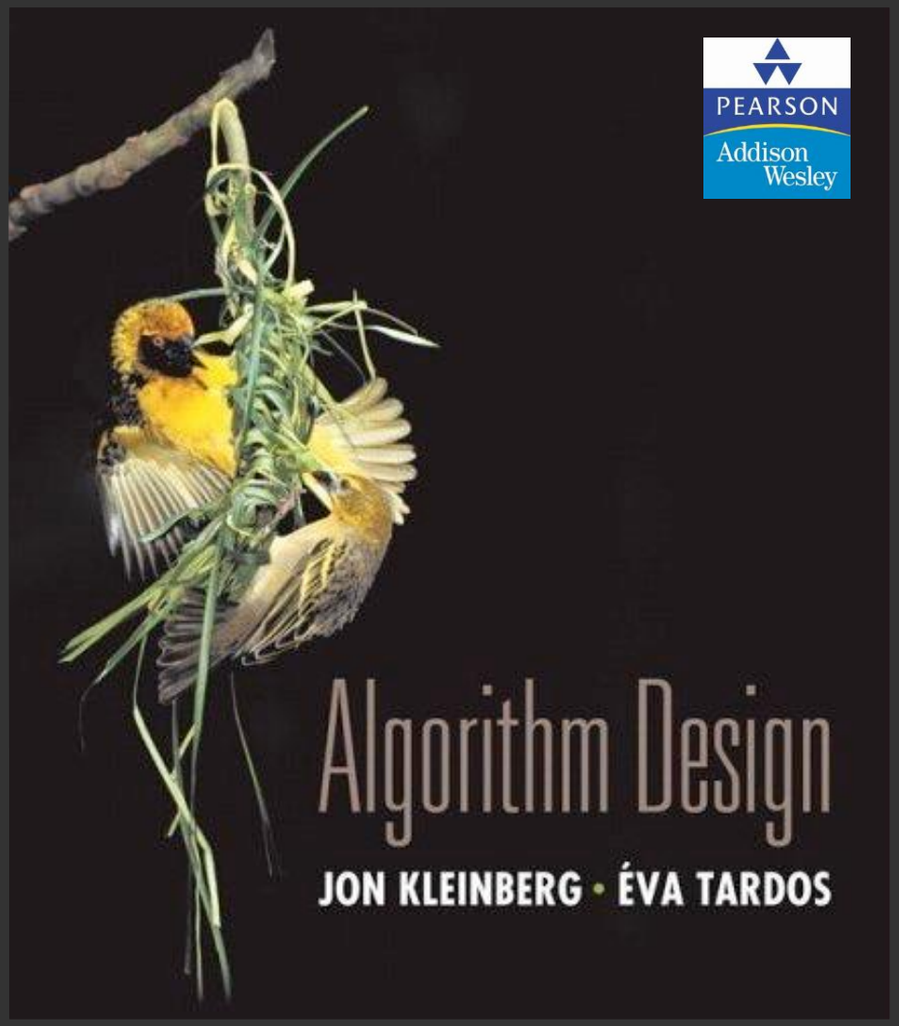
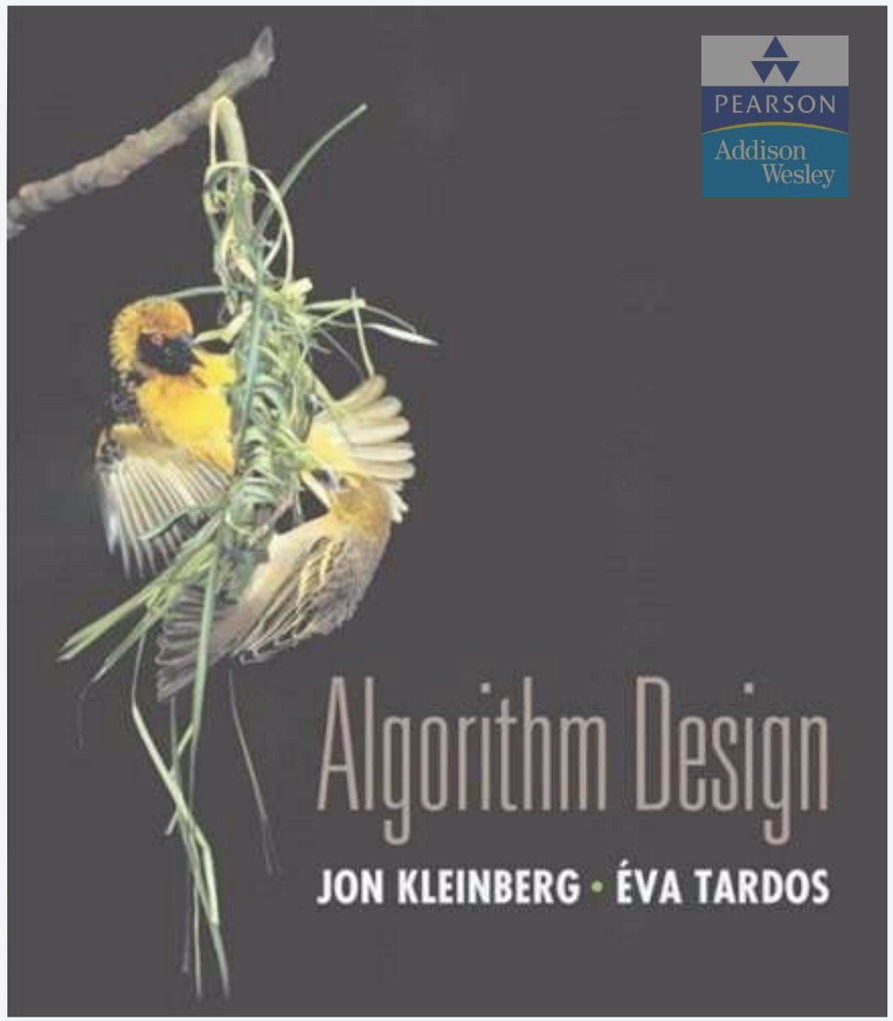


2. Análisis de algoritmos

- *tratabilidad computacional*
- *orden asintótico de crecimiento*
- *estudio de los tiempos de ejecución habituales*

Diapositivas de Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

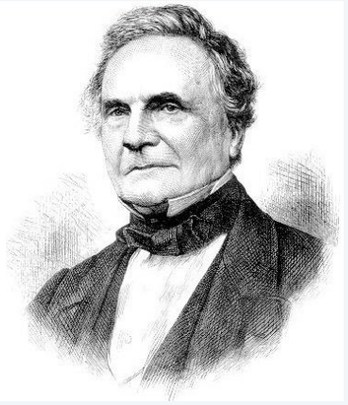


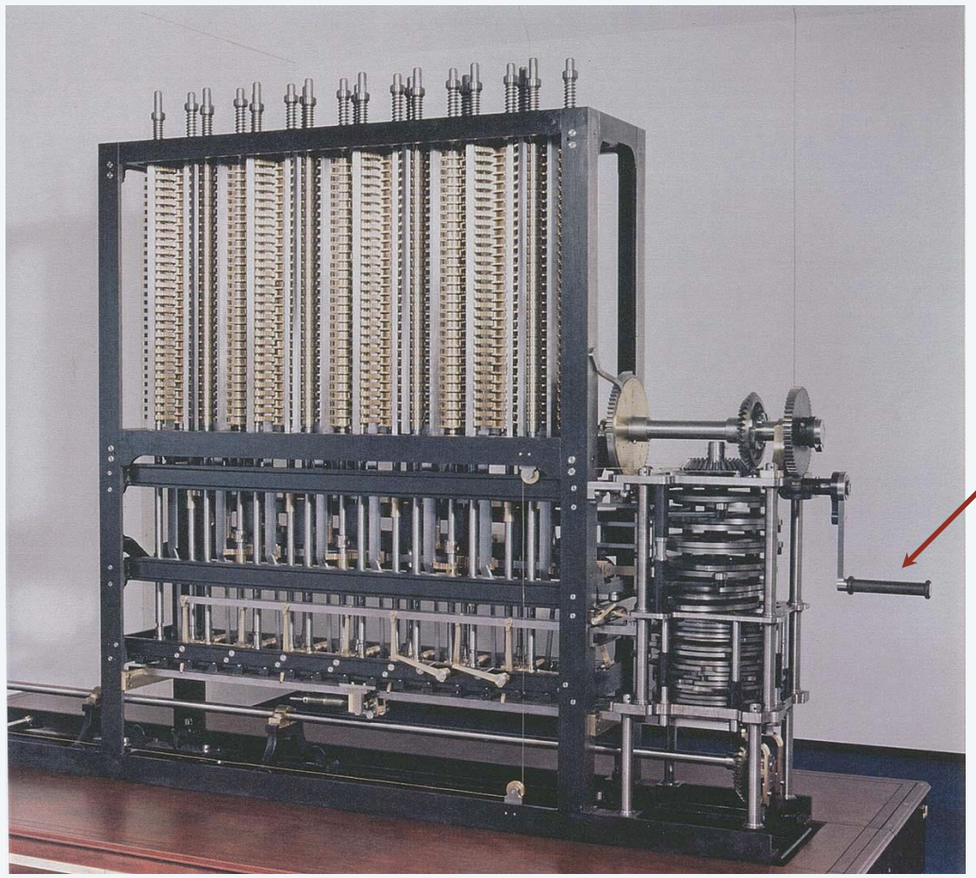
2. Análisis de algoritmos

- *tratabilidad computacional*
- *orden asintótico de crecimiento*
- *estudio de los tiempos de ejecución habituales*

Un pensamiento sorprendentemente moderno

" Tan pronto como exista una Máquina Analítica, ésta guiará necesariamente el futuro de la ciencia. Siempre que se busque algún resultado con su ayuda, surgirá la pregunta: ¿con qué método de cálculo puede la máquina llegar a ese resultado en el menor tiempo posible?" *Charles Babbage (1864)*





¿cuántas veces hay que girar la manivela?

Máquina Analítica

Fuerza bruta

Fuerza bruta. Para muchos problemas no triviales, existe un algoritmo natural de búsqueda por fuerza bruta que comprueba todas las soluciones posibles.

- Típicamente toma 2^n tiempo o peor para entradas de tamaño n.
- Inaceptable en la práctica.

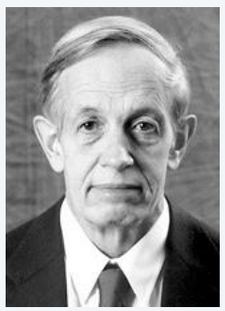
Tiempo de ejecución polinómico

Propiedad de escala deseable. Cuando el tamaño de la entrada se duplica, el algoritmo debería ralentizarse como máximo un factor constante C.

Def. Un algoritmo es de tiempo polinómico si se cumple la propiedad de escalado anterior.

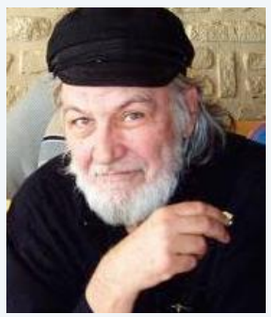
Existen las constantes c > 0 y d > 0 tales que,
para cada entrada de tamaño n, el tiempo de ejecución del algoritmo
está acotado superiormente por **cn^d** pasos computacionales primitivos. ← elija $C = 2^d$


von Neumann (1953)


Nash (1955)


Gödel (1956)


Cobham (1964)


Edmonds (1965)


Rabin (1966)

Tiempo de ejecución polinómico

Decimos que un algoritmo es eficiente si tiene un tiempo de ejecución polinómico.

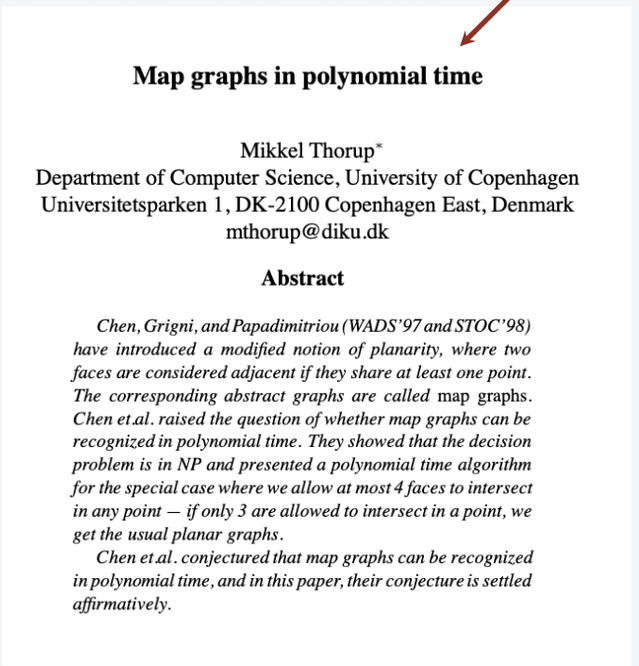
Justificación. ¡Realmente funciona en la práctica!

- En la práctica, los algoritmos poli-tiempo que se desarrollan tienen constantes y exponentes bajos.
- Superar la barrera exponencial de la fuerza bruta suele dejar al descubierto alguna estructura crucial del problema.

Excepciones. Algunos algoritmos poli-tiempo tienen constantes altas y/o exponentes, y/o son inútiles en la práctica.

Q. ¿Qué prefieres $20\,n^{120}$ vs. $n^{1 + 0.02 \ln n}$?

n^{120}



Map graphs in polynomial time

Mikkel Thorup*
Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark
mthorup@diku.dk

Abstract

Chen, Grigni, and Papadimitriou (WADS'97 and STOC'98) have introduced a modified notion of planarity, where two faces are considered adjacent if they share at least one point. The corresponding abstract graphs are called map graphs. *Chen et al. raised the question of whether map graphs can be recognized in polynomial time. They showed that the decision problem is in NP and presented a polynomial time algorithm for the special case where we allow at most 4 faces to intersect in any point — if only 3 are allowed to intersect in a point, we get the usual planar graphs.*

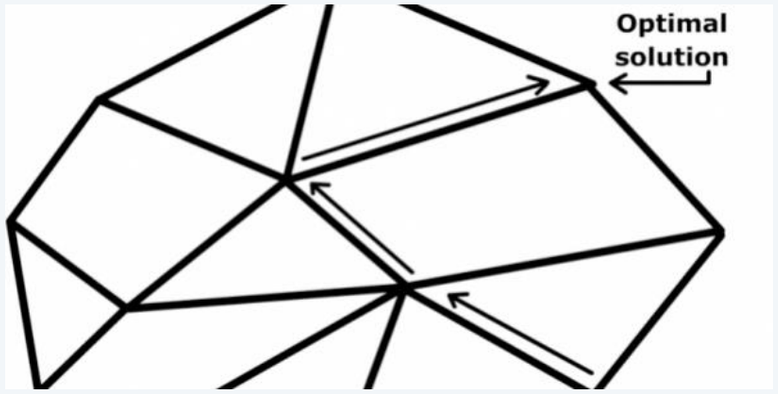
Chen et al. conjectured that map graphs can be recognized in polynomial time, and in this paper, their conjecture is settled affirmatively.

Análisis del peor caso

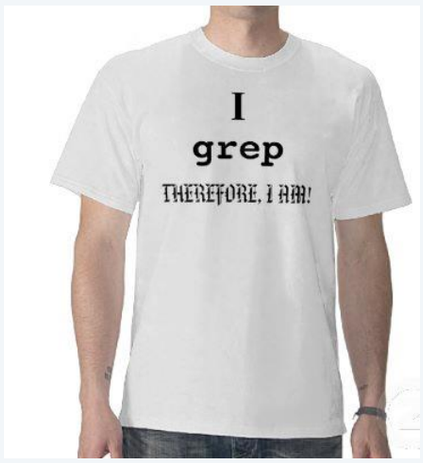
En el peor de los casos. Tiempo de ejecución garantizado para cualquier entrada de tamaño n.

- Generalmente capta la eficiencia en la práctica.
- Opinión draconiana, pero difícil de encontrar una alternativa eficaz.

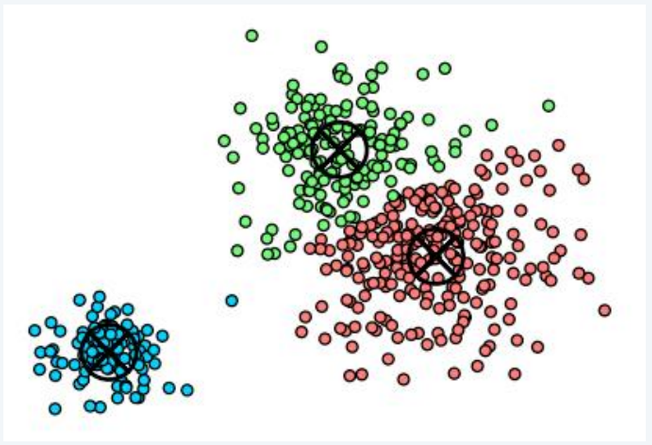
Excepciones. Algunos algoritmos de tiempo exponencial se utilizan ampliamente en la práctica porque los casos más desfavorables parecen ser raros.



algoritmo simplex



Linux grep



algoritmo k-means

Tipos de análisis

En el peor de los casos. Tiempo de ejecución garantizado para cualquier entrada de tamaño n. Ej. Heapsort requiere como máximo $2n\log_2 n$ comparaciones para ordenar n elementos.

Probabilístico. Tiempo de ejecución esperado de un algoritmo aleatorio.

Ej. El número esperado de comparaciones para quicksort n elementos es $\sim 2n \ln n$.

Amortizado. Tiempo de ejecución en el peor de los casos para cualquier secuencia de n operaciones. Ej. Partiendo de una pila vacía, cualquier secuencia de n *operaciones de* inserción y extracción requiere $O(n)$ pasos computacionales primitivos utilizando una matriz de redimensionamiento.

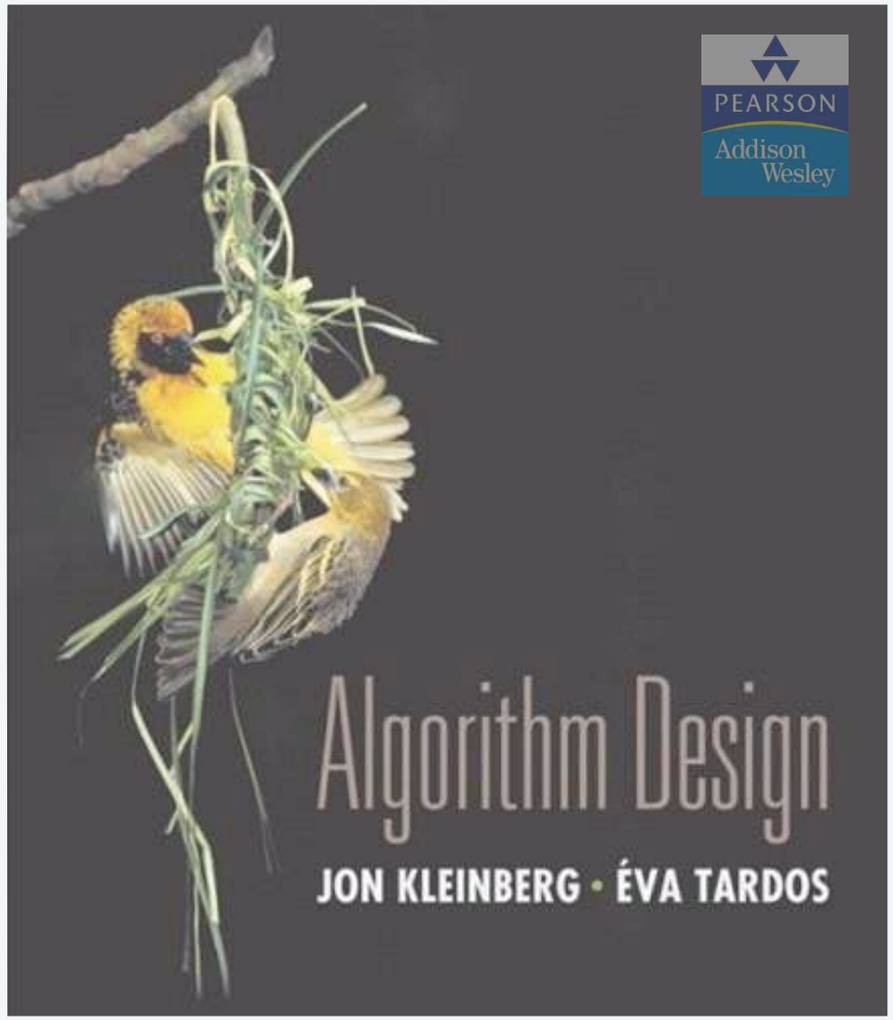
Caso medio. Tiempo de ejecución esperado para una entrada aleatoria de tamaño n. Ej. El número esperado de comparaciones de caracteres realizadas por 3-way radix quicksort en n cadenas uniformemente aleatorias es $\sim 2n \ln n$.

También. Análisis suavizado, análisis competitivo, ...

Por qué es importante

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long



2. Análisis de algoritmos

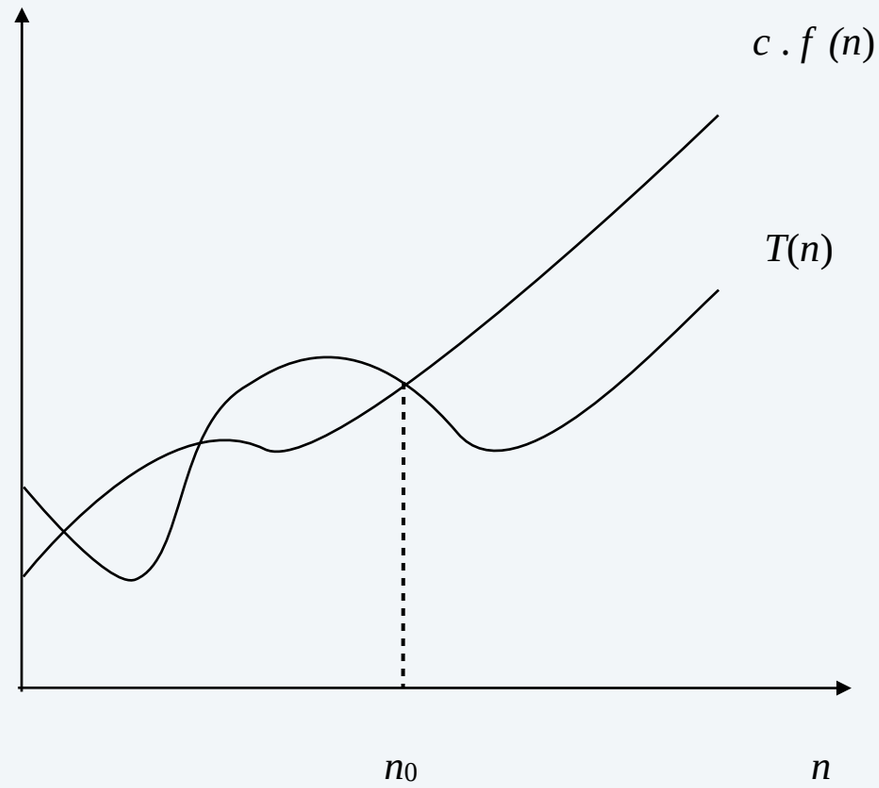
- *trazabilidad computacional*
- *orden asintótico de crecimiento*
- *estudio de los tiempos de funcionamiento habituales*

Notación "O mayúscula"

Cotas superiores. $T(n)$ es $O(f(n))$ si existen las constantes $c > 0$ y $n_0 \geq 0$ tales que $T(n) \leq c \cdot f(n)$ para todo $n \geq n_0$.

Ej. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ es $O(n^2)$. ← elija c = 50, n_0 = 1
- $T(n)$ también es $O(n^3)$.
- $T(n)$ no es ni $O(n)$ ni $O(n \log n)$.



Uso típico. La ordenación por inserción hace $O(n^2)$ comparaciones para ordenar n elementos.

Definición alternativa. $T(n)$ es $O(f(n))$ si $\limsup_{n \to \infty} \frac{T(n)}{f(n)} < \infty$.

Abusos notacionales

Signo de igual. $O(f(n))$ es un conjunto de funciones, pero los informáticos suelen escribir $T(n) = O(f(n))$ en lugar de $T(n) \in O(f(n))$.

Ej. Consideremos $f(n) = 5n^3$ y $g(n) = 3n^2$.

- Tenemos $f(n) = O(n^3) = g(n)$.
- Por tanto, $f(n) = g(n)$. X

Dominio. El dominio de $f(n)$ suele ser los números naturales $\{0, 1, 2, \ldots\}$.

- A veces nos limitamos a un subconjunto de los números naturales. Otras veces nos extendemos a los reales.

Funciones no negativas. Al utilizar la notación O grande, suponemos que las funciones implicadas son (asintóticamente) no negativas.

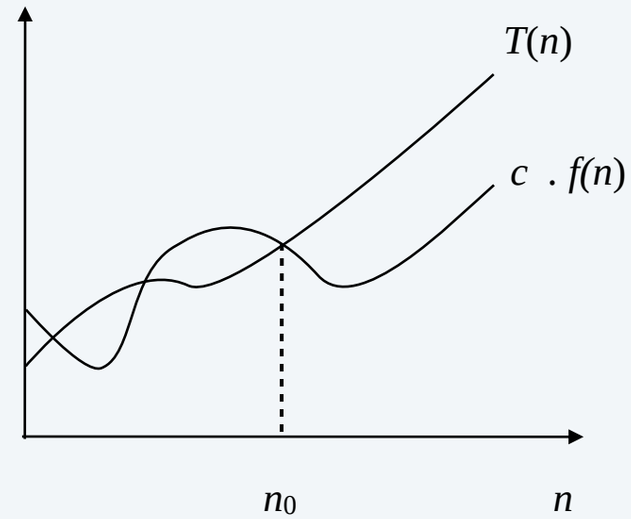
En resumen. Está bien abusar de la notación, pero no está bien usarla mal.

Notación Omega mayúscula

Cotas inferiores. $T(n)$ es $\Omega(f(n))$ si existen las constantes $c > 0$ y $n_0 \geq 0$ tales que $T(n) \geq c \cdot f(n)$ para todo $n \geq n_0$.

Ej. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ es tanto $\Omega(n^2)$ como $\Omega(n)$. ← elija c = 32, n_0 = 1
- $T(n)$ no es ni $\Omega(n^3)$ ni $\Omega(n^3 \log n)$.



Uso típico. Cualquier algoritmo de ordenación basado en comparaciones requiere $\Omega(n \log n)$ comparaciones en el peor de los casos.

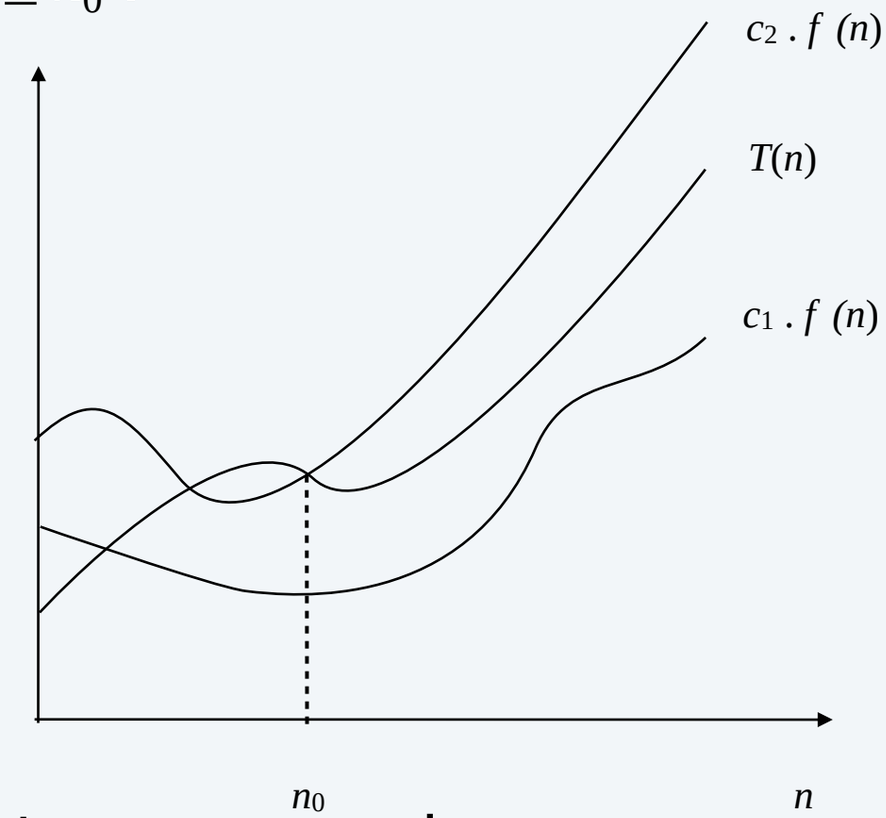
Afirmación sin sentido. Cualquier algoritmo de ordenación basado en comparaciones requiere al menos $O(n \log n)$ comparaciones en el peor de los casos.

Notación Theta mayúscula

Límites estrechos. $T(n)$ es $\Theta(f(n))$ si existen las constantes $c_1 > 0$, $c_2 > 0$, y $n_0 \geq 0$ tales que $c_1 . f(n) \leq T(n) \leq c_2 . f(n)$ para todo $n \geq n_0$.

Ej. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ es $\Theta(n^2)$. ← elija $c_1 = 32$, $c_2 = 50$, $n_0 = 1$
- $T(n)$ no es ni $\Theta(n)$ ni $\Theta(n^3)$.



Uso típico. Mergesort realiza $\Theta(n \log n)$ comparaciones para ordenar n elementos.

Datos útiles

Proposición. Si, $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0$ entonces $f(n)$ es $\Theta(g(n))$.

Pf. Por definición del límite, existe n_0 tal que para todo $n \geq n_0$

$$\frac{1}{2} c \leq \frac{f(n)}{g(n)} \leq 2c$$

- Así, $f(n) \leq 2\, c\ \mathrm{g}(n)$ para todo $n \geq n_0$, lo que implica que $f(n)$ es $O(g(n))$.
- Del mismo modo, $f(n) \geq \frac{1}{2}\, c\ \mathrm{g}(n)$ para todo $n \geq n_0$, lo que implica que $f(n)$ es $\Omega(g(n))$.

Proposición. Si , $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ entonces $f(n)$ es $O(g(n))$.

Límites asintóticos para algunas funciones comunes

Polinomios. Sea $T(n) = a_0 + a_1 \, n + \ldots + a \, n_d{}^d$ con $a_d > 0$. Entonces, $T(n)$ es $\Theta(n^d)$.

Pf. $$\lim_{n \to \infty} \frac{a_0 + a_1 n + \ldots + a_d n^d}{n^d} = a_d > 0$$

Logaritmos. $\Theta(\log_a n)$ es $\Theta(\log_b n)$ para cualquier constante ~~$a, b > 0$~~.

no es necesario especificar la base (suponiendo que sea una constante)

Logaritmos y polinomios. Para todo $d > 0$, $\log n$ es $O(n^d)$.

Exponenciales y polinomios. Para cada $r > 1$ y cada $d > 0$, n^d es $O(r^n)$.

Pf. $$\lim_{n \to \infty} \frac{n^d}{r^n} = 0$$

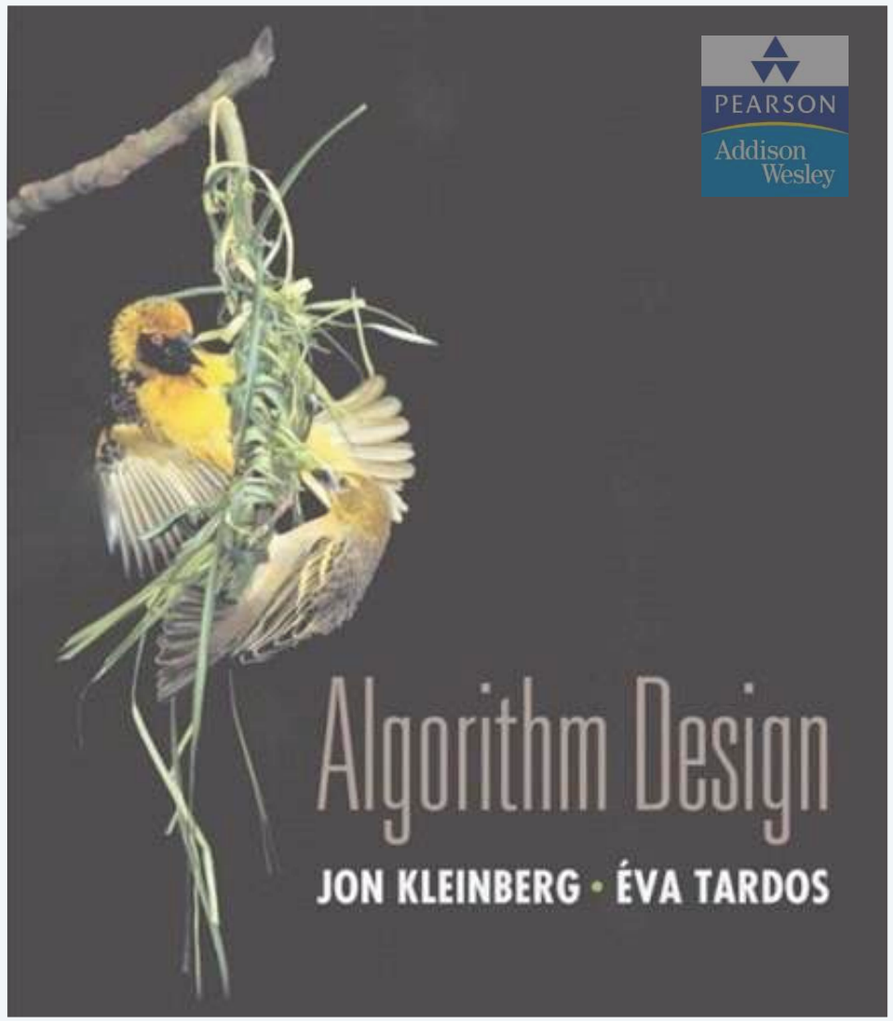
Notación Big-Oh con múltiples variables

Límites superiores. $T(m, n)$ es $O(\mathrm{f}\ (m, n))$ si existen las constantes $c > 0$, $m_0 \geq 0$ y $n_0 \geq 0$ tales que $T(m, n) \leq c \cdot f\ (m, n)$ para todo $n \geq n_0$ y $m \geq m_0$.

Ej. $T(m, n) = 32mn^2 + 17mn + 32n^3$.

- $T(m, n)$ es tanto $O(mn^2 + n^3)$ como $O(mn^3)$.
- $T(m, n)$ no es ni $O(n^3)$ ni $O(mn^2)$.

Uso típico. La búsqueda de ancho primero tarda $O(m + n)$ en encontrar el camino más corto de s a t en un dígrafo.



2. Análisis de algoritmos

- *trazabilidad computacional*
- *orden asintótico de crecimiento*
- *estudio de los tiempos de funcionamiento habituales*

Tiempo lineal: O(n)

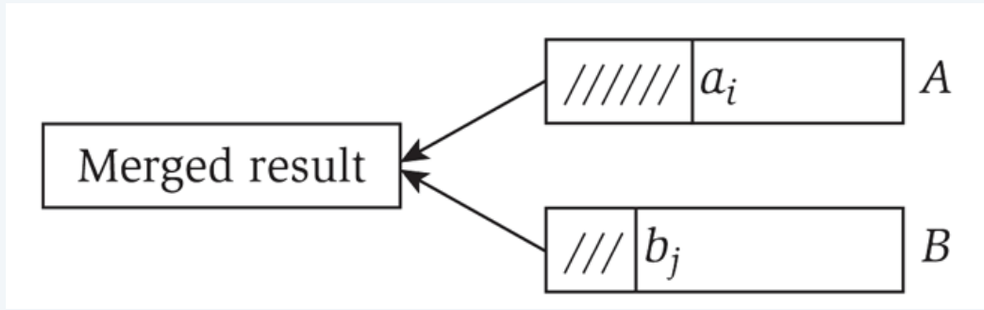
Tiempo lineal. El tiempo de ejecución es proporcional al tamaño de la entrada.

Calcular el máximo. Calcular el máximo de n números a_1 , ..., a_n .

```
max ← a1
for i = 2 to n {
   if (ai > max)
      max ← ai
}
```

Tiempo lineal: O(n)

Combinar. Combina dos listas ordenadas $A = a_1, a_2, \ldots, a_n$ con $B = b_1, b_2, \ldots, b_n$ en un todo ordenado.




```
i = 1, j = 1
while (ambas listas no están vacías) {
   if (a_i £ b_j ) añade a_i a la lista de salida e incrementa i
   else          añade b_j a la lista de salida e incrementa j
}
añadir el resto de la lista no vacía a la lista de salida
```

Afirmación. Fusionar dos listas de tamaño n lleva $O(n)$ tiempo.

Prueba. Después de cada comparación, la longitud de la lista de salida aumenta en 1.

Tiempo lineal: O(n log n)

Tiempo O(n log n). Surge en algoritmos de divide y vencerás.

Ordenación. Mergesort y heapsort son algoritmos de ordenación que realizan $O(n \log n)$ comparaciones.

Mayor intervalo vacío. Dadas n marcas de tiempo x_1 , ..., x_n en las que llegan copias de un fichero a un servidor, ¿cuál es el mayor intervalo en el que no llega ninguna copia del fichero?

Solución O(n log n). Ordena las marcas de tiempo. Escanea la lista ordenada, identificando la máxima distancia entre marcas de tiempo sucesivas.

Tiempo cuadrático: $O(n^2)$

Ej. Enumerar todos los pares de elementos.

Par de puntos más cercano. Dada una lista de n puntos en el plano (x_1, y_1), ..., (x_n, y_n), hallar el par más cercano entre sí.

Solución $O(n^2)$. Prueba todos los pares de puntos.

```
min ← (x1 - x )2^2 + (y1 - y )2^2
for i = 1 to n {
   for j = i+1 to n {
      d ← (xi - x )j^2 + (yi - y )j^2
      if (d < min)
         min ← d
   }
}
```

Observación. $\Omega(n2)$ parece inevitable, pero esto es sólo una ilusión. (véase el Capítulo 5)

Tiempo cúbico: $O(n^3)$

Ej. Enumerar todos los triples de elementos.

Conjuntos disjuntos. Dados n conjuntos S_1 , ..., S_n cada uno de los cuales es un subconjunto de $1, 2, ..., n$, ¿existe algún par de ellos que sea disjunto?

$O(n^3)$ solución. Para cada par de conjuntos, determinar si son disjuntos.

```
foreach conjunto Si {
   foreach otro conjunto Sj {
      foreach elemento p de Si {
         determinar si p también pertenece a Sj
      }
      if (ningún elemento de Si pertenece a S )j
         informan de que Si y Sj son disjuntos
   }
}
```

Tiempo polinómico: $O(n^k)$

Conjunto independiente de tamaño k. Dado un grafo, ¿existen k nodos tales que no haya dos unidos por una arista?

k es una constante

Solución $O(n^k)$. Enumerar todos los subconjuntos de k nodos.

```
for each subconjunto S de k nodos {
   comprobar si S es un conjunto independiente
   if (S es un conjunto independiente)
      informe S es un conjunto independiente
   }
}
```

- Comprobar si S es un conjunto independiente lleva $O(k^2)$ tiempo.
- Número de subconjuntos de k elementos =
- $O(k^2 n^k / k!) = O(n^k)$.

$$\binom{n}{k} = \frac{n(n-1)(n-2) \times \cdots \times (n-k+1)}{k(k-1)(k-2) \times \cdots \times 1} \leq \frac{n^k}{k!}$$

poli-tiempo para k=17,
pero no es práctico

Tiempo exponencial

Conjunto independiente. Dado un gráfico, ¿cuál es la cardinalidad máxima de un conjunto independiente?

Solución $O(n\, 2^{2\,n})$. Enumerar todos los subconjuntos.

```
S* ← ϕ

foreach subconjunto S de nodos {
   comprobar si S es un conjunto independiente
   if (S es el mayor conjunto independiente visto hasta ahora)
      actualizar S* ← S
   }
}
```

Tiempo sublineal

Búsqueda en una matriz ordenada. Dada una matriz ordenada A de n números, ¿se encuentra un determinado número x en la matriz?

Solución O(log n). Búsqueda binaria.

```
lo ← 1, hi ← n
while (lo ≤ hi) {
   mid ← (lo + hi) / 2
   if (x < A[mid]) hi ← mid - 1
   else if (x > A[mid]) lo ← mid + 1
   else return yes
}
devolver no
```