# **Límites Computacionales**:
## Introducción a la Teoría de la Computación y de la Complejidad

*Borja Sotomayor*
*15 de julio de 2005*



Charla organizada por:


Cursillos de Julio 2005
http://www.eside.deusto.es/eventos/cursillos

## Índice

- Introducción
- Turing y su máquina
- Problemas indecidibles
- Notación asintótica
  - O(n), (n), (n), ...
- Clases de complejidad
  - P, NP, NP-Completo, NP-Dificil, ...
  - ¿P=NP?
- Rompiendo los límites



## Índice

- Introducción
- Turing y su máquina
- Problemas indecidibles
- Notación asintótica
  - O(n), (n), (n), ...
- Clases de complejidad
  - P, NP, NP-Completo, NP-Dificil, ...
  - ¿P=NP?
- Rompiendo los límites

## Introducción (I)

- La informática se asienta sobre una enorme base teórica
- Estos fundamentos apenas se ven en las carreras de informática y de ingeniería porque "no sirven para nada".
  - De acuerdo, no tienen ninguna utilidad práctica palpable. P.ej. ¿Cuanta gente acabará programando compiladores?
  - Pero, nos permiten entender mucho mejor la informática y, en concreto, sus *límites*.



## Introducción (II)

- Muchas ciencias tienen límites teóricos:
  - La velocidad de la luz
    - Tal y como entendemos el universo actualmente, es imposible transmitir información más rápidamente que *c*
  - Cero absoluto
    - Límite inferior para las temperaturas.
  - Principio de indeterminación de Heisenberg [1]
    - No se puede determinar, simultáneamente y con precisión arbitraria, ciertos pares de variables físicas, como son, por ejemplo, la posición y la cantidad de movimiento de un objeto dado.
  - Teorema de la incompletud de Gödel [2]
    - En cualquier formalización consistente de las matemáticas que es lo bastante fuerte para definir el concepto de números naturales, se puede construir una afirmación que ni se puede demostrar ni se puede refutar dentro de ese sistema.
    - Ningún sistema consistente se puede usar para demostrarlo a él mismo.

## Introducción (III)

- La informática no iba a ser menos, y también tiene una serie de límites teóricos
- Esto límites están impuestos por el modelo computacional en el que se apoya *toda* la informática.
- Dos límites importantes:
  - Hay ciertas cosas que *nunca* podrán computarse.
  - Hay ciertas cosas que *nunca* podrán computarse *eficientemente*.



## Introducción (IV)

- Fundamentos teóricos de la informática
  - **Teoría de la computación**: ¿Qué es computable?
  - **Teoría de autómatas y lenguajes formales**: ¿Qué lenguajes puede reconocer un ordenador?
  - **Teoría de la complejidad**: ¿Cual es la complejidad de un problema dado? Es decir, ¿cuantos recursos consumirá?
- Fundamentos ligeramente menos teóricos:
  - Análisis de algoritmos
  - Metodología de la programación

## Introducción (V)

- En esta charla nos centraremos en la teoría de la computación y la complejidad.
- Las explicaciones serán muy informales. Hay muchos detalles y matices que no explicaremos.
- El objetivo de la charla es proporcionar una visión general, no una explicación matemáticamente exquisita.
  - El símbolo ☆ indica explicaciones avanzadas.
  - Para explicaciones más rigurosas, consultar la bibliografía.



## Índice

## Alan Turing (I)

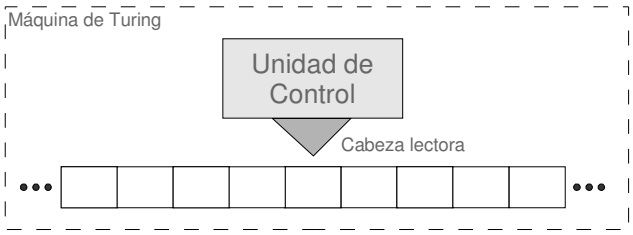
**Alan Turing**
Padre de la Informática
(1912-1954)



## Alan Turing (II)

- Sentó las bases de la teoría de la computación con su artículo "*On Computable Numbers, with an Application to the Entscheidungsproblem*" (1936)
- En este artículo presentó lo que hoy se llama la *máquina de Turing*.
  - Maquina abstracta capaz de *computar*.
  - Proporciona una definición matemática y precisa de *algoritmo*.

## La Máquina de Turing (I)






## La Máquina de Turing (II)

- La máquina de Turing tiene:
  - Una *cinta* infinita dividida en celdas. Cada celda puede tener un símbolo de un alfabeto previamente definido.
  - Una *cabeza lectora* que puede moverse a la izquierda, a la derecha, quedarse quieta, o escribir un símbolo en la "celda actual".
  - Una unidad de control que decide qué hacer en función de la celda actual y el "estado actual"
    - La máquina de Turing puede encontrarse en varios estados (similar a un autómata)

## La Máquina de Turing (III)

- Dos maneras de trabajar con la máquina de Turing:
  - Proporcionar una cadena de símbolos para que la máquina la transforme. P.ej. Sumar números.
  - Proporcionar una cadena de símbolos y verificar si cumple una propiedad (observar si la máquina de Turing termina su ejecución en un "estado de parada")
    - Verificar propiedades de otras máquinas de Turing, representadas como una cadena de símbolos.
- La máquina de Turing es una abstracción matemática que nos permite realizar demostración formales.
  - Programar en código máquina x86 es fácil en comparación...



## La Máquina de Turing (IV)

- La máquina de Turing es el modelo computacional sobre el que se apoya toda la informática.
- Marca los limites de lo computable.
  - Tesis de Turing-Church: "Si la máquina de Turing no puede computar un problema, ningún otro ordenador puede computarlo"
  - ¡El ordenador más potente del mundo!

## Introducción

- Turing y su máquina
- Problemas indecidibles
- Notación asintótica
  - O(n), (n), (n), ...
- Clases de complejidad
  - P, NP, NP-Completo, NP-Dificil, ...
  - ¿P=NP?
- Rompiendo los límites



## Problemas indecidibles (I)

- Hay ciertas cosas que una máquina de Turing no puede computar. Por lo tanto, ningún ordenador podrá computarlo.
  - "¡Es que a nadie se le ha ocurrido un algoritmo!" --> NO
  - Está demostrado matemáticamente.
  - Da igual cuanta memoria tenga, la velocidad del procesador, cuanto avance la informática...

    nunca nunca nunca nunca NUNCA NUNCA podrá computarlo.
- Problemas indecidibles.

## Problemas indecidibles (II)

- El problema de la parada («The halting problem»)
  - ¿Dada una máquina de Turing, podemos determinar si finalizará su ejecución en algún momento?
  - Otra manera de plantear el problema: ¿Dado un programa en Java, podemos detectar si caerá en un bucle infinito?
  - Turing demostró que el problema de la parada es indecidible.



## Problemas indecidibles (III)

- ☆ En general, el teorema de Rice dice que "ninguna pregunta no-trivial sobre el comportamiento o resultado de una máquina de Turing es decidible"
  - ¿Escribirá este programa el carácter "H" en algún momento?
  - ¿Será invocada la funcion foo() en algún momento?
- ¡Ojo! Podríamos resolver estos problemas para casos concretos, pero no *en general*.
  - De hecho, bajo el supuesto de que la computación dispone de recursos finitos, el problema de la parada puede resolverse por fuerza bruta.
  - No es práctico.
  - Si la respuesta es "No", entonces para continuar la búsqueda de la solución habría que añadir más recursos (ad nauseam)

## Problemas indecidibles (IV)

- Más ejemplos:
  - Problema de correspondencia de Post
    - Problema abstracto, pero útil para demostrar que otros problemas son indecidibles.
  - Ambigüedad de gramáticas
  - Wang tiles [11]
    - Curioso "juego" indecidible
  - En matemáticas:
    - Teorema de Goodstein [12]
    - Resolución de ecuaciones diofantinas generales [13]



## A mitad de camino...

- Hasta ahora hemos visto que hay ciertos problemas que un ordenador nunca podrá resolver.
  - Teoría de la Computación
  - Problemas indecidibles
- A continuación, vamos a hablar sobre problemas que un ordenador no puede resolver eficientemente.
  - Teoría de la Complejidad
  - Problemas intratables

## Índice

- Introducción
- Turing y su máquina
- Problemas indecidibles
- Notación asintótica
  - O(n), Ω(n), Θ(n), ...
- Clases de complejidad
  - P, NP, NP-Completo, NP-Dificil, ...
  - ¿P=NP?
- Rompiendo los límites



## Notación Asintótica (I)

- Antes de abordar la Teoría de la Complejidad, tenemos que conocer la notación asintótica.
- Es una manera muy útil de expresar el coste (en tiempo o espacio) de un algoritmo.

# Notación Asintótica (II)

```
funcion HacerAlgo(a: array de enteros)

   por cada elemento x de a
      // Op1
      // Op2
      // ...
      // Opk
   fin-por

fin-funcion
```

Numero *constante* $k$ de operaciones elementales

Asumiendo que cada operación elemental es 1 paso, el tiempo de ejecución de esta función "crece" al mismo ritmo que $n$, el numero de elementos.

| Número de elementos: | 1 | 2 | 3 | ... | $n$ |
|---|---|---|---|---|---|
| Número de pasos: | $1 \cdot k$ | $2 \cdot k$ | $3 \cdot k$ | ... | $n \cdot k$ |



# Notación Asintótica (III)

```
funcion HacerAlgo(m: matriz (n x n) de enteros)

   por cada fila f de m
      por cada columna c de m
         // Op1
         // Op2
         // ...
         // Opk
      fin-por
   fin-por

fin-funcion
```

Numero *constante* $k$ de operaciones elementales

Asumiendo que cada operación elemental es 1 paso, el tiempo de ejecución de esta función crece al mismo ritmo que $n^2$.

| Número de elementos: | 1 | 2 | 3 | ... | $n$ |
|---|---|---|---|---|---|
| Número de pasos: | $1 \cdot k$ | $4 \cdot k$ | $9 \cdot k$ | ... | $n^2 \cdot k$ |

## Notación Big-Oh (I)

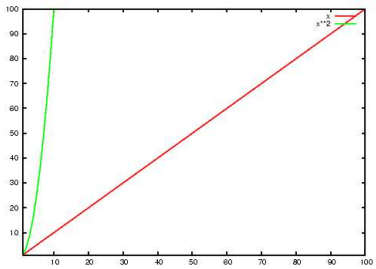
- Podemos expresar esto con notación asintótica:
  - $O(n)$ y $O(n^2)$
- $O(f(n))$ --> "Big-Oh"
- ☆De nuevo, me estoy dejando muchos detalles.
  - "Big-Oh" expresa una cota superior. Es decir, un algoritmo $O(n^2)$ requiere *como máximo* $n^2$ pasos (el algoritmo puede ejecutarse en menos pasos).
  - $O(n^2)$ no significa *exactamente* $n^2$ pasos.
    - $n^2 + 5n + 3 = O(n^2)$
    - $10000n^2 + 0.7n + 2 = O(n^2)$
  - Podemos pensar que la notación Big-Oh expresa el número máximo "aproximado" de pasos que requiere un algoritmo en función del tamaño de los datos de entrada (esta definición no es estrictamente correcta, pero nos sirve para lo que vamos a explicar a continuación)

☆ $a_n = O(b_n) \Leftrightarrow \exists c > 0, n_0 : \forall n \geq n_0 \ |a_n| \leq c|b_n|$

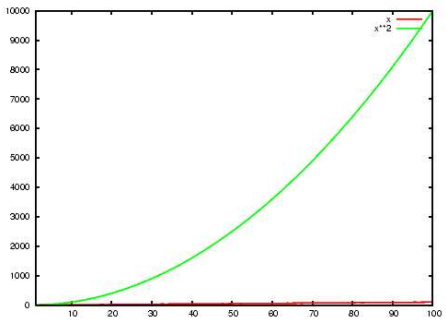


## Notación Big-Oh (II)

- Es interesante conocer el tiempo de ejecución de un algoritmo en notación asintótica para saber qué algoritmo es preferible.
  - Si tenemos que escoger entre un algoritmo $O(n^2)$ y $O(n)$, es preferible el algoritmo $O(n)$.



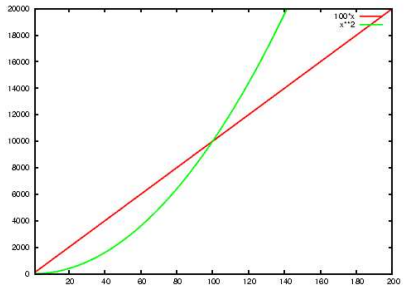
## Notación Big-Oh (III)






## Notación Big-Oh (IV)

- ¡Ojo! La notación Big-Oh puede ser engañosa.
  - $100n = O(n)$
  - $n^2 = O(n^2)$
  - $100n$ no es "mejor" que $n^2$ hasta que $n=100$
  - Constante multiplicativa. Es habitual indicar si es pequeña o grande.

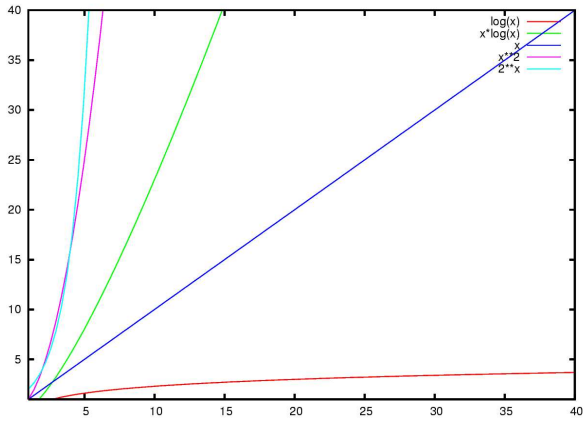


## Notación Big-Oh (V)

- De peor a mejor...
  - $O(k) \rightarrow$ Tiempo constante
  - $O(\log n) \rightarrow$ Tiempo logarítmico
    - Búsqueda binaria
  - $O(n \cdot \log n)$
    - Algoritmos de búsqueda (Mergesort, Quicksort, ...)
  - $O(n) \rightarrow$ Tiempo lineal
  - $O(n^2) \rightarrow$ Tiempo cuadrático
    - Algoritmos de búsqueda (Burbuja, Insertion, ...)
  - $O(n^k) \rightarrow$ Tiempo polinómico
    - $O(n^2), O(n^3), O(n^4), \ldots$
  - $O(a^n) \rightarrow$ Tiempo exponencial
    - $O(2^n), O(10^n), \ldots$



## Notación Big-Oh (VI)



## Más notación asintótica

- $(f(n))$ – Big-Omega
  - Cota inferior
  - P.ej. el tiempo de ejecución de todo algoritmo de búsqueda que no se ejecute en paralelo es $(n \cdot \log n)$
- $\Theta(f(n))$- Big-Theta
  - Cota inferior y superior
  - $\Theta(f(n)) \Leftrightarrow O(f(n))$ y $(f(n))$



## Índice

- Introducción
- Turing y su máquina
- Problemas indecidibles
- Notación asintótica
  - O(n), (n), (n), ...
- Clases de complejidad
  - P, NP, NP-Completo, NP-Dificil, ...
  - ¿P=NP?
- Rompiendo los límites

## Teoría de la Complejidad

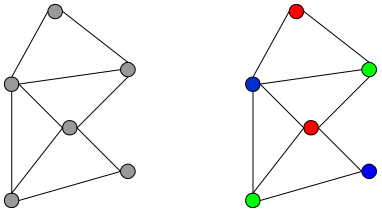
- La Teoría de la Complejidad se centra principalmente en los problemas de decisión.
- Problema de decisión
  - La definición exacta involucra lenguajes, máquinas de Turing, ...
  - Por simplificar: Es un problema en el que tenemos que decidir si una propiedad se cumple o no.
  - P.ej. "Dado un grafo, determinar si puedo colorearlo con tres colores"
  - Si sabemos resolver el problema de decisión, sabremos resolver su correspondiente "problema de búsqueda" ("Dado un grafo, buscar una 3-coloración valida"), aunque no necesariamente de manera eficiente.
  - Otro ejemplo:
    - Problema de búsqueda: "Dado un número, buscar sus factores" (819 = 7*9*13)
    - Problema de decisión: "Dado un número, ¿es divisible por un número entre 2 y $k$?"



## P y NP (I)

- La clase P contiene aquellos problemas de decisión que pueden resolverse en tiempo polinómico.
  - $O(n^k)$: $O(n^2)$, $O(n^3)$, $O(n^4)$, ...
- La clase NP contiene aquellos problemas de decisión para los que puede proporcionarse un "testigo" verificable en tiempo polinómico.
  - Problema de decisión: "Dado un grafo, determinar si puedo colorearlo con tres colores".
  - Si yo proporciono una 3-coloración válida ("el testigo"), evidentemente el grafo es 3-coloreable.
    - La 3-coloración es verificable en tiempo polinómico. Proceso cada nodo del grafo, y compruebo que sus nodos vecinos no tienen el mismo color (y, por supuesto, me aseguro de que el número total de colores es 3).

## P y NP (II)

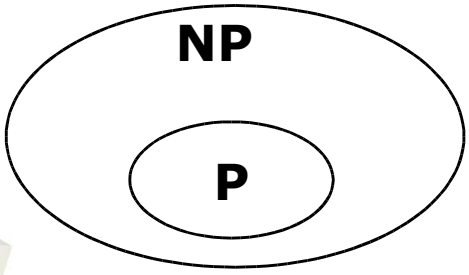




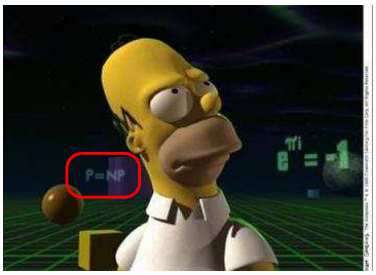
## P y NP (III)

- P es un subconjunto de NP ($P \subseteq NP$).
- Sin embargo, no sabemos si P=NP o si $P \subset NP$ (subconjunto propio)
- Mayor incógnita de la informática teórica:

**¿P=NP?**

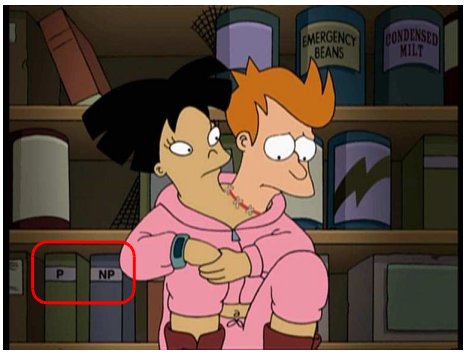


## P y NP (IV)






## P y NP (V)

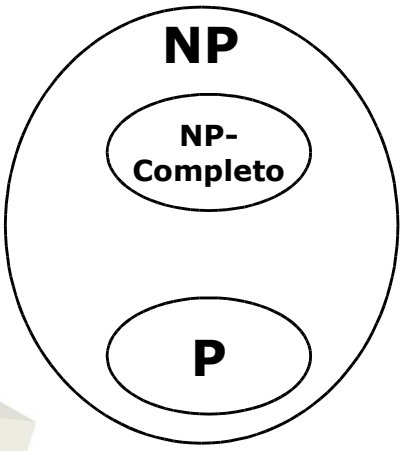


## Problemas NP-Completos (I)

- ¿Por qué es esto importante?
  - Si P≠NP, entonces hay ciertos problemas que *no* pueden resolverse en tiempo polinómico, únicamente en tiempo exponencial
    - Tiempo exponencial: ¡Prohibitivo!
  - En concreto, no podrían resolverse en tiempo polinómico los problemas NP-Completos.
    - Estos son los problemas "más difíciles".
    - Tienen la propiedad interesante de que si se encontrase un algoritmo polinómico para cualquiera de estos problemas, automáticamente resultaría que P=NP.
    - ➔ ☆ Todos los problemas en NP pueden reducirse a los problemas NP-Completos. Teorema Cook-Levin. [4]



## Problemas NP-Completos (II)



## ¿P=NP? (I)

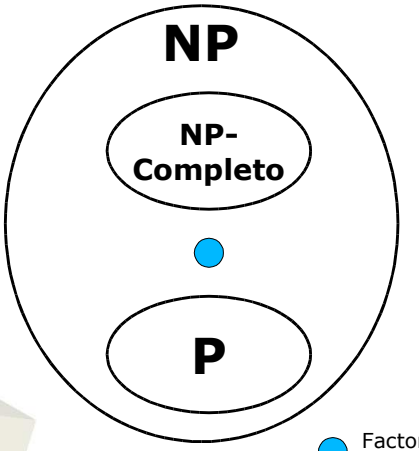
- Conjetura: P≠NP
- ¡Ojo! Todavía habría problemas en NP (pero no en P) que igual si tienen una solución en tiempo polinómico, pero que sencillamente no ha sido encontrada todavía.
  - P.ej.: En 2002 se demostró que el problema de determinar la primalidad de un número está en P. [3]



## ¿P=NP? (II)

- Venga, en serio, ¿Por qué es esto importante?
  - Problemas NP-Completos:
    - Prácticamente cualquier problema de planificación que se nos pueda ocurrir es NP-Completo: planificación de procesos en un procesador, planificación de aulas, horarios de profesores, etc.
    - El problema del vendedor ambulante («*Travelling Salesman problem*»)
    - Camino más corto en un grafo con pesos («*min-cost path in a weighted graph*»)
    - Problemas de interés matemático
      - 3-coloración de grafos, búsqueda de cliques, boolean satisfiability, ciclos hamiltonianos, etc.
    - Lista de problemas: [5]
  - Si P=NP, entonces seremos capaz de resolver estos problemas y muchos otros problemas en NP para los que no se conoce una solución en tiempo polinómico.
    - P.ej.: Factorización de enteros. Si P=NP, entonces el algoritmos RSA se va al traste.

## ¿P=NP? (III)




Factorización de enteros



## Índice

## Rompiendo los límites

- ¿Hay alguna manera de romper estos límites?
  - Estos límites están impuestos por el modelo computacional que utilizamos (la máquina de Turing). Es posible que, con un modelo computacional distinto, algunos de estos límites desparezcan.
  - Hipercomputación: La búsqueda de una "supermáquina de Turing" capaz de resolver problemas indecidibles.
- Posibles maneras de romper los límites:
  - Soluciones no óptimas a problemas NP-Completos
  - Otro modelo computacional
  - Computación cuántica



## Soluciones no optimas

- Soluciones no óptimas a problemas NP-Completos
  - Los problemas NP-Completos se refieren a solucionar un problema de manera óptima. En algunos casos, nos basta con una aproximación o con solucionar parte del problema.
  - *Aproximación*: Algoritmos que encuentran una solución aproximada ("casi óptima"). Sólo aplicable a algunos problemas, como el vendedor ambulante.
    - ➔☆ Algoritmos probabilistas, uso de heurísticas
  - *Subproblemas*: A veces no necesitamos resolver un problema en su formato general. Por ejemplo, encontrar el camino más corto en *cualquier* grafo con pesos (weighted graph) es un problema NP-Completo. Sin embargo, si ninguna arista tiene un peso negativo, entonces el problema puede resolverse en tiempo polinómico (Dijkstra)

## Otro modelo computacional

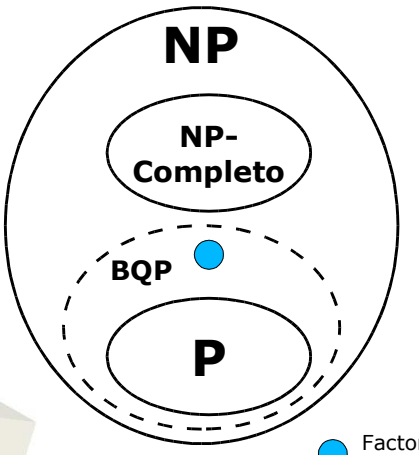
- Otro modelo computacional
  - Es posible que exista otro modelo computacional, distinto a la máquina de Turing, en el que existan menos límites.
  - Se han propuesto muchos modelos alternativos, pero todos han resultado ser equivalentes a la máquina de Turing.



## Computación cuántica (I)

- Computación cuántica [6]
  - Se basa en fenómenos de la mecánica cuántica para almacenar información y realizar computaciones.
- De momento, se sabe lo siguiente sobre la computación cuántica:
  - No puede resolver problemas indecidibles.
  - BQP es el conjunto de problemas que pueden resolverse en tiempo polinómico con un ordenador cuántico. Se sabe que P es un subconjunto de BQP.
- Conjeturas:
  - P es un subconjunto estricto de BQP. Es decir, hay problemas que pueden resolverse en tiempo polinómico con un ordenador cuántico pero que sólo pueden resolverse en tiempo exponencial con ordenadores clásicos. P.ej.: Factorización de enteros.
  - BQP y NPC son conjuntos disjuntos. Es decir, no podemos utilizar un computador cuántico para resolver eficientemente un problema NP-Completo.

## Computación cuántica (II)






## Computación cuántica (III)

- Factorización de enteros en un ordenador cuántico
  - Algoritmo de Shor [7]
  - Puede factorizar un número entero en tiempo polinómico.
  - Si llegase a construirse un ordenador cuántico práctico, podría servir para romper el algoritmo RSA.
  - Sin embargo, de momento sólo se ha conseguido construir un ordenador cuántico capaz de factorizar el número 15 (3 y 5).
    - → Experimento realizado por IBM [8]

## Computación cuántica (IV)

- Se conjetura que quizás es posible elaborar un nuevo modelo computacional basado en la computación cuántica.
- Esta conjetura ha sido recibida con escepticismo.



## Conclusiones

- ¿Por qué nos hace falta saber...
  - ... matemáticas?
    - Para aprender a razonar y pensar lógicamente Para poder entender los fundamentos teóricos de las informática es especialmente importante la *matemática discreta* y, en menor medida, el álgebra (abstracta y lineal) y la lógica formal.
  - ... teoría de la computación?
    - Para saber cuales son los límites de lo computable.
  - ... teoría de la complejidad?
    - Para saber qué problemas no pueden ser resueltos eficientemente.
  - ... algoritmia?
    - Para saber qué tipo de algoritmo es el mejor para resolver cierto problema eficientemente. Para poder comparar distintas soluciones y ver cual es la mejor.

# ¿Preguntas?

**Borja Sotomayor**
**Department of Computer Science**
**University of Chicago**
**borja@cs.uchicago.edu**
**http://people.cs.uchicago.edu/~borja/**



# Bibliografía

[1] Principio de indeterminación de Heisenberg.
http://es.wikipedia.org/wiki/Principio_de_indeterminaci%C3%B3n_de_Heisenberg
[2] Teorema de la incompletud de Gödel.
http://es.wikipedia.org/wiki/Teorema_de_la_incompletud_de_G%C3%B6del
[3] Manindra Agrawal, Neeraj Kayal y Nitin Saxena. *PRIMES is in P.*
http://www.cse.iitk.ac.in/news/primality.html
[4] John E. Hopcroft, Rajeev Motwani, y Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.
[5] Michael Garey y David Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness.* Freeman, 1979.
[6] Computación cuántica.
http://es.wikipedia.org/wiki/Computaci%C3%B3n_cu%C3%A1ntica
[7] Shor's algorithm.
http://en.wikipedia.org/wiki/Shor%27s_algorithm
[8] IBM's Test-Tube Quantum Computer Makes History: First demonstration of Shor's historic factoring algorithm.
http://domino.research.ibm.com/comm/pr.nsf/pages/news.20011219_quantum.html
[9] Kim B. Bruce, Robert L. Scot Drysdale. y Charles Kelemen. *Why Math?*
*http://cs.williams.edu/~kim/ftp/WhyMath.pdf*
[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein. *Introduction to Algorithms*. MIT Press.
[11] Wang Tiles.
http://en.wikipedia.org/wiki/Wang_tile
[12] Goodstein's theorem
http://en.wikipedia.org/wiki/Goodstein%27s_theorem
[13] Diophantine equation
http://en.wikipedia.org/wiki/Diophantine_equation