Multiplicación recursiva

- Para toda x real definimos $\lfloor x \rfloor$ como el mayor entero que no excede x (la parte entera de x).

Función multiplica(y, z)

1. Si $z = 0$ entonces regresa 0.
2. Si z es impar
   1. entonces regresa multiplica$(2y, \lfloor z/2 \rfloor) + y$,
   2. si no regresa multiplica$(2y, \lfloor z/2 \rfloor)$.

- Demostremos por inducción en z que para toda $y, z \geq 0$ la llamada multiplica(y, z) regresa yz.

Multiplicación iterativa

Función multiplica(y, z)

1. $x \leftarrow 0$.
2. Mientras $z > 0$ haz:
   1. Si z es impar entonces $x \leftarrow x + y$.
   2. $y \leftarrow 2y$.
   3. $z \leftarrow \lfloor z/2 \rfloor$.
3. Regresa x.

- Demostraremos que multiplica(y, z) regresa $x = yz$.

Tiempo de ejecución de la multiplicación

Función multiplica(y, z)

1. $x \leftarrow 0$.
2. Mientras $z > 0$ haz:
   1. Si z es impar entonces $x \leftarrow x + y$.
   2. $y \leftarrow 2y$.
   3. $z \leftarrow \lfloor z/2 \rfloor$.
3. Regresa x.

- Supongamos que y, z tienen n bits.
- La llamada, la asignación y el regreso cuestan $O(1)$ cada una.
- La decisión y las asignaciones dentro del ciclo cuestan $O(1)$ cada una.
- El ciclo se ejecuta un máximo de n veces.
- Por lo tanto todo el proceso toma tiempo $O(n)$.

Recurrencia para multiplicación de enteros

Función multiplica(y, z)

1. Si $z = 0$ entonces regresa 0.
2. Si z es impar
   1. entonces regresa multiplica$(2y, \lfloor z/2 \rfloor) + y$,
   2. si no regresa multiplica$(2y, \lfloor z/2 \rfloor)$.

- Sea $T(n)$ el tiempo de ejecución de multiplica(y, z) cuando z es entero de n bits. Entonces

$$T(n) = \begin{cases} c & \text{si } n = 1 \\ T(n-1) + d & \text{en otro caso} \end{cases}$$

para algunas constantes c y d.

Sustitución repetida

- Suponga que $n > 1$. Entonces:

$$\begin{aligned} T(n) &= T(n-1) + d \\ &= (T(n-2) + d) + d \\ &= T(n-2) + 2d \\ &= (T(n-3) + d) + 2d \\ &= T(n-3) + 3d. \end{aligned}$$

- Hay un patrón: parece que después de i sustituciones

$$T(n) = T(n-i) + id.$$

- Si escogemos $i = n - 1$ entonces

$$T(n) = T(n-(n-1)) + (n-1)d = T(1) + (n-1)d = c + (n-1)d.$$

- Pero esto no es una demostración. ¿Por qué?

Demostración por inducción

- Demostraremos que $T(n) = c + (n-1)d$ por inducción en n.
- Para $n = 1$ tenemos que $T(1) = c + (1-1)d = c$, lo cual es cierto.
- Ahora supongamos que $n \geq 1$ y que $T(n) = c + (n-1)d$. Entonces

$$\begin{aligned} T(n+1) &= T(n) + d \\ &= c + (n-1)d + d \\ &= c + nd. \end{aligned}$$

- Que es lo que queríamos demostrar.

Primera idea

- Suponga que y y z son dos enteros de n bits.
- Suponga también que n es una potencia de 2.
- Divida y y z en dos mitades (cada una con $n/2$ bits)

$$\begin{aligned} y &= a2^{n/2} + b \\ z &= c2^{n/2} + d. \end{aligned}$$

- Por lo tanto

$$\begin{aligned} yz &= (a2^{n/2} + b)(c2^{n/2} + d) \\ &= ac2^n + (ad + bc)2^{n/2} + bd. \end{aligned}$$

Segunda idea

- Sin embargo, yz se puede calcular de esta otra forma:
  1. Sea $u = (a + b)(c + d)$.
  2. Sea $v = ac$.
  3. Sea $w = bd$.
  4. Sea $x = v2^n + (u - v - w)2^{n/2} + w$.
- Verifiquemos que esto funciona:

$$\begin{aligned} x &= v2^n + (u - v - w)2^{n/2} + w \\ &= ac2^n + ((a + b)(c + d) - ac - bd)2^{n/2} + bd \\ &= ac2^n + (ad + bc)2^{n/2} + bd \\ &= yz. \end{aligned}$$

Segundo análisis

- Esta nueva idea nos sirve para calcular yz con 3 multiplicaciones de números de $n/2$ bits y algunas sumas y corrimientos.
- El tiempo de ejecución queda dado por

$$T(n) = \begin{cases} \gamma & \text{si } n = 1 \\ 3T(n/2) + \delta n & \text{si } n > 1 \end{cases}$$

- El teorema general nos dice que $T(n) \in O(n^{\log_2 3}) \approx O(n^{1.585})$.
- Esto sí mejora el algoritmo anterior.