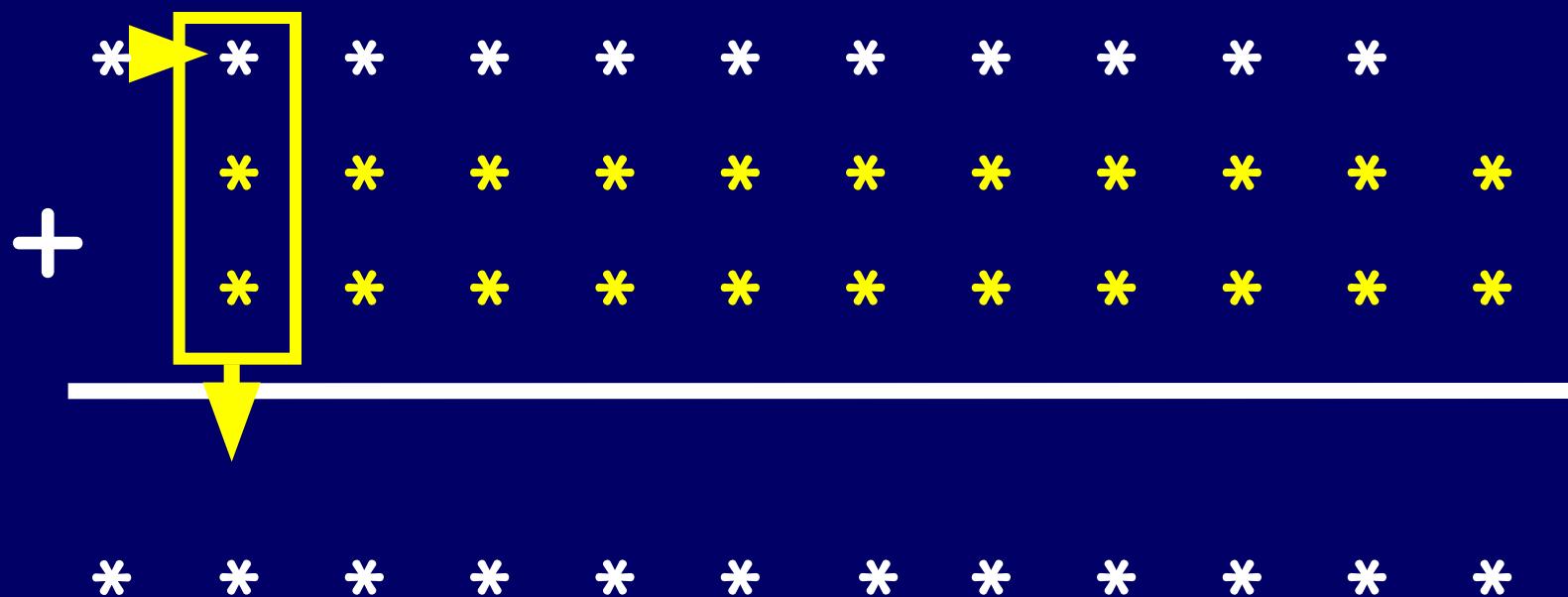
Cómo sumar 2 números de n bits.



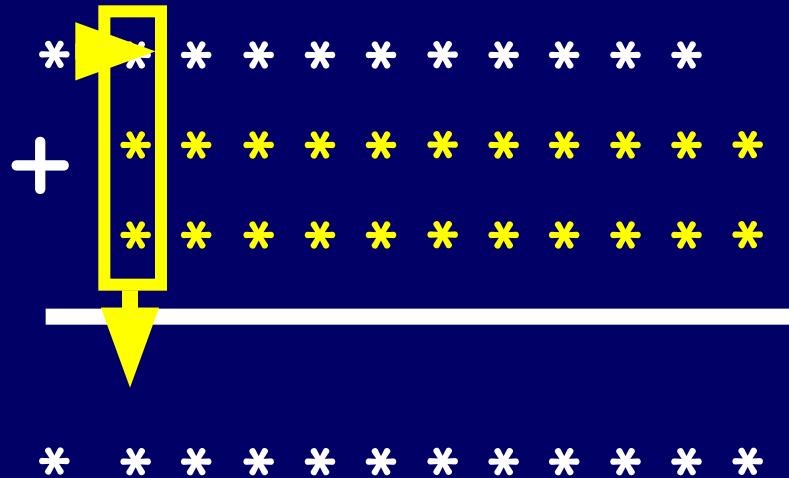
Cómo sumar 2 números de n bits.



Cómo sumar 2 números de n bits.



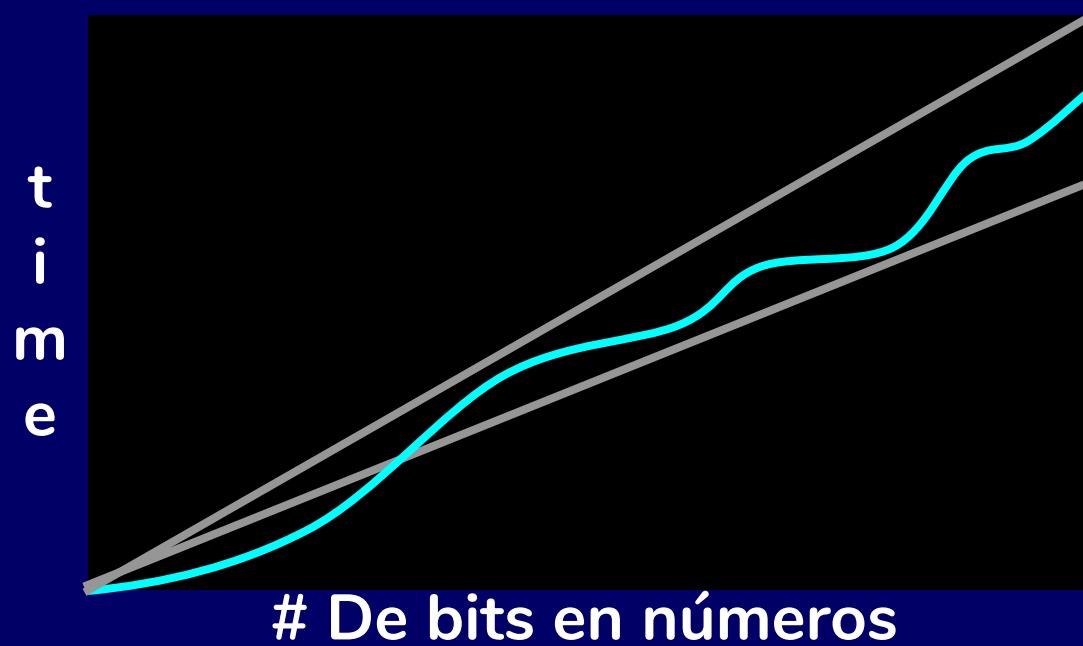
Tiempo de complejidad de la suma (método de la escuela primaria)



La suma de 3 bits se puede hacer en un tiempo constante.

$T(n) =$ La cantidad de “sumas” en tiempo que se utiliza para sumar dos números de n bits
 $= \theta(n) =$ tiempo lineal.

$f = \Theta(N)$ significa que f puede ser intercalado entre dos líneas



Por favor siéntase libre de
ípreguntar!



¿Hay una manera más rápida para sumar?

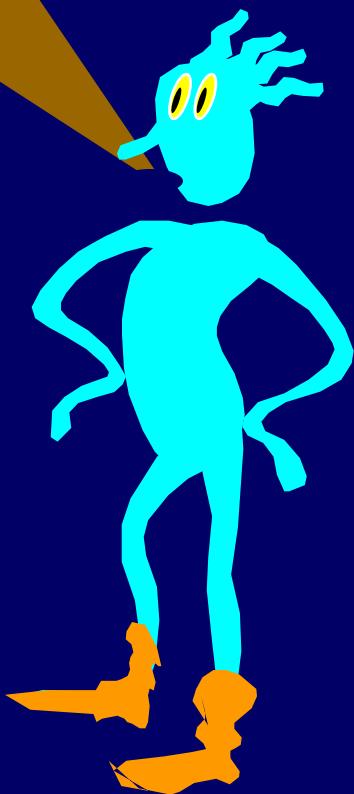
- **PREGUNTA:** ¿Existe un algoritmo para sumar dos números de n bits cuyo tiempo crece linealmente menor a “ n ”?

Cualquier algoritmo para la adición debe leer todos los bits de entrada

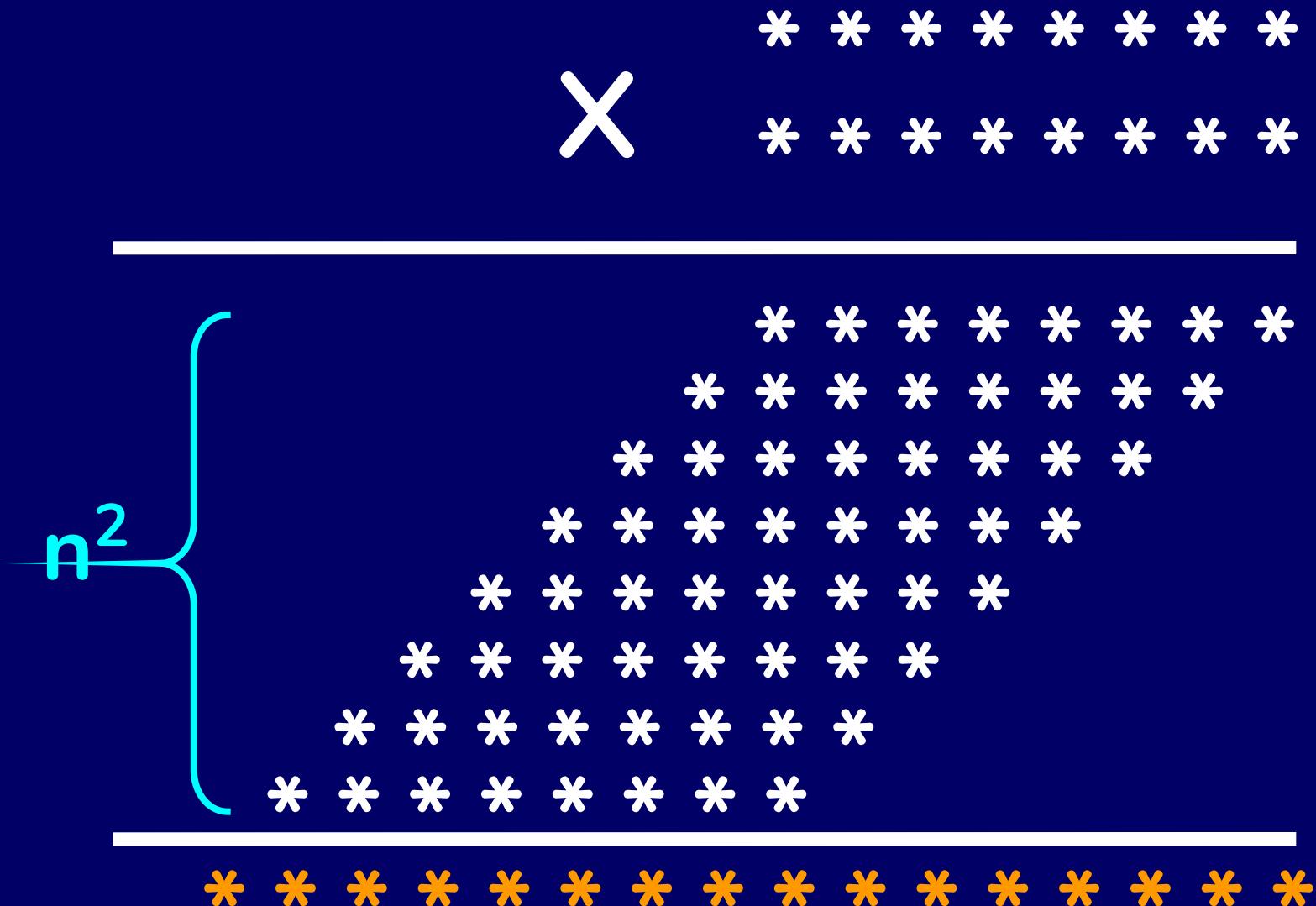
- (Hipótesis) Supongamos que existe un algoritmo “misterioso” que no examina cada bit.
- Dar al algoritmo un par de números de n bits.
- El algoritmo “misterioso” implica que existirá un bit que no será examinado.
- Si cambiamos el bit que no fue examinado, tendríamos la misma respuesta final.
- Entonces la hipótesis queda descartada, por que, si mudamos un solo bit, el resultado debería ser diferente.

Así que **cualquier algoritmo** para la adición debe usar el tiempo de al menos lineal en el tamaño de los números.

El método que me enseñaron en la escuela es realmente bueno :)



Como multiplicar 2 números de n bits.



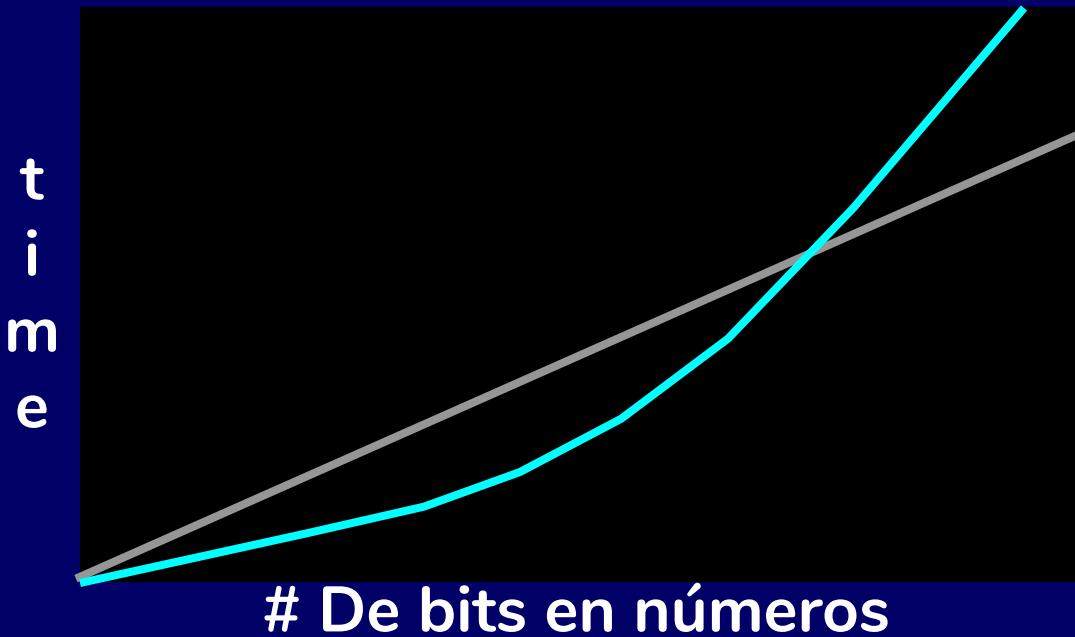
How to multiply 2 n-bit numbers.

$$\begin{array}{r} \times \quad * * * * * * * \\ * * * * * * * \\ \hline n^2 \left\{ \begin{array}{l} * * * * * * * \\ * * * * * * * \\ * * * * * * * \\ * * * * * * * \\ * * * * * * * \\ * * * * * * * \\ * * * * * * * \\ * * * * * * * \end{array} \right. \\ \hline * * * * * * * * * * * * * \end{array}$$

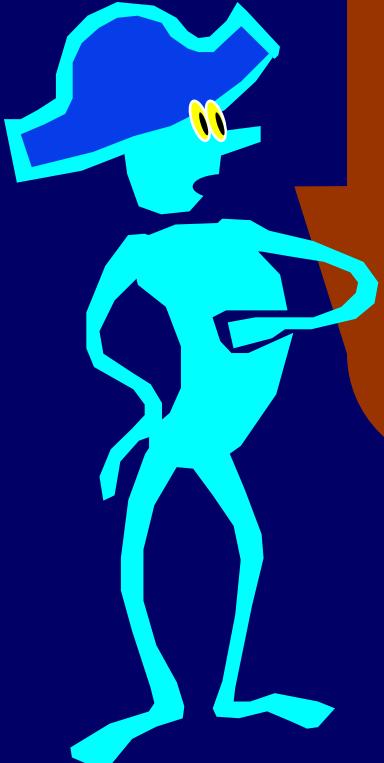
¡Lo entiendo! El tiempo total está limitada por cn^2 .



Adición: El tiempo lineal
Multiplicación: tiempo cuadrática



No importa qué tan dramática sea la diferencia entre constantes, la **curva cuadrática** finalmente dominará a la **curva lineal**



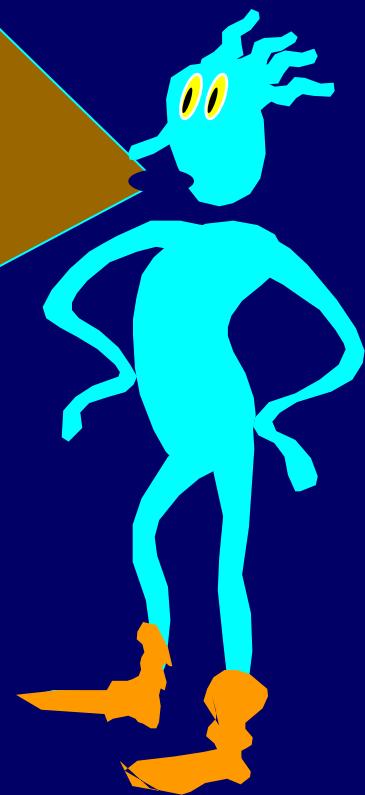
¡Ordenado! Hemos demostrado que la multiplicación consume más tiempo que la adición.

Las matemáticas confirmaron nuestro sentido común.

En Conclusión!

Hemos argumentado que la multiplicación utiliza más tiempo que la adición. Esto mediante la comparación de las complejidades de los dos algoritmos.

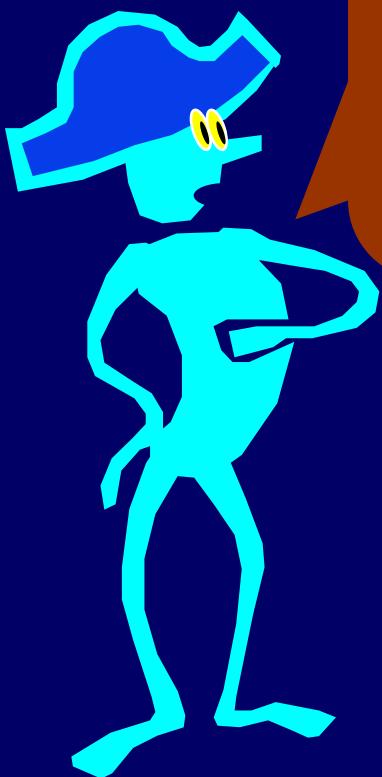
Para argumentar que la multiplicación es inherentemente más difícil que la adición, habría que demostrar que no existe un algoritmo de la multiplicación que se ejecute en tiempo lineal.



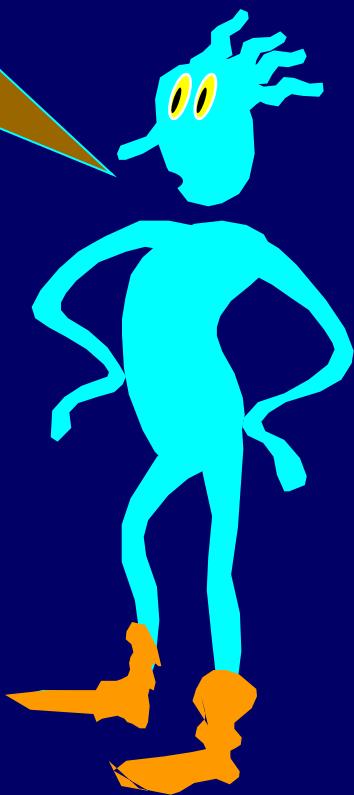
Adición: $\theta(n)$ tiempo

Multiplicación: $\theta(n^2)$ tiempo

¿Existe un algoritmo para multiplicar dos números en tiempo lineal?



A pesar de años de investigación, no se sabe! Si resuelve esta cuestión, York le dará un doctorado!





¿Hay una manera más rápida para multiplicar dos números de la forma en que ha aprendido en la escuela primaria?

Divide y conquistarás

(Un enfoque para los algoritmos más rápidos)

- **DIVIDIR** el problema en instancias pequeñas de la misma naturaleza.
- **Tener un amigo** (Recursivamente) que me ayude a resolverlos.
- **UNIR** las subrespuestas con el fin de obtener la respuesta a la instancia general.



Multiplicación de 2 números de n bits

- $X = \begin{array}{|c|c|} \hline a & b \\ \hline \end{array}$
- $Y = \begin{array}{|c|c|} \hline c & d \\ \hline \end{array}$

$$\bullet X = a2^{n/2} + b \quad Y = c2^{n/2} + d$$

$$\bullet XY = ac2^n + (ad+bc)2^{n/2} + bd$$

Multiplicación de 2 números de n bits

$$\begin{array}{l} X = \begin{array}{|c|c|} \hline a & b \\ \hline \end{array} \\ Y = \begin{array}{|c|c|} \hline c & d \\ \hline \end{array} \end{array}$$

$$XY = ac2^n + (ad+bc)2^{n/2} + bd$$

MULT(X, Y):

If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

RETURN

$$\text{MULT}(a,c)2^n + (\text{MULT}(a,d) + \text{MULT}(b,c))2^{n/2} + \text{MULT}(b,d)$$

Tiempo requerido por MULT

- $T(n)$ = tiempo empleado por MULT en dos números de n bits
- ¿Qué es $T(n)$?
¿Cuál es su tasa de crecimiento?
¿Es $\theta(n^2)$?

Relación de Recurrencia

- $T(1) = k$ constante
- $T(n) = 4T(n/2) + k'n + k''$ para algunas constantes k' y k''

MULT(X, Y):

If $|X| = |Y| = 1$ then RETURN XY

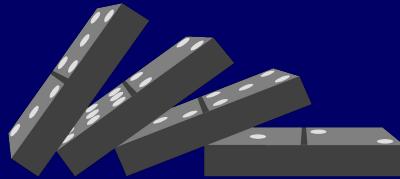
Break X into a;b and Y into c;d

RETURN

$$\text{MULT}(a,c)2^n + (\text{MULT}(a,d) + \text{MULT}(b,c))2^{n/2} + \text{MULT}(b,d)$$

Resumiendo la R.R.

- $T(1) = 1$
- $T(n) = 4T(n/2) + n$
- ¿Cómo podemos despejar $T(n)$ de manera que podamos determinar su tasa de crecimiento?



Técnica 1: Guess and verify

- Relación de recurrencia:

$$T(1) = 1 \text{ y } T(n) = 4T(n/2) + n$$

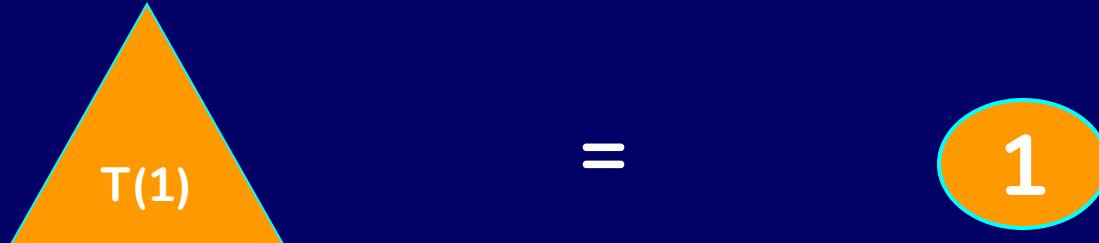
- Guess: $G(n) = 2n^2 - n$

- Verify:

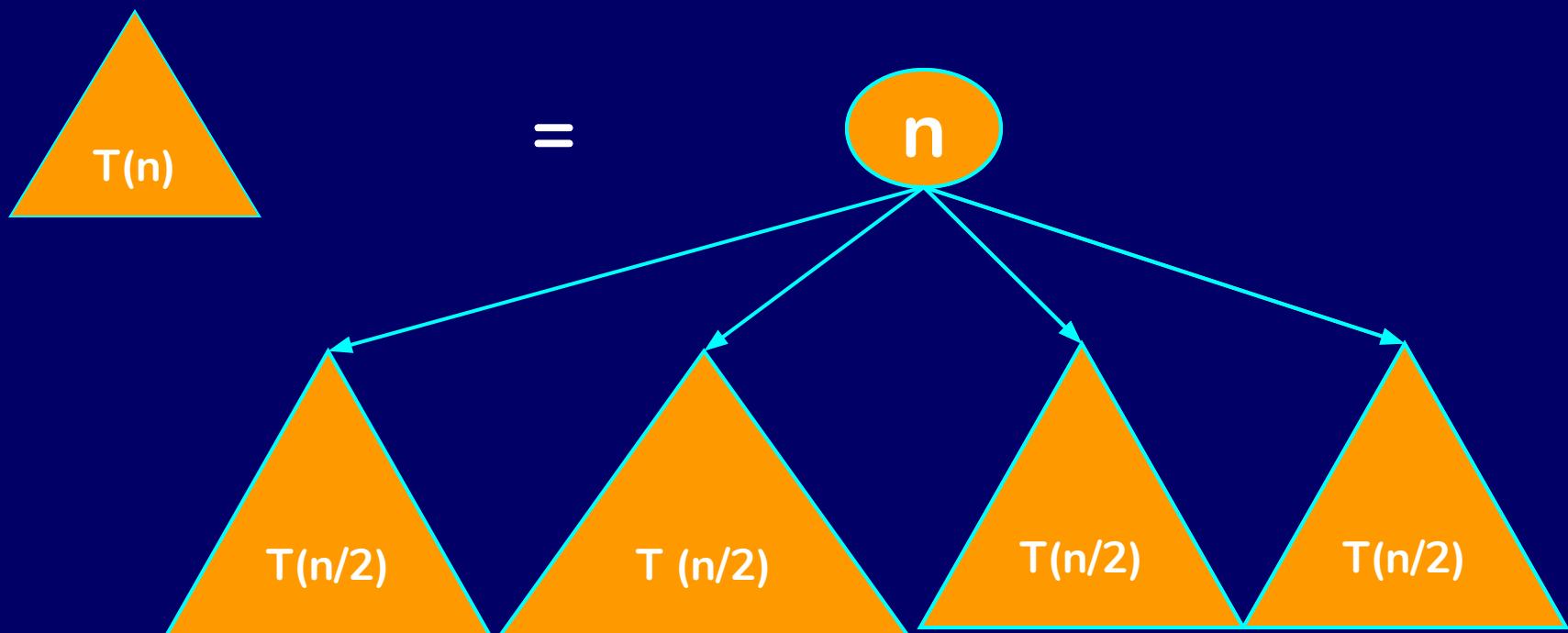
Lado izquierdo	Lado derecho
$T(1) = 2(1)^2 - 1$	1
$T(n) = 2n^2 - n$	$4T(n/2) + n$ $= 4 [2(n/2)^2 - (n/2)] + n$ $= 2n^2 - n$

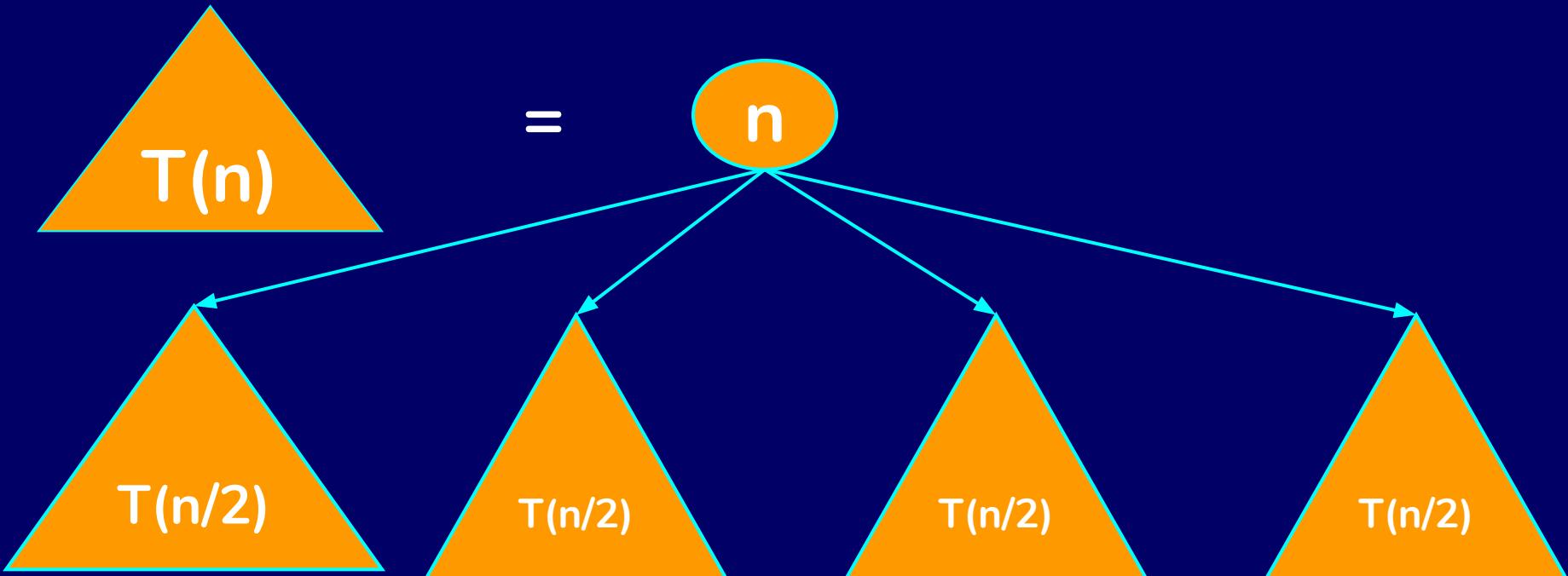
Técnica 2: diseñando el árbol de recurrencia

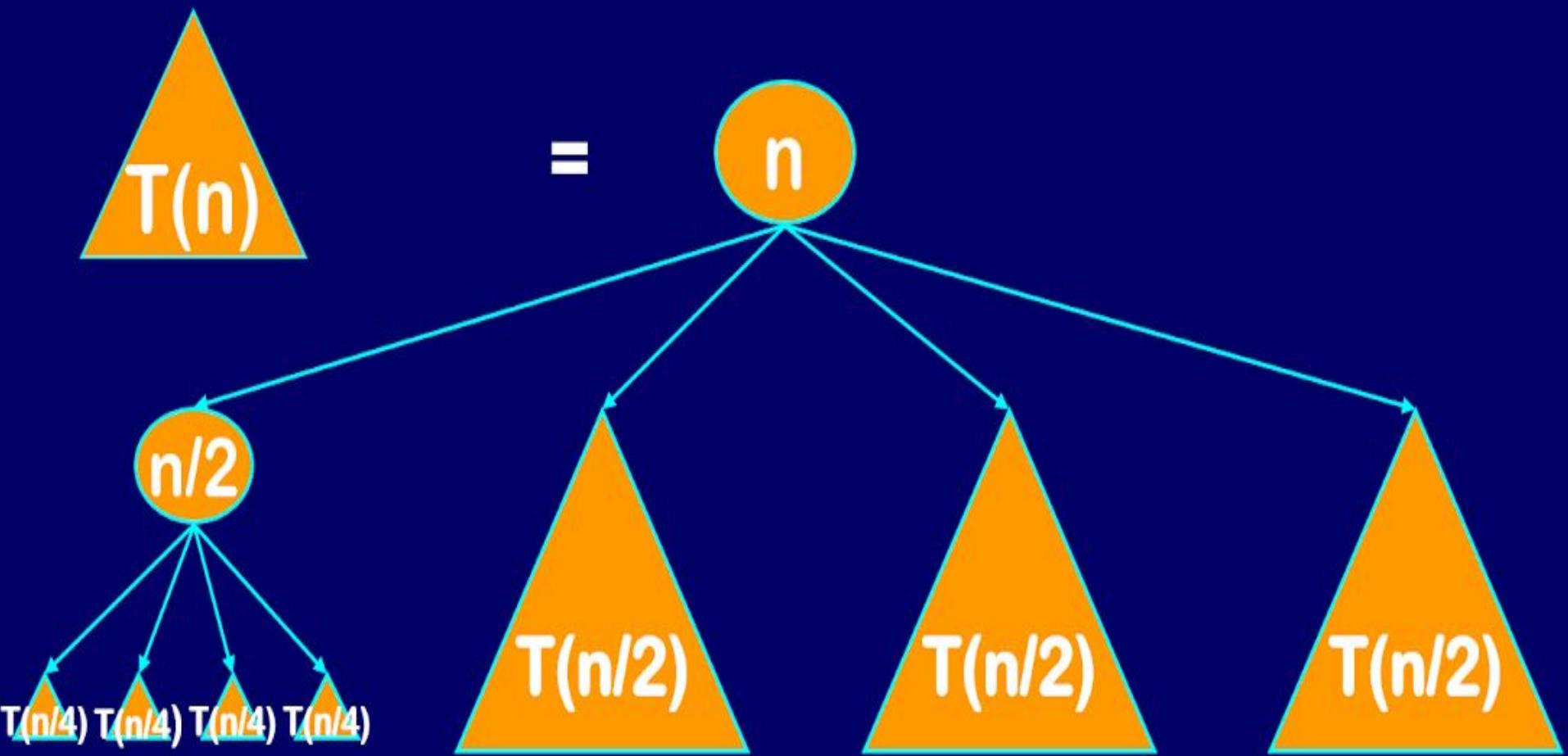
$$\cdot \underline{T(1)} \quad \equiv \quad 1$$

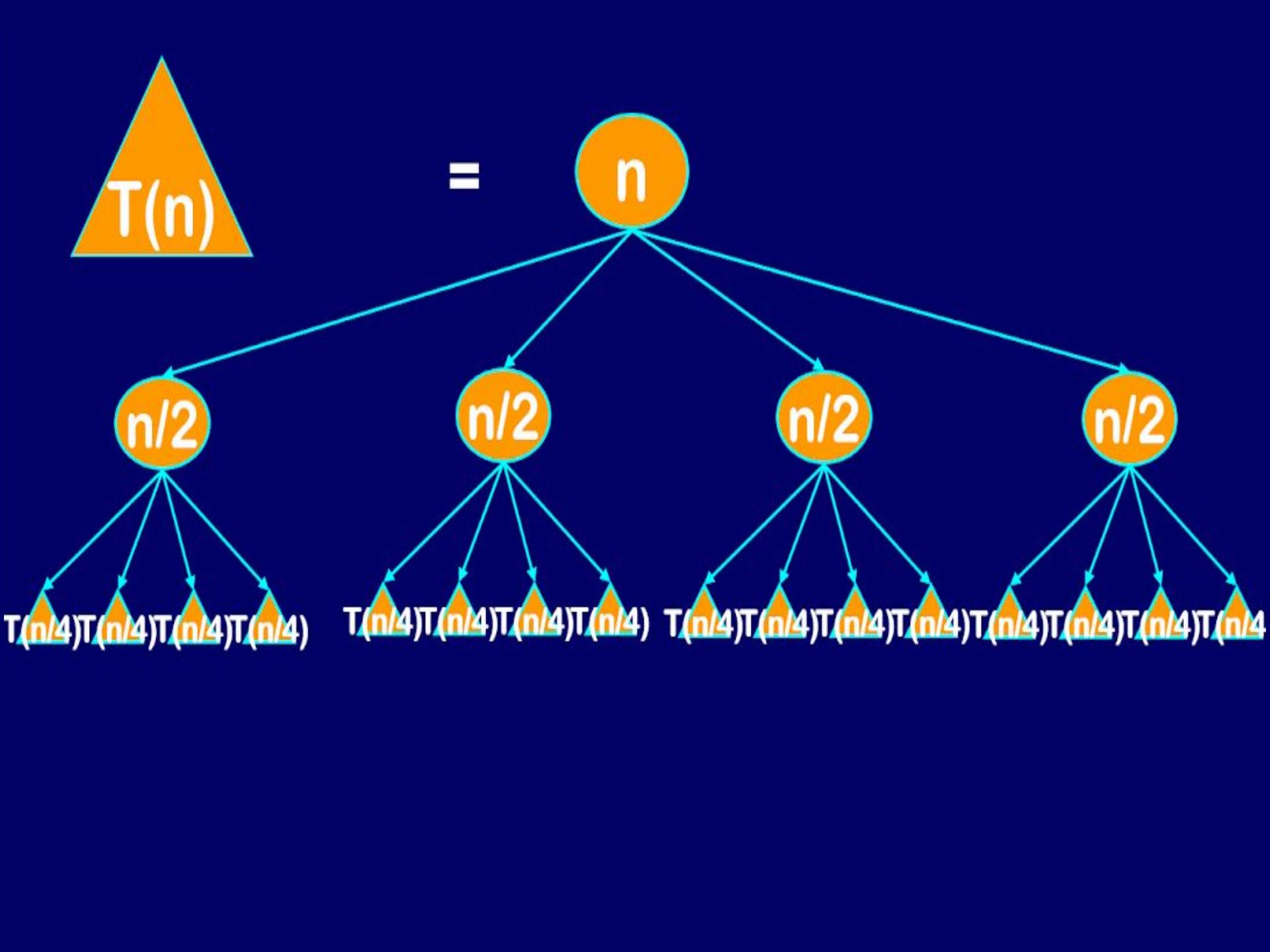


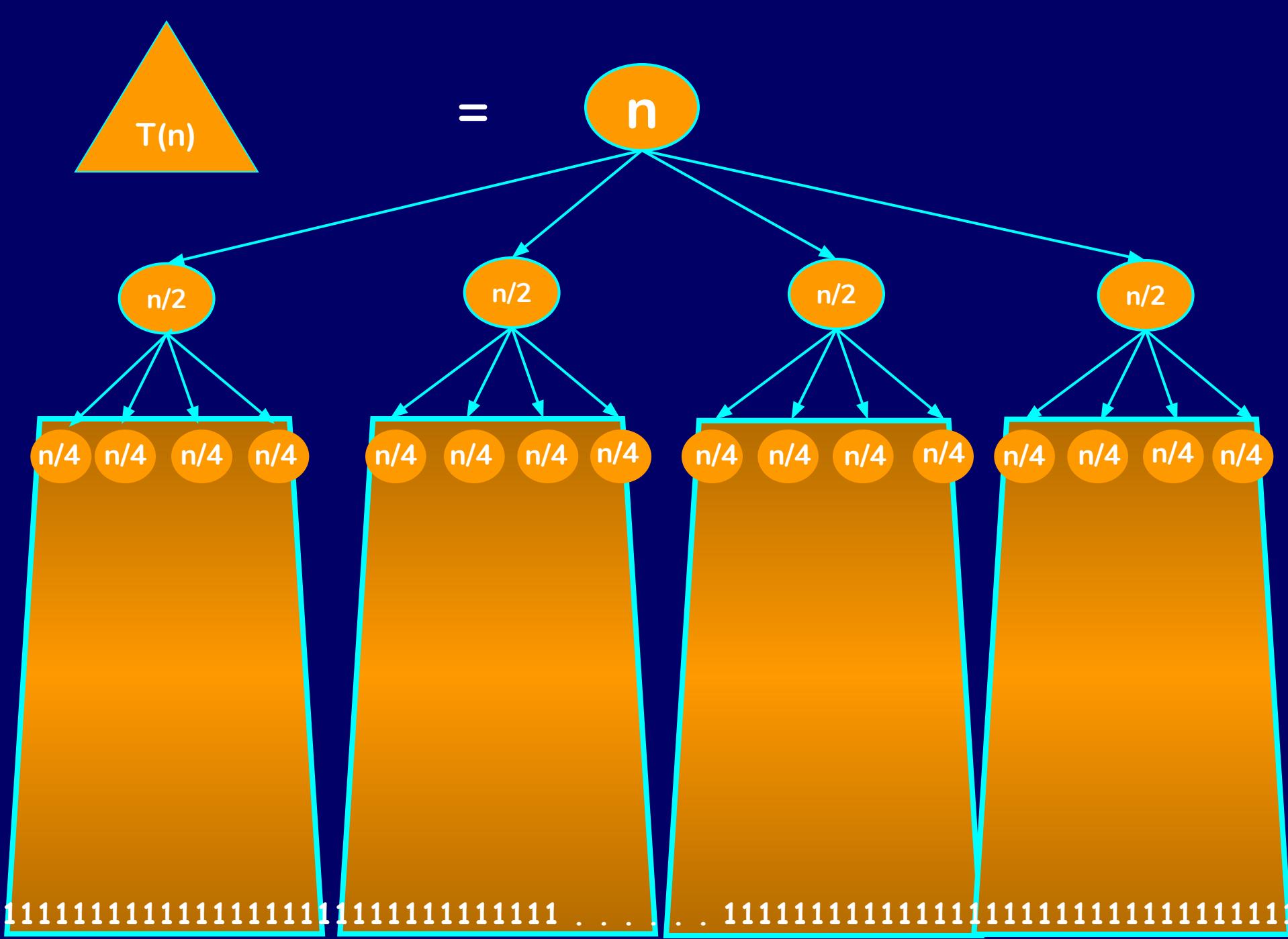
$$\cdot \underline{T(n)} \quad = \quad \underline{n + 4 T(n / 2)}$$

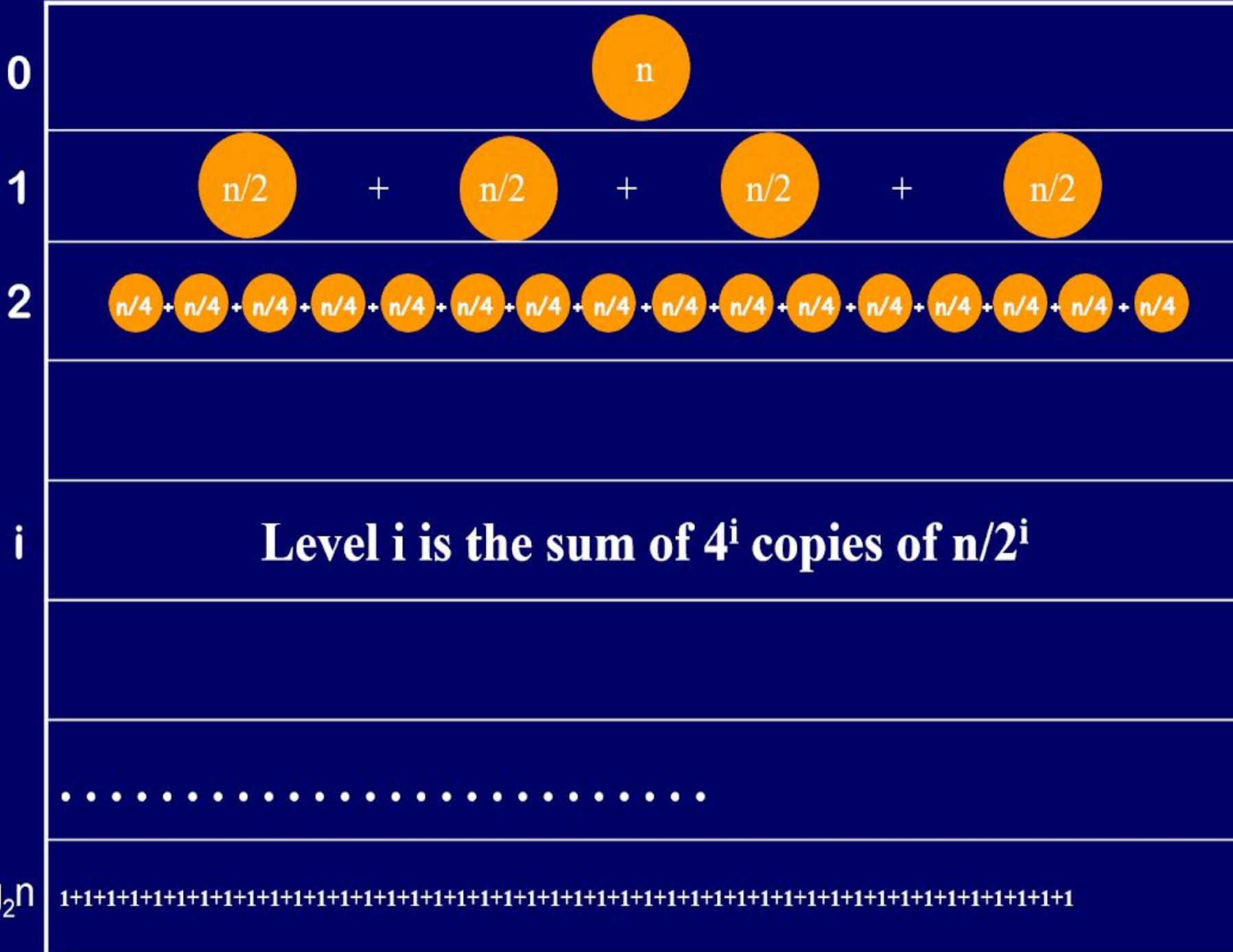




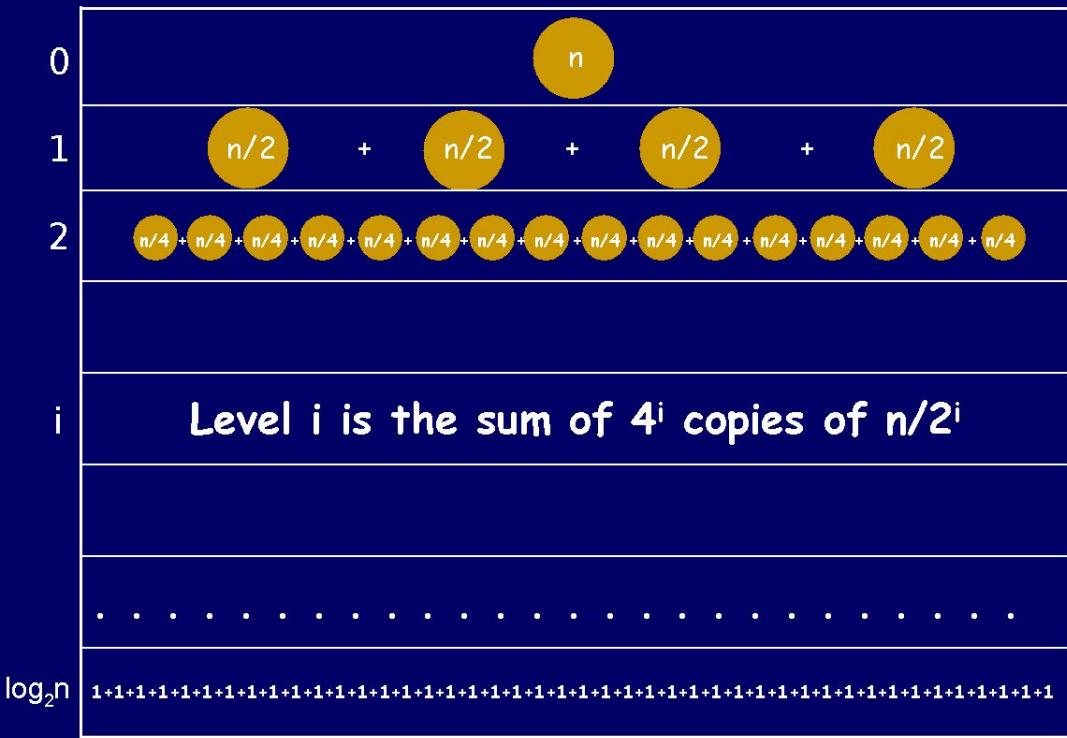








Level i is the sum of 4^i copies of $n/2^i$



=1·n

$$= 4 \cdot n / 2$$

$$= 16 \cdot n / 4$$

$$= 4^i \cdot n / 2^i$$

$$= 4^{\log n} \cdot n / 2^{\log}$$

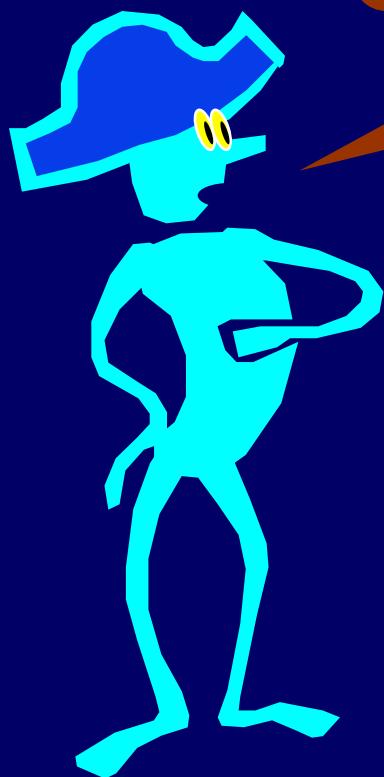
$$= n^{\log 4 \cdot 1}$$

Total: $\theta(n^{\log 4}) = \theta(n^2)$

2da Técnica Divide y Vencerás: tiempo $\Theta(n^2)$

1ra Técnica: tiempo $\Theta(n^2)$

¡Todo este trabajo para nada!



Volviendo a revisar

MULT(X,Y):

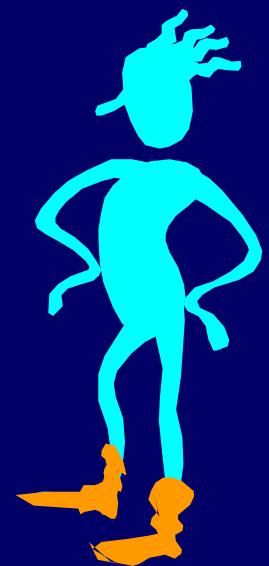
If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

RETURN

$\text{MULT}(a,c) \cdot 2^n + (\text{MULT}(a,d) + \text{MULT}(b,c)) \cdot 2^{n/2} + \text{MULT}(b,d)$

- MULT se llama a sí mismo 4 veces.
¿Se puede reducir el número de llamadas?



Metodo de Gauss:

De entrada: a, b, c, d Salida: ac, ad + bc, bd

- $A_1 = ac$
- $A_3 = bd$
- $m_3 = (a+b)(c+d) = \cancel{ac} + \cancel{ad} + \cancel{bc} + \cancel{bd}$
- $A_2 = m_3 - A_1 - A_3 = ad + bc$

Gaussified MULT (Karatsuba 1962)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

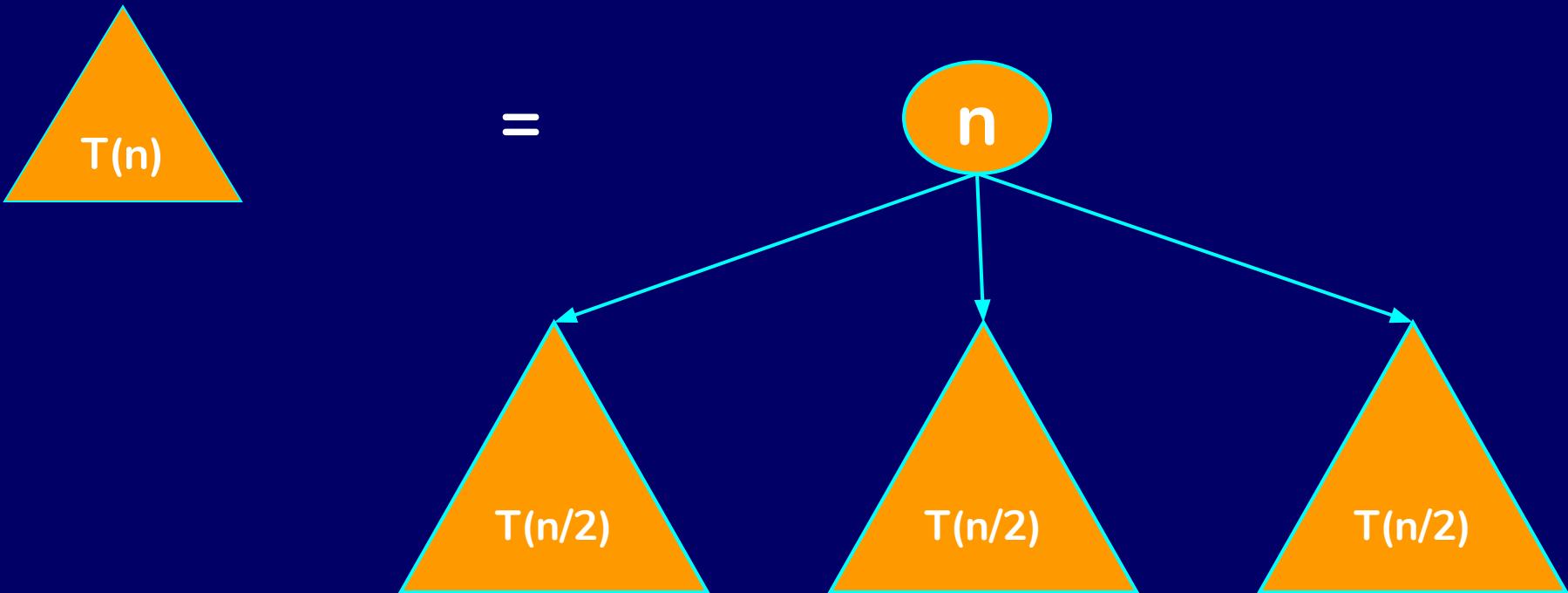
Break X into a;b and Y into c;d

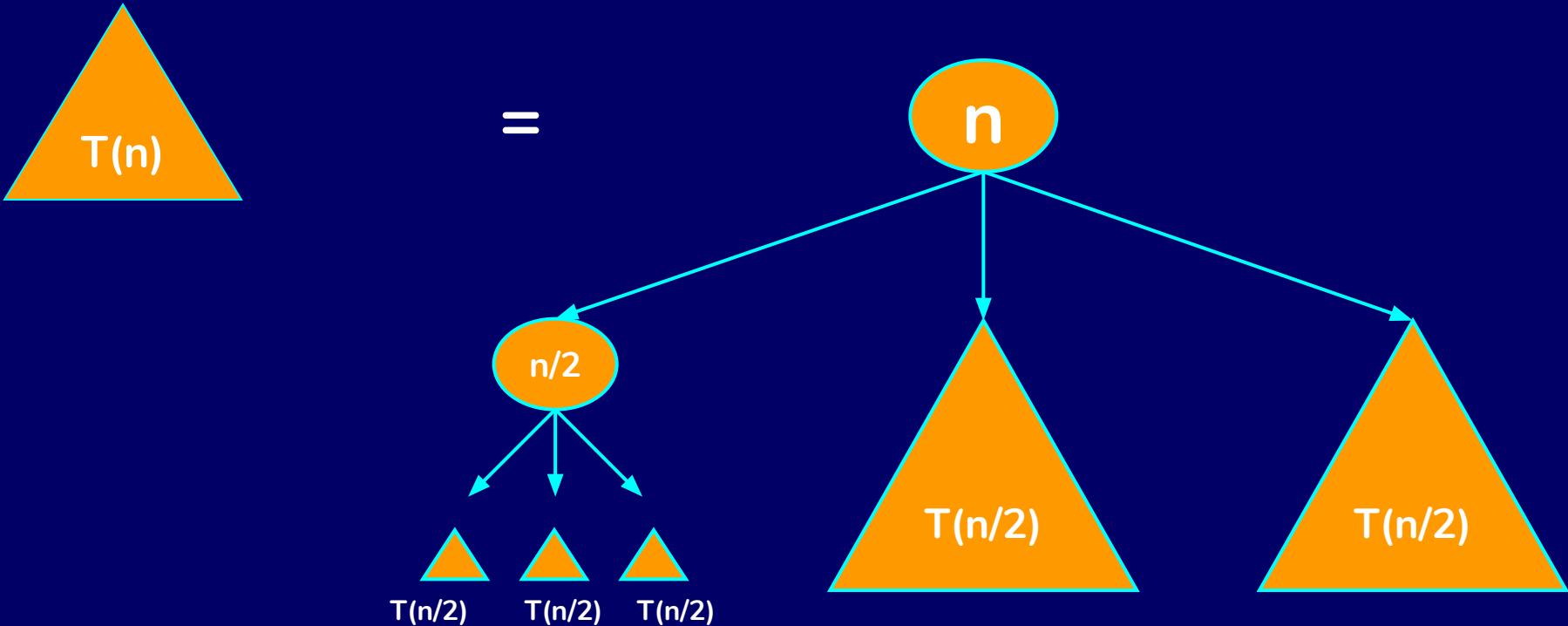
e = MULT(a,c) and f = MULT(b,d)

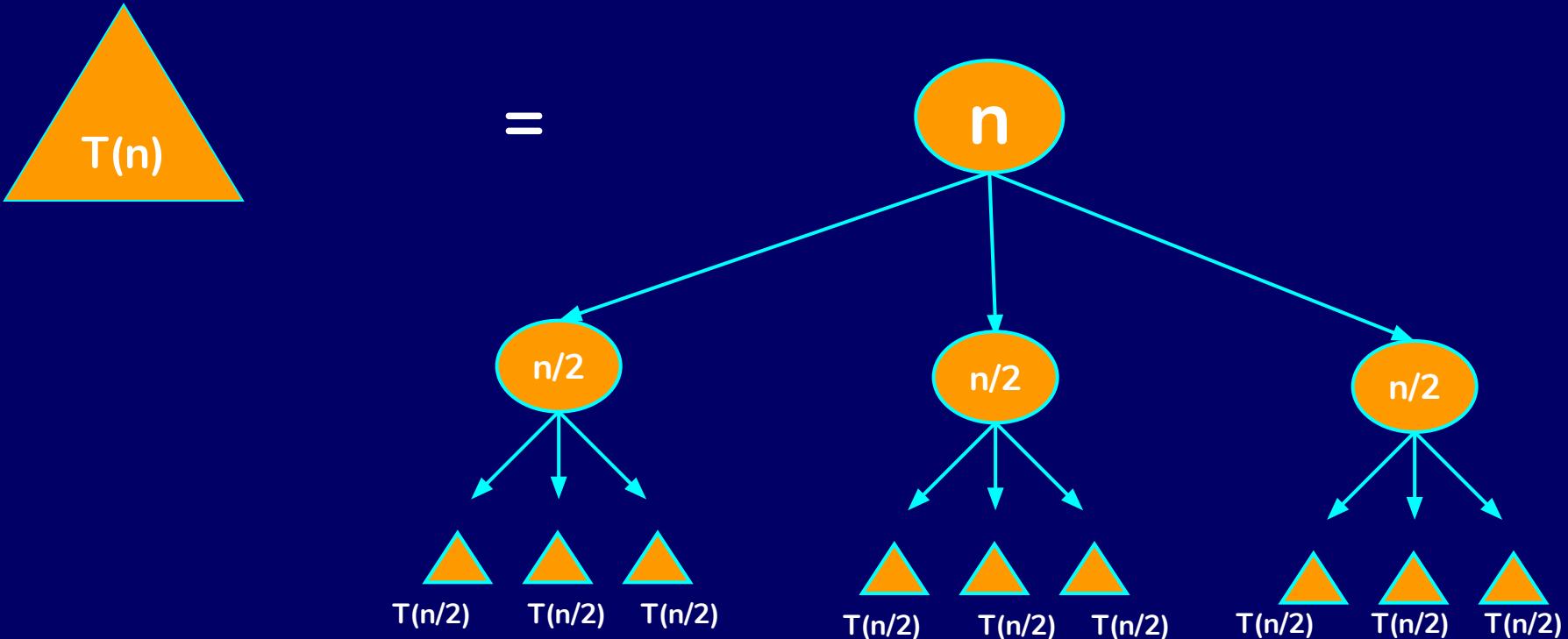
RETURN $e2^n + (\text{MULT}(a+b, c+d) - e - f) 2^{n/2} + f$

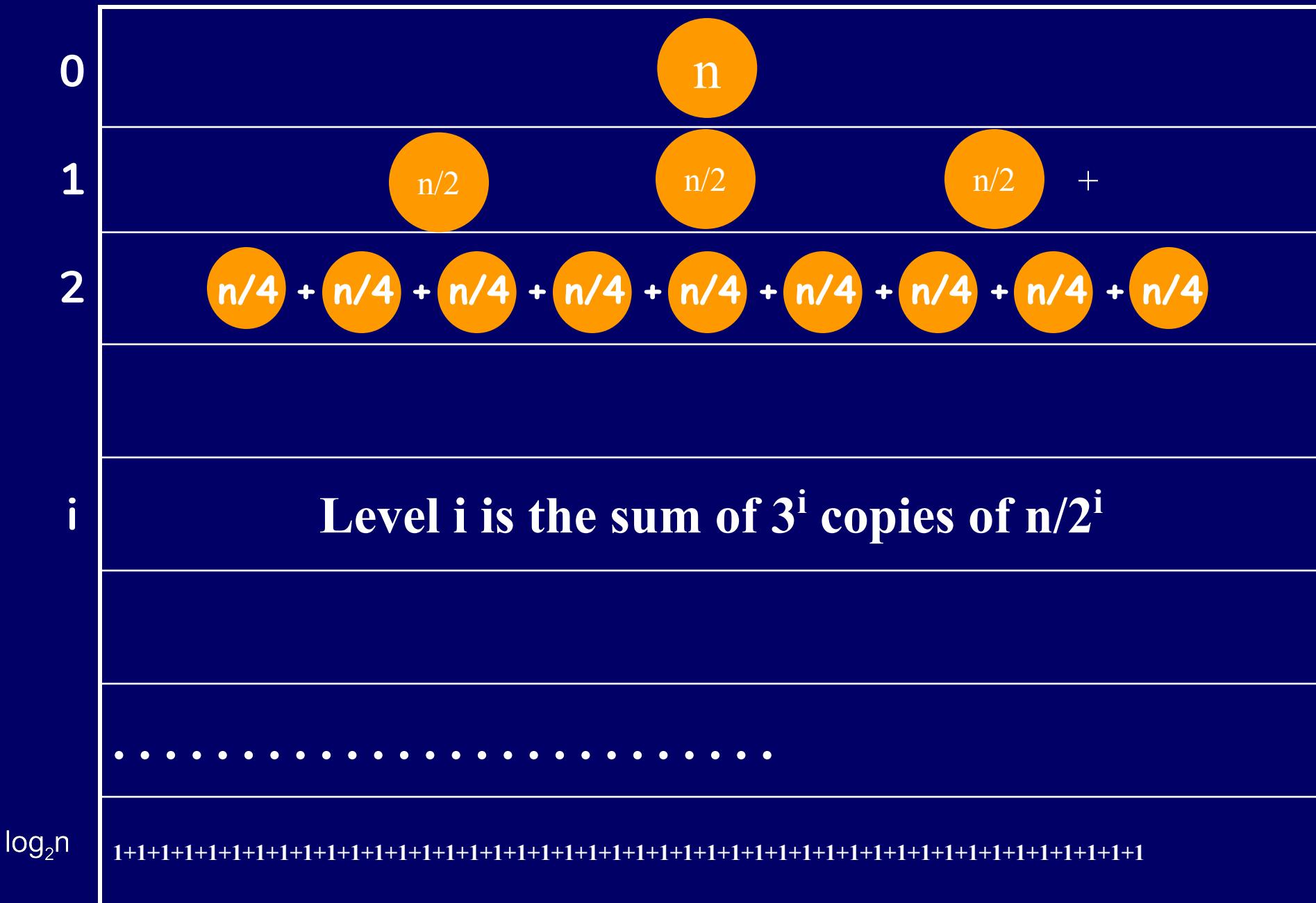
- $T(n) = 3T(n/2) + n$
- *Actualmente:* $T(n) = 2T(n/2) + T(n/2 + 1) + kn$

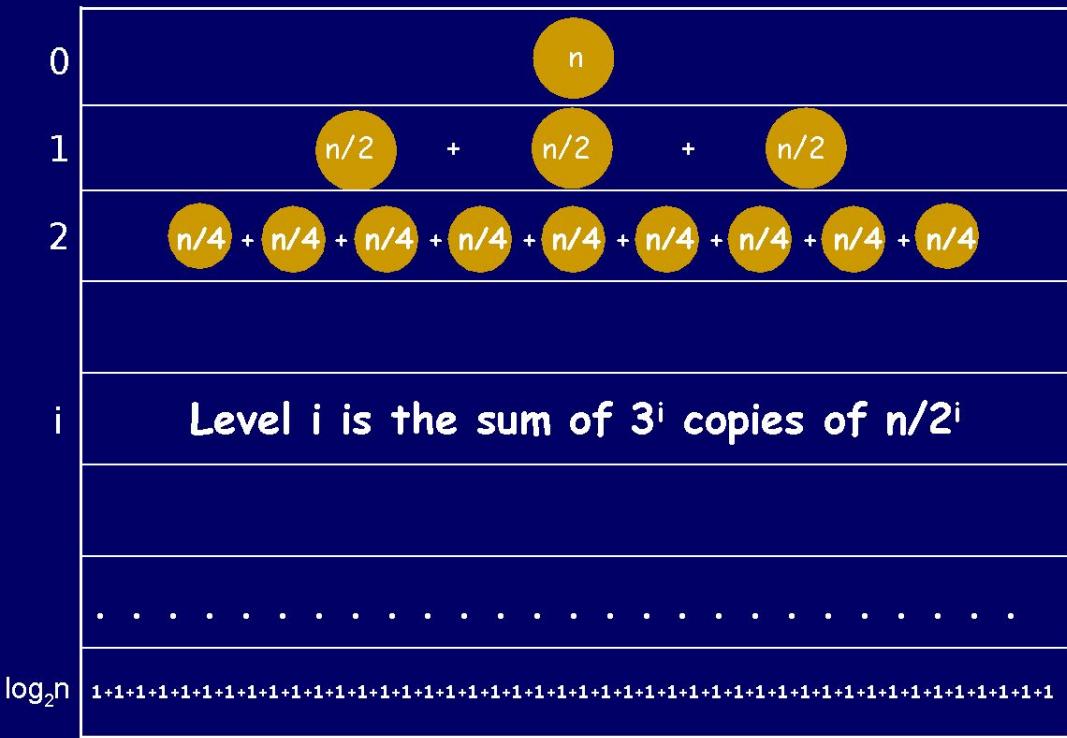
$T(n)$ $=$ n $n/2$ $n/2$ $n/2$ $n/2$ $T(n/4)T(n/4)T(n/4)T(n/4)$ $T(n/4)T(n/4)T(n/4)T(n/4) \quad T(n/4)T(n/4)T(n/4)T(n/4) \quad T(n/4)T(n/4)T(n/4)T(n/4)T(n/4)T(n/4)T(n/4)T(n/4)$











=1·n

$$= 3 \cdot n / 2$$

$$= 9 \cdot n / 4$$

$$= 3^i \cdot n / 2^i$$

$$= 3^{\log n} \cdot n / 2^{\log}$$

$$= n^{\log 3 \cdot 1}$$

Total: $\theta(n^{\log 3}) = \theta(n^{1.58\dots})$

Una considerable mejora para un
(n) grande

No sólo un 25% de
ahorro!

$\theta(n^2)$ vs $\theta(n^{1.58..})$

Multiplication Algorithms

Grade School	n^2
Karatsuba	$n^{1.58\dots}$
Más rápido hasta ahora conocido	$n \log \log n$

¡Eres genial! ¿Estás libre este fin de semana?



No me interesa, Bonzo.
Ya reservé ese dia para mis
amigos del curso de Análisis y
Diseño de Algoritmos.

La mejor forma de aprender es resolviendo los ejercicios en Grupos

Lo único individual son los exámenes ☺