MULTIPLICACIÓN DE ENTEROS LARGOS

ESTRATEGIA "DIVIDE Y VENCERÁS"

- La estrategia de divide y vencerás resuelve un problema de la siguiente manera:

1. Dividirlo en subproblemas que son en sí mismos instancias más pequeñas del mismo tipo de problema.
2. Resolver estos subproblemas de forma recursiva.
3. Combinando adecuadamente sus respuestas.

- El trabajo real se hace poco a poco, en tres lugares diferentes:
  - En la partición de problemas en subproblemas.
  - Al final de la recursividad, cuando los subproblemas son tan pequeños que se resuelven por completo.
  - Y en la combinación de las respuestas parciales.
- Estos elementos se mantienen juntos y coordinados por el núcleo del algoritmo con estructura recursiva.

MULTIPLICACIÓN DE ENTEROS LARGOS ALGORITMO CLÁSICO

1234 * 5678 =

= 1234 * (5*1000 + 6*100 + 7*10 + 8)

= 1234*5*1000 + 1234*6*100 + 1234*7*10 + 1234*8

= 6170000 + 740400 + 86380 + 9872

= 7006652

MULTIPLICACIÓN DE ENTEROS GRANDES ALGORITMO DIVIDE Y VENCERÁS SIMPLE

- 1234 = 12*100 + 34
- 5678 = 56*100 + 78
- 1234*5678 = (12*100 + 34) * (56*100 + 78)
  = 12*56*10000 + (12*78 + 34*56)*100 +34*78

Se reduce de una multiplicación de 4 cifras a

4 multiplicaciones de 2 cifras, más tres sumas y varios desplazamientos.

MULTIPLICACIÓN DE ENTEROS GRANDES ALGORITMO DIVIDE Y VENCERÁS SIMPLE

n = 4

xi = 12
xd = 34
yi = 56
yd = 78

- 1234 = 12*100 + 34
- 5678 = 56*100 + 78
- 1234*5678 = (12*100 + 34) * (56*100 + 78)
  = 12*56*10000 + (12*78 + 34*56)*100 +34*78
- 1234*5678 = $(xi * 10^2 + xd) * (yi * 10^2 + yd)$
  = $(xi * yi) * 10^4 + (xi * yd + xd * yi)*10^2 + (xd * yd)$

Multiplicación de enteros de n cifras:

Algoritmo "divide y vencerás" simple

n = 8

1. Dividir

$$X = 12345678 = xi*10^4 + xd \qquad xi=1234 \qquad xd=5678$$
$$Y = 24680135 = yi*10^4 + yd \qquad yi=2468 \qquad yd=0135$$

2. Combinar

$$X*Y = (xi*10^4 + xd) * (yi*10^4 + yd)$$
$$= xi*yi*10^8 + (xi*yd+xd*yi)*10^4 + xd*yd$$

En general:

$$X = xi*10^{n/2} + xd$$

$$Y = yi*10^{n/2} + yd$$

$$X*Y = (xi*10^{n/2} + xd) * (yi*10^{n/2} + yd)$$

$$= xi*yi*10^{n} + (xi*yd+xd*yi)*10^{n/2} + xd*yd$$

FUNCIÓN MULTIPLICA

```
funcion multiplica(X, Y, n)
{
    if(P es pequeño){
        return X * Y;
    }
    else    {
        Obtener xi, xd, yi, yd;

        z1 = multiplica(xi, yi, n/2);
        z2 = multiplica(xi, yd, n/2);
        z3 = multiplica(xd, yi, n/2);
        z4 = multiplica(xd, yd, n/2);

        aux = suma(z2, z3);

        z1  = desplaza_izq(z1, n);
        aux = desplaza_izq(aux, n/2);

        resultado = suma(z1, aux);
        resultado = suma(resultado, z4);
        return resultado;
    }
}
```

DIVIDIR

COMBINAR

Complejidad
$O(n^2)$

EJEMPLO ENTEROS LARGOS

- A = 12345678
- B = 24680135
- A * B = 3.046929997 x 10^{14}

xi = 1234 **xd** = 5678

yi = 2468 **yd** = 0135

z1 = xi * yi = 3045512
z2 = xi * yd = 166590
z3 = xd * yi = 14013304
z4 = xd * yd = 766530

aux = z2 + z3 = 14179894

z1 = 3045512 x 10^8
aux = 14179894 x 10^4

								1	1					
3	0	4	5	5	1	2	0	0	0	0	0	0	0	0
			1	4	1	7	9	8	9	4	0	0	0	0
									7	6	6	5	3	0
3	0	4	6	9	2	9	9	9	7	0	6	5	3	0

$$X*Y = \underbrace{xi*yi*10^n}_{z1} + \underbrace{(xi*yd+xd*yi)*10^{n/2}}_{aux} + \underbrace{xd*yd}_{z4}$$

MULTIPLICACIÓN DE ENTEROS LARGOS

Optimización

MULTIPLICACIÓN DE ENTEROS LARGOS

- El matemático Carl Friedrich Gauss (1777-1855) notó una vez que, aunque el producto de dos números complejos

$$(a + bi)(c + di) = ac - bd + (bc + ad)i$$

- parece implicar cuatro multiplicaciones de números reales, de hecho se puede hacer con solo tres: **ac, bd y (a + b) (c + d)**, ya que:

$$bc + ad = (a + b)(c + d) - ac - bd.$$

- En nuestra forma de pensar big-O, reducir el número de multiplicaciones de cuatro a tres parece un desperdicio de ingenio.

- Pero esta modesta mejora se vuelve muy significativa cuando se aplica de forma recursiva.

- Alejémonos de los números complejos y veamos cómo esto ayuda con la multiplicación regular.
- Suponga que **x**, **y** son dos enteros de **n** bits, y suponga por conveniencia que **n** es una potencia de 2 (el caso más general apenas es diferente).
- Como primer paso hacia la multiplicación de **x** * **y**, divida cada uno de ellos en sus mitades izquierda y derecha, que tienen n/2 bits de largo.

$$x = \boxed{\quad x_L \quad}\ \boxed{\quad x_R \quad} = 2^{n/2} x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\ \boxed{\quad y_R \quad} = 2^{n/2} y_L + y_R.$$

- Por ejemplo, si $x = 10110110_2$ (el subíndice $_2$ significa "binario") entonces:
  - $x_L = 1011_2$
  - $x_R = 0110_2$
  - $x = 1011_2 \times 2^4 + 0110_2$.

- El producto de **x * y** se puede reescribir como:

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R.$$

- Calcularemos **x * y** mediante la expresión de la derecha.
- Las adiciones toman un tiempo lineal, al igual que las multiplicaciones por potencias de 2 (que son simplemente desplazamientos a la izquierda).
- Las operaciones significativas son las cuatro multiplicaciones de n/2 bits

$$x_L y_L, \quad x_L y_R, \quad x_R y_L, \quad x_R y_R$$

estas las podemos manejar por cuatro llamadas recursivas.

- Por lo tanto, nuestro método para multiplicar números de **n** bits comienza haciendo llamadas recursivas para multiplicar estos cuatro pares de números de n/2 bits (cuatro subproblemas de la mitad del tamaño), y luego evalúa la expresión anterior en O(n) tiempo.

- Escribiendo T(n) para el tiempo de ejecución total en entradas de **n** bits, obtenemos la relación de recurrencia:

$$T(n) = 4T(n/2) + O(n).$$

- Mientras tanto, esta recurrencia en particular funciona para $O(n^2)$, el mismo tiempo de ejecución que la técnica tradicional de multiplicación.

- Tenemos un algoritmo radicalmente nuevo, pero todavía no hemos avanzado en eficiencia. ¿Cómo se puede acelerar nuestro método?

- Aquí es donde viene a la mente el truco de Gauss.

$$bc + ad = (a+b)(c+d) - ac - bd.$$

- Aunque la expresión para **x * y** parece exigir cuatro multiplicaciones de n/2 bits, como antes, solo tres bastarán:

$$x_L y_L,\ x_R y_R,\ \text{and}\ (x_L + x_R)(y_L + y_R)$$

- Ya que: **b c + a d = (b + a) (d + c) - b d - a c**

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

- El algoritmo resultante tiene un tiempo de ejecución mejorado de:

$$T(n) = 3T(n/2) + O(n).$$

- El punto es que ahora la mejora constante del factor, de 4 a 3, ocurre en cada nivel de la recursividad, y este efecto compuesto conduce a tener un límite de tiempo dramáticamente menor de $O(n^{1.59})$.

1

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

2

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

function multiply(x, y)
Input: n-bit positive integers x and y
Output: Their product

if $n = 1$: return xy

x_L, x_R = leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of x
y_L, y_R = leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of y

$P_1 = $ multiply(x_L, y_L)
$P_2 = $ multiply(x_R, y_R)
$P_3 = $ multiply($x_L + x_R, y_L + y_R$)
return $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$