Introducción:

¿Qué es un algoritmo? Es un conjunto de reglas muy específicas que describen cómo realizar una tarea. Un algoritmo es más que una "tarea a realizar" es una receta de cómo hacerla (y puede haber muchas recetas distintas para el mismo objetivo)

Ejemplo: (Multiplicación de enteros)

Dados dos enteros x, y queremos calcular su producto $x \cdot y$

INPUT: enteros x, y de longitud n ——— algoritmo ——→ OUTPUT: $x \cdot y$

Algo 1: (del colegio)

```
   5678   x
   1234   =
-----------
   2271 (2) ← n
  17034
 11356
 5678
-----------
 7006.652
```

PREGUNTAS:

(1) Siempre lleva a respuesta correcta? [CORRECTNESS]

(2) Cuántas operaciones requiere? multiplicar dos enteros de longitud n

$\leq$ [$2n + 2n + \dots + 2n$] + $\leq 2n^2$

- $2n + 2n + \dots + 2n$: n-veces; cada renglón intermedio requirió $\leq n$ mult y (n sumas (una por dígito)); filas intermedias multiplicando
- $2n^2$: sumas; llevar

$\leq 4n^2$ ← Trabajo crece cuadráticamente

si duplicamos la long. ⇝ 4x trabajo

3x long ⇝ 9x "

⋮ ⋮

(3) Es esto lo mejor posible? O habrá maneras "preferibles"

↳ típicamente queremos soluciones más eficientes pero no siempre, a veces tenemos otros criterios (paralelizabilidad, robustez, etc...).

Otro método: (recursión)

Suponga que $n = 2^m$ (que hasta x y y a potencia de 2 es una potencia de 2)

```
def product(x, y):
    n = length(x)
    If length(n) == 1:
        return x*y
    else:
        a,b = Primera y segunda mitades de x
        c,d = Primera " " " y
```

(5678 : a = 56, b = 78)

Calculamos recursivamente

```
ac := product(a, c)
ad := product(a, d)
bc := product(b, c)
bd := product(b, d)
```

Con algo de sumas del colegio calculamos

$$10^n \cdot (ac) + 10^{\frac{n}{2}} [ad + bc] + bd$$

(1) Es un método correcto? Sí porque

$$x = 10^{\frac{n}{2}} a + b$$
$$y = 10^{\frac{n}{2}} c + d$$

← así es como funciona el sistema decimal...

$$x \cdot y = 10^n ac + 10^{\frac{n}{2}} ad + 10^{\frac{n}{2}} bc + bd$$
$$= 10^n ac + 10^{\frac{n}{2}} [ad + bc] + bd$$

← Resultado es efectivamente $x \cdot y$

(2) Cuántas operaciones? ... Ya veremos después

(3) Es óptimo? Obs: Para el cálculo solo nos interesa $ad + bc$ pero no ad y bc individualmente

IDEA: [Gauss]

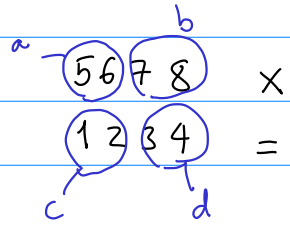
$$(a+b)\cdot(c+d) - a\cdot c - b\cdot d = ad + bc$$

↑ ↑ ↑

sólo 3 multiplicaciones y no 4 ($a\cdot c$, $b\cdot d$, $a\cdot d$ y $b\cdot c$)

Esta obs lleva a otro método...

Otro método [KARATSUBA]



(1) $a \cdot c = 56 \times 78 = 672$

(2) $b \cdot d = 78 \times 34 = 2652$

(3) $(a+b)\cdot(c+d) = 134 \times 46 = 6164$

(4) $(3) - (1) - (2) = 6164 - 672 - 2652 = 2840$

(5) $10^4 (1) + 10^2 (4) + (2) = 6720000 + 284000 + 2652 = 7006652$

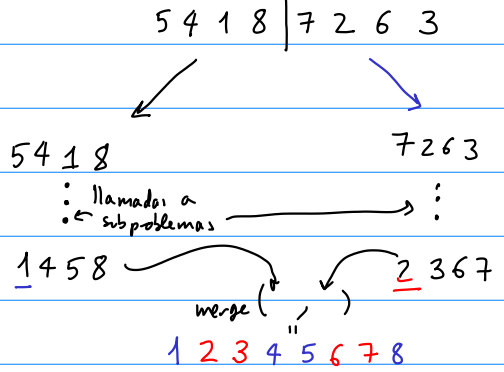
Demostremos que el número de operaciones del algoritmo de Karatsuba es $\leq C\, n^{1.59}$, mucho mejor que $4n^2$ para n grande.

Ejemplo 2: [Merge Sort]

INPUT: Lista de n enteros distintos 5 4 1 8 7 2 6 3

OUTPUT: Lista de los mismos n enteros, escritos en orden creciente. 1 2 3 4 5 6 7 8

Algoritmo [Merge sort, conocido en 1945 John Von Neumann]



La parte más interesante es merge (C, D)

INPUT: Listas ordenadas C, D de longitud $\frac{n}{2}$ cada uno

OUTPUT: Lista ordenada con entradas $C \cup D$.

merge(C, D)

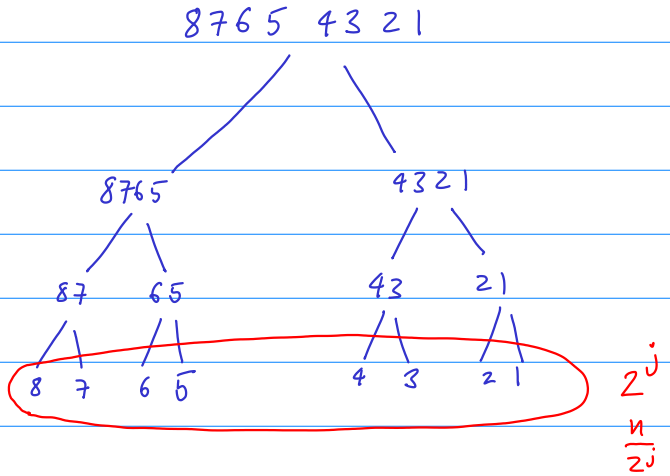
```
i = 0 ✓
j = 0 ✓ 3
res = [] ✓
for k in range(n):
    if C[i] < D[j]: ✓
        res.append(C[i])
        i = i+1 ✓
    if D[j] < C[i]:
        res.append(D[j])
        j = j+1
```

P) Cuántas operaciones realiza merge? Operaciones quiere decir asignaciones, incrementos, comparaciones, etc.

$\leq 4n + 3$, si $n = \frac{l}{2}$ $\leq 2l + 3 \leq 6l$

↑ simplificar $l \geq 1$

Cuántas operaciones en total?



2^j problemas con strings de tamaño $\frac{n}{2^j}$

Trabajo realizado exclusivamente en el nivel j

$\leq 2^j \cdot 6 \frac{n}{2^j} = 6n$

Trabajo realizado en todos los niveles

$6n \cdot \#\text{ de niveles} = 6n \cdot \log_2(n)$

$\leq C\, n \log(n)$

mucho, mucho mejor que n^2

Teorema: El número de operaciones de MergeSort en listas de longitud n es $\leq K n \log(n)$.

el tiempo de ejecución

Para nuestro análisis de algoritmos seguiremos varios principios:

(1) Queremos tener una cota que sea "worst-case" analysis (válida para todos los inputs de una longitud dada)

(2) Queremos ganar una descripción "a vuelo de pájaro" (big picture) (ejemplo $4\ell + 2 \leq 6\ell$) ← más simple
que ignore constantes y características propias de la implementación

(3) Queremos entender comportamiento asintótico
($n \log(n) + 100n \leq n^2$)
↖ para valores grandes de n

"Un algoritmo es rápido si su tiempo de ejecución crece lentamente con el tamaño del input incluso en el peor de los casos, asintóticamente" (para inputs grandes).

Formalismo de notación O-grande

Sea $T: \mathbb{N} \longrightarrow \mathbb{N}$

Def:

$T(n) \sim O(f(n)) \iff \exists\ c \in \mathbb{R}_{>0},\ n_0 \in \mathbb{N}$

tal que $T(n) \leq c\, f(n)$, si $n \geq n_0$

"T es eventualmente menor que un multiplo de f"

IMPORTANTE: c y n_0 no dependen de n.

Def: De manera semejante, para cotas inferiores decimos

$T(n) \sim \Omega(g(n)) \iff \exists\ E \in \mathbb{R}_{>0},\ n_0 \in \mathbb{N}$

tal que $T(n) \geq E \cdot g(n)$, si $n \geq n_0$.

"T es eventualmente más grande que un multiplo de g"

Def: $T(n) \sim \Theta(g(n)) \iff$

$T(n) = O(g(n))$ y $T(n) = \Omega(g(n))$

"T tiene el mismo orden asintótico que g"

Ejemplo: Si $T(n) = n^k + a_{k-1} n^{k-1} + \cdots + a_1$ es un polinomio entonces

$$T \sim \Theta(n^k)$$

Ejemplo:

$$\max\{f(n), g(n)\} \sim \Theta(f(n) + g(n))$$