

Dividir y Conquistar

Técnicas de Diseño de Algoritmos (Ex Algoritmos y Estructuras de Datos III)

Basada en trabajos previos de Esteban Feuerstein, Fernando Schapachnik y Flavia Bonomo

25 de agosto de 2024

(2) Nombres

- Esta técnica es conocida como



(2) Nombres

- Esta técnica es conocida como
 - Divide & Conquer



(2) Nombres

- Esta técnica es conocida como
 - Divide & Conquer
 - Dividir y Conquistar



(2) Nombres

- Esta técnica es conocida como
 - Divide & Conquer
 - Dividir y Conquistar
 - Divide y Reinarás



(2) Nombres

- Esta técnica es conocida como
 - Divide & Conquer
 - Dividir y Conquistar
 - Divide y Reinarás
 - D&C



(2) Nombres

- Esta técnica es conocida como
 - Divide & Conquer
 - Dividir y Conquistar
 - Divide y Reinarás
 - D&C
 - etc.



(3) D&C

- Se basa en:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa: no
 - Buscar al máximo en una matriz recursivamente:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa: no
 - Buscar al máximo en una matriz recursivamente:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa: no
 - Buscar al máximo en una matriz recursivamente: sí
- Algunas características de algoritmos D&C:

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa: no
 - Buscar al máximo en una matriz recursivamente: sí
- Algunas características de algoritmos D&C:
 - Las subpartes tienen que ser más pequeñas.

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa: no
 - Buscar al máximo en una matriz recursivamente: sí
- Algunas características de algoritmos D&C:
 - Las subpartes tienen que ser más pequeñas.
 - Y ser el mismo tipo de tarea.

(3) D&C

- Se basa en:
 - Dividir un problema en subproblemas del mismo tipo que el original.
 - Resolver los problemas más pequeños.
 - Combinar las soluciones.
- ¿Cuáles de estos son D&C?
 - Pintar una pared: sí
 - Construir una casa: no
 - Buscar al máximo en una matriz recursivamente: sí
- Algunas características de algoritmos D&C:
 - Las subpartes tienen que ser más pequeñas.
 - Y ser el mismo tipo de tarea.
 - Dividir y combinar pueden no ser nulas, pero no tienen que ser demasiado costosas.

(4) Forma general de D&C

- $F(X)$

(4) Forma general de D&C

- $F(X)$
 - Si X es suficientemente chico o simple, solucionar de manera ad hoc.

(4) Forma general de D&C

- $F(X)$
 - Si X es suficientemente chico o simple, solucionar de manera ad hoc.
 - Si no,

(4) Forma general de D&C

- $F(X)$
 - Si X es suficientemente chico o simple, solucionar de manera ad hoc.
 - Si no,
 - Dividir a X en X_1, X_2, \dots, X_k

(4) Forma general de D&C

- $F(X)$
 - Si X es suficientemente chico o simple, solucionar de manera ad hoc.
 - Si no,
 - Dividir a X en X_1, X_2, \dots, X_k
 - $\forall i \leq k$, hacer $Y_i = F(X_i)$

(4) Forma general de D&C

- $F(X)$
 - Si X es suficientemente chico o simple, solucionar de manera ad hoc.
 - Si no,
 - Dividir a X en X_1, X_2, \dots, X_k
 - $\forall i \leq k$, hacer $Y_i = F(X_i)$
 - Combinar los Y_i en un Y que es una solución para X .

(4) Forma general de D&C

- $F(X)$
 - Si X es suficientemente chico o simple, solucionar de manera ad hoc.
 - Si no,
 - Dividir a X en X_1, X_2, \dots, X_k
 - $\forall i \leq k$, hacer $Y_i = F(X_i)$
 - Combinar los Y_i en un Y que es una solución para X .
 - Devolver Y

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo n/c , siempre que $n/c > n_0$.
 - El costo de efectivamente hacer la subdivisión y luego unir los resultados.

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo n/c , siempre que $n/c > n_0$.
 - El costo de efectivamente hacer la subdivisión y luego unir los resultados.
 - Además hay que resolver los subproblemas: $aT(n/c)$.

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo n/c , siempre que $n/c > n_0$.
 - El costo de efectivamente hacer la subdivisión y luego unir los resultados.
 - Además hay que resolver los subproblemas: $aT(n/c)$.
 - Vamos a utilizar otra función $g(n)$ tal que $g(n) \geq T(n)$ para cualquier valor de n . Podemos suponer que $n_0 = 1$ sino definir $T'(\frac{n}{n_0}) = T(n)$ y usar T' en lugar de T .

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo n/c , siempre que $n/c > n_0$.
 - El costo de efectivamente hacer la subdivisión y luego unir los resultados.
 - Además hay que resolver los subproblemas: $aT(n/c)$.
 - Vamos a utilizar otra función $g(n)$ tal que $g(n) \geq T(n)$ para cualquier valor de n . Podemos suponer que $n_0 = 1$ sino definir $T'(\frac{n}{n_0}) = T(n)$ y usar T' en lugar de T .
 - Sea $b'n^d$, para algún d , una cota superior del costo de dividir en subproblemas y combinar los resultados para un problema de tamaño n . Definimos $g(1) = b = \max\{b', T(1)\}$.

(5) ¿Y cuánto tarda?

- ¿Cómo calculamos la complejidad de un algoritmo D&C?
- El costo de un algoritmo D&C de tamaño n se puede expresar como $T(n)$, que debe considerar:
 - Dividir el problema en a subproblemas de tamaño máximo n/c , siempre que $n/c > n_0$.
 - El costo de efectivamente hacer la subdivisión y luego unir los resultados.
 - Además hay que resolver los subproblemas: $aT(n/c)$.
 - Vamos a utilizar otra función $g(n)$ tal que $g(n) \geq T(n)$ para cualquier valor de n . Podemos suponer que $n_0 = 1$ sino definir $T'(\frac{n}{n_0}) = T(n)$ y usar T' en lugar de T .
 - Sea $b'n^d$, para algún d , una cota superior del costo de dividir en subproblemas y combinar los resultados para un problema de tamaño n . Definimos $g(1) = b = \max\{b', T(1)\}$.
- Es decir, $T(n) = aT(n/c) + b'n^d \leq g(n) = ag(n/c) + bn^d$.

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$
- $= a(ag(c^{k-2}) + (b c^{(k-1)d})) + bc^{kd}$

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$
- $= a(ag(c^{k-2}) + (b c^{(k-1)d})) + bc^{kd}$
- $= a^2g(c^{k-2}) + abc^{(k-1)d} + bc^{kd}$

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$
- $= a(ag(c^{k-2}) + (b c^{(k-1)d})) + bc^{kd}$
- $= a^2g(c^{k-2}) + abc^{(k-1)d} + bc^{kd}$
- $= a^3g(c^{k-3}) + a^2b(c^{k-2})^d + abc^{(k-1)d} + bc^{kd}$

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$
- $= a(ag(c^{k-2}) + (b c^{(k-1)d})) + bc^{kd}$
- $= a^2g(c^{k-2}) + abc^{(k-1)d} + bc^{kd}$
- $= a^3g(c^{k-3}) + a^2b(c^{k-2})^d + abc^{(k-1)d} + bc^{kd}$
- ...

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$
- $= a(ag(c^{k-2}) + (b c^{(k-1)d})) + bc^{kd}$
- $= a^2g(c^{k-2}) + abc^{(k-1)d} + bc^{kd}$
- $= a^3g(c^{k-3}) + a^2b(c^{k-2})^d + abc^{(k-1)d} + bc^{kd}$
- ...
- $= a^jg(c^{k-j}) + \sum_{i=0}^{j-1} a^i bc^{(k-i)d}$

(6) ¿Y cuánto tarda? (cont.)

- Supongamos que $n = c^k$ para algún k y analicemos la recurrencia.
- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= g(c^k) = ag(c^{k-1}) + b(c^k)^d$
- $= a(ag(c^{k-2}) + (b c^{(k-1)d})) + bc^{kd}$
- $= a^2g(c^{k-2}) + abc^{(k-1)d} + bc^{kd}$
- $= a^3g(c^{k-3}) + a^2b(c^{k-2})^d + abc^{(k-1)d} + bc^{kd}$
- ...
- $= a^jg(c^{k-j}) + \sum_{i=0}^{j-1} a^i bc^{(k-i)d}$
- $= a^jg(c^{k-j}) + b\sum_{i=0}^{j-1} a^i c^{(k-i)d}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- Si tenemos en cuenta que $g(1) = b$, nos queda

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- Si tenemos en cuenta que $g(1) = b$, nos queda
- $= a^k b + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- Si tenemos en cuenta que $g(1) = b$, nos queda
- $= a^k b + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{(k-i)d}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- Si tenemos en cuenta que $g(1) = b$, nos queda
- $= a^k b + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{dk-di}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- Si tenemos en cuenta que $g(1) = b$, nos queda
- $= a^k b + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{dk-di}$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di}$

(7) ¿Y cuánto tarda? (cont.)

- $g(c^k) = a^j g(c^{k-j}) + b \sum_{i=0}^{j-1} a^i c^{(k-i)d}$
- Seguimos teniendo una recurrencia. ¿Cuál es el caso base?
 $g(1)$
- $c^{k-j} = 1$, es decir $c^k / c^j = 1$, es decir, hasta que
 $j = k = \log_c n$
- $= a^k g(1) + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- Si tenemos en cuenta que $g(1) = b$, nos queda
- $= a^k b + b \sum_{i=0}^{k-1} a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{(k-i)d}$
- $= b \sum_{i=0}^k a^i c^{dk-di}$
- $= b c^{dk} \sum_{i=0}^k a^i c^{-di}$
- ¿Y esto cuánto es?

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir,

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir,

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir,

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir,

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:
 - Cuando $n \rightarrow \infty$:

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:
 - Cuando $n \rightarrow \infty$:
 - $bn \sum_{i=0}^{\log_c n} (a/c)^i \rightarrow bn \text{ cte} = O(n)$

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:
 - Cuando $n \rightarrow \infty$:
 - $bn \sum_{i=0}^{\log_c n} (a/c)^i \rightarrow bn \text{ cte} = O(n)$
 - Si $a = c$:

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:
 - Cuando $n \rightarrow \infty$:
 - $bn \sum_{i=0}^{\log_c n} (a/c)^i \rightarrow bn \text{ cte} = O(n)$
 - Si $a = c$:
 - $bn \sum^{\log_c n} 1 = O(n \log_c n)$

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:
 - Cuando $n \rightarrow \infty$:
 - $bn \sum_{i=0}^{\log_c n} (a/c)^i \rightarrow bn \text{ cte} = O(n)$
 - Si $a = c$:
 - $bn \sum^{\log_c n} 1 = O(n \log_c n)$
 - Si $a > c$ ("muchos subproblemas")

(8) Análisis por caso

- $T(n) \leq g(n) = ag(n/c) + bn^d$
- $= bc^{dk} \sum_{i=0}^k a^i c^{-di} = bn^d \sum_{i=0}^k a^i c^{-di}$
- Si $a = 1$ y $d = 0$, es decir, 1 subproblema, combinar tiene costo constante: $b \sum_{i=0}^{\log_c n} 1 = O(\log_c n)$
- Si $d = 1$, es decir, división + unión tiene costo lineal.
- $g(n)$ queda como $bn \sum_{i=0}^{\log_c n} (a/c)^i$
 - Si $a < c$ ("pocos subproblemas"), $a/c < 1$, por ende, la serie converge:
 - Cuando $n \rightarrow \infty$:
 - $bn \sum_{i=0}^{\log_c n} (a/c)^i \rightarrow bn \text{ cte} = O(n)$
 - Si $a = c$:
 - $bn \sum_{i=0}^{\log_c n} 1 = O(n \log_c n)$
 - Si $a > c$ ("muchos subproblemas")
 - Recordemos que $\sum_{i=0}^x y^i = \frac{y^{x+1}-1}{y-1} \quad (1)$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$
- Aplicando $O()$ queda como $O(n(\frac{a}{c})^{\log_c n})$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$
- Aplicando $O()$ queda como $O(n(\frac{a}{c})^{\log_c n})$
- $= O(n \frac{a^{\log_c n}}{c^{\log_c n}})$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$
- Aplicando $O()$ queda como $O(n(\frac{a}{c})^{\log_c n})$
- $= O(n \frac{a^{\log_c n}}{c^{\log_c n}})$
- $= O(n \frac{a^{\log_c n}}{n})$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$
- Aplicando $O()$ queda como $O(n(\frac{a}{c})^{\log_c n})$
- $= O(n \frac{a^{\log_c n}}{c^{\log_c n}})$
- $= O(n \frac{a^{\log_c n}}{n})$
- $= O(a^{\log_a n \cdot \log_c a})$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$
- Aplicando $O()$ queda como $O(n(\frac{a}{c})^{\log_c n})$
- $= O(n \frac{a^{\log_c n}}{c^{\log_c n}})$
- $= O(n \frac{a^{\log_c n}}{n})$
- $= O(a^{\log_a n \cdot \log_c a})$
- $= O((a^{\log_a n})^{\log_c a})$

(9) Análisis por caso (cont.)

- Caso $d = 1$, $a > c$, $g(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i$
- Usando (1) nos queda
- $T(n) \leq g(n) = bn \frac{(a/c)^{\log_c n+1} - 1}{a/c - 1}$
- Aplicando $O()$ queda como $O(n(\frac{a}{c})^{\log_c n})$
- $= O(n \frac{a^{\log_c n}}{c^{\log_c n}})$
- $= O(n \frac{a^{\log_c n}}{n})$
- $= O(a^{\log_a n \cdot \log_c a})$
- $= O((a^{\log_a n})^{\log_c a})$
- $= O(n^{\log_c a})$

(10) Teorema Maestro

- Permite resolver relaciones de recurrencia de la forma:

$$T(n) = \begin{cases} a T(n/c) + f(n) & \text{si } n > 1 \\ 1 & \text{si } n = 1 \end{cases}$$

(10) Teorema Maestro

- Permite resolver relaciones de recurrencia de la forma:

$$T(n) = \begin{cases} a T(n/c) + f(n) & \text{si } n > 1 \\ 1 & \text{si } n = 1 \end{cases}$$

- Si $f(n) = O(n^{\log_c a - \epsilon})$ para $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_c a})$

(10) Teorema Maestro

- Permite resolver relaciones de recurrencia de la forma:

$$T(n) = \begin{cases} a T(n/c) + f(n) & \text{si } n > 1 \\ 1 & \text{si } n = 1 \end{cases}$$

- Si $f(n) = O(n^{\log_c a - \epsilon})$ para $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_c a})$
- Si $f(n) = \Theta(n^{\log_c a})$, entonces $T(n) = \Theta(n^{\log_c a} \log n)$

(10) Teorema Maestro

- Permite resolver relaciones de recurrencia de la forma:

$$T(n) = \begin{cases} a T(n/c) + f(n) & \text{si } n > 1 \\ 1 & \text{si } n = 1 \end{cases}$$

- Si $f(n) = O(n^{\log_c a - \epsilon})$ para $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_c a})$
- Si $f(n) = \Theta(n^{\log_c a})$, entonces $T(n) = \Theta(n^{\log_c a} \log n)$
- Si $f(n) = \Theta(n^{\log_c a} \log^k n)$ para algún $k \geq 0$, entonces $T(n) = \Theta(n^{\log_c a} \log^{k+1} n)$ (generalización del caso anterior)

(10) Teorema Maestro

- Permite resolver relaciones de recurrencia de la forma:

$$T(n) = \begin{cases} a T(n/c) + f(n) & \text{si } n > 1 \\ 1 & \text{si } n = 1 \end{cases}$$

- Si $f(n) = O(n^{\log_c a - \epsilon})$ para $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_c a})$
- Si $f(n) = \Theta(n^{\log_c a})$, entonces $T(n) = \Theta(n^{\log_c a} \log n)$
- Si $f(n) = \Theta(n^{\log_c a} \log^k n)$ para algún $k \geq 0$, entonces $T(n) = \Theta(n^{\log_c a} \log^{k+1} n)$ (generalización del caso anterior)
- Si $f(n) = \Omega(n^{\log_c a + \epsilon})$ para $\epsilon > 0$ y $af(n/c) < kf(n)$ para $k < 1$ y n suficientemente grandes, entonces $T(n) = \Theta(f(n))$

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).
- $x = x_1 b^{n/2} + x_0$ e $y = y_1 b^{n/2} + y_0$.

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).
- $x = x_1 b^{n/2} + x_0$ e $y = y_1 b^{n/2} + y_0$.
- Entonces xy es $x_1 y_1 b^n + (x_0 y_1 + x_1 y_0) b^{n/2} + x_0 y_0$

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).
- $x = x_1b^{n/2} + x_0$ e $y = y_1b^{n/2} + y_0$.
- Entonces xy es $x_1y_1b^n + (x_0y_1 + x_1y_0)b^{n/2} + x_0y_0$
- Todavía no gané nada, pero qué pasa si defino:

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).
- $x = x_1 b^{n/2} + x_0$ e $y = y_1 b^{n/2} + y_0$.
- Entonces xy es $x_1 y_1 b^n + (x_0 y_1 + x_1 y_0) b^{n/2} + x_0 y_0$
- Todavía no gané nada, pero qué pasa si defino:
- $m_1 = x_0 y_0$, $m_2 = x_1 y_1$ y $m_3 = (x_0 - x_1)(y_1 - y_0)$

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).
- $x = x_1 b^{n/2} + x_0$ e $y = y_1 b^{n/2} + y_0$.
- Entonces xy es $x_1 y_1 b^n + (x_0 y_1 + x_1 y_0) b^{n/2} + x_0 y_0$
- Todavía no gané nada, pero qué pasa si defino:
- $m_1 = x_0 y_0$, $m_2 = x_1 y_1$ y $m_3 = (x_0 - x_1)(y_1 - y_0)$
- La multiplicación se vuelve
$$xy = m_2 b^n + (m_1 + m_2 + m_3) b^{n/2} + m_1$$

(11) Multiplicación entera

- ¿Cuál es la complejidad de multiplicar dos números enteros?
- Si tienen n dígitos en base b la complejidad es $O(n^2)$.
- ¿Puedo hacer algo mejor?
- Podemos expresarlos como una suma donde cada sumando tiene la mitad de los dígitos (aprox).
- $x = x_1 b^{n/2} + x_0$ e $y = y_1 b^{n/2} + y_0$.
- Entonces xy es $x_1 y_1 b^n + (x_0 y_1 + x_1 y_0) b^{n/2} + x_0 y_0$
- Todavía no gané nada, pero qué pasa si defino:
- $m_1 = x_0 y_0$, $m_2 = x_1 y_1$ y $m_3 = (x_0 - x_1)(y_1 - y_0)$
- La multiplicación se vuelve
$$xy = m_2 b^n + (m_1 + m_2 + m_3) b^{n/2} + m_1$$
- Esto se llama algoritmo de Karatsuba.

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x , y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x , y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.
- ¿Cuál es la complejidad?

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x , y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.
- ¿Cuál es la complejidad?
- Separaciones, sumas y desplazados son lineales en n .

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.
- ¿Cuál es la complejidad?
- Separaciones, sumas y desplazados son lineales en n .
- Hay 3 llamadas recursivas.

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.
- ¿Cuál es la complejidad?
- Separaciones, sumas y desplazados son lineales en n .
- Hay 3 llamadas recursivas.
- $T(n) = 3T(\lceil n/2 \rceil) + cn + c'$

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.
- ¿Cuál es la complejidad?
- Separaciones, sumas y desplazados son lineales en n .
- Hay 3 llamadas recursivas.
- $T(n) = 3T(\lceil n/2 \rceil) + cn + c'$
- $T(n) = 3T(\lceil n/2 \rceil) + f(n)$ con $f(n) = O(n^{\log_2 3 - \epsilon})$

(12) Algoritmo de Karatsuba

- $xy = m_2b^n + (m_1 + m_2 + m_3)b^{n/2} + m_1$
- Pensémoslo algorítmicamente.
- Karatsuba(x, y)
 - 1) Si son suficientemente chicos, multiplicarlos “a mano” y retornar.
 - 2) Separar a x en x_1 y x_0 .
 - 3) Separar a y en y_1 y y_0 .
 - 4) Calcular m_1 , m_2 y m_3 mediante llamadas recursivas.
 - 5) Sumar m_2 desplazado n b -bits + $(m_1 + m_2 + m_3)$ desplazado $n/2$ b -bits + m_1 .
 - 6) Retornar esa suma.
- ¿Cuál es la complejidad?
- Separaciones, sumas y desplazados son lineales en n .
- Hay 3 llamadas recursivas.
- $T(n) = 3T(\lceil n/2 \rceil) + cn + c'$
- $T(n) = 3T(\lceil n/2 \rceil) + f(n)$ con $f(n) = O(n^{\log_2 3 - \epsilon})$
- Que por el Teorema Maestro tiene $\Theta(n^{\log_2 3})$, es decir aprox. $\Theta(n^{1.59})$.