Intro a las Menciones: Computación Científica - IV - Algoritmos y precisión

Graeme Candlish

Semestre I 2018

Institúto de Física y Astronomía, UV
graeme.candlish@ifa.uv.cl

Contenido

¿Qué es un algoritmo?

Algoritmo

Como vimos en la clase anterior, un algoritmo es como una receta: un conjunto de instrucciones para completar una tarea.

Algoritmos

Qué es un algoritmo?

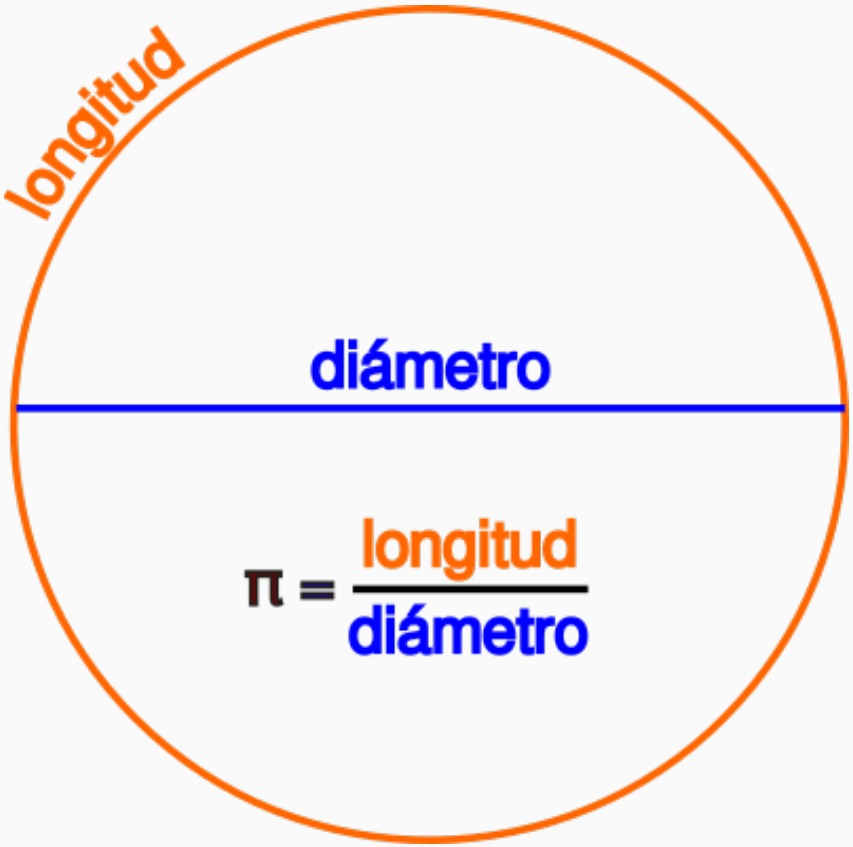
Un conjunto prescrito de instrucciones o reglas bien definidas cuyo fin es obtener una solución. La palabra viene del nombre del matemático, Mohammed ibn-Musa al-Khwarizmi, que formaba parte de la corte real en Baghdad, y vivió desde el año 780 hasta 850 (aprox).

Algoritmos

- Los algoritmos son independientes de los medios utilizados para completar la tarea.
- El mismo algoritmo podría implementarse con lapiz y papel, un computador, un abaco, etc.
- Por lo tanto, son independientes del *lenguaje de programación* que usamos.
- Un programa de computador es simplemente una implementación o realización de un algoritmo.

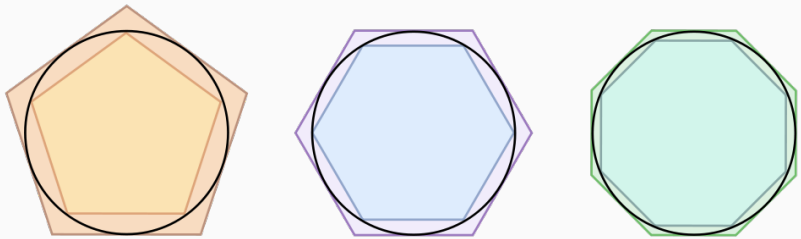
Aproximación del número π

- π es un *número irracional* y es la relación entre la longitud de una circunferencia y su diámetro.



Aproximación del número π

- π es un *número irracional* y es la relación entre la longitud de una circunferencia y su diámetro.
- El matemático Chino Liu Hui (225-295) hizo una aproximación de π en el siglo III.



Aproximación del número π

- Liu Hui, usando polígonos de 3072 lados, encontró:

$$\pi \approx 3,1416$$

Aproximación del número π

- El matemático y astrónomo Chino Zu Chongzhi (429-500) y su hijo Zu Gengzhi (480-525) usaron polígonos de 24576 lados...

$$3,1415926 < \pi < 3,1415927$$

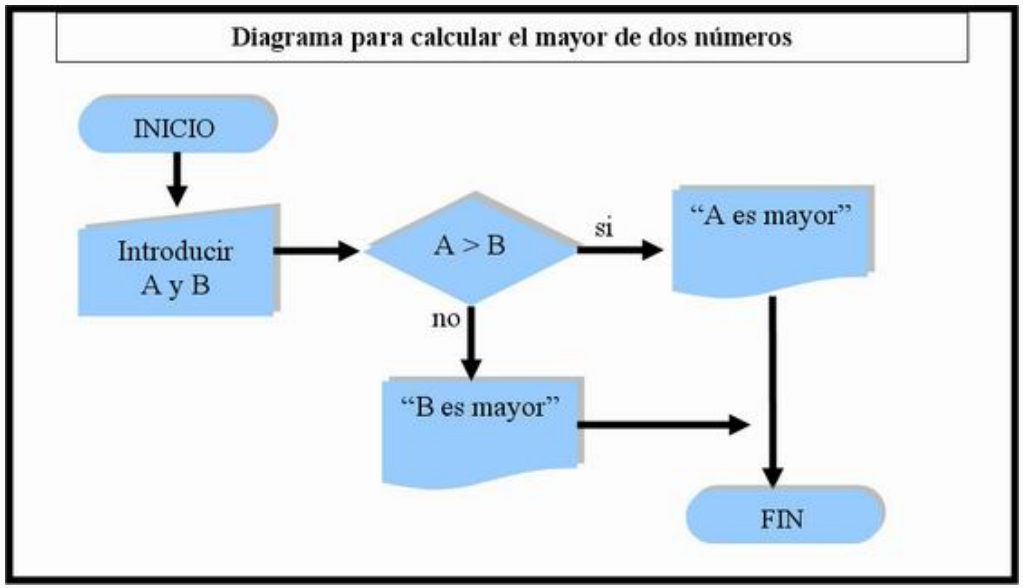
- Nadie mejoró este resultado durante los siguientes 1000 años!

Precisión y rapidez de un algoritmo

- La precisión del resultado y la rapidez en obtenerla dependen del algoritmo usado.
- Si el algoritmo está diseñado mal, el problema será mucho más difícil de resolver.

Diseño del algoritmo

- El primer paso para resolver un problema computacionalmente es diseñar el algoritmo.
- Del mismo modo que un edificio no puede construirse sin un plan de arquitecto, deberíamos planificar nuestro método para resolver el problema antes de programar.



Un ejemplo: ordenar números

- Dado un conjunto de números, queremos ordenarlos en orden ascendiente.
- ¿Cómo podemos hacer eso?

Un ejemplo: ordenar números

- No es necesario pensar en ningún lenguaje de programación específica cuando estamos diseñando un algoritmo!
- Primero, hay que pensar en los pasos lógicos que cualquier persona puede seguir para convertir una lista de números sin orden en una lista de números ordenados...

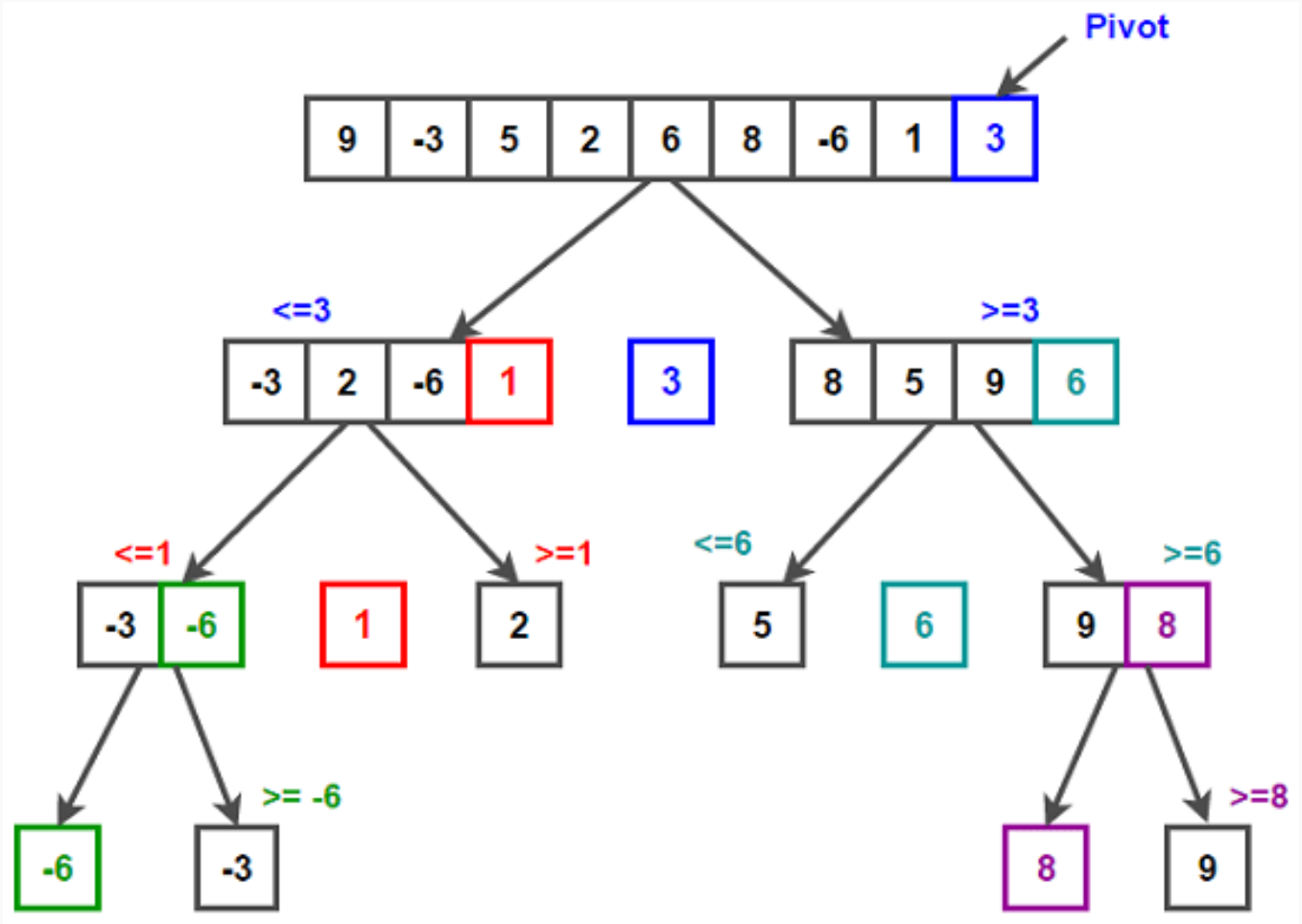
Un ejemplo de un algoritmo para ordenar números

Una posible solución a este problema es el siguiente algoritmo (que se llama "quicksort"):

1. Elegir un elemento de la lista, que se llama el elemento *pivote*.
2. Colocar todos los números menores que el elemento pivote a la izquierda, y todos los números mayores a la derecha.
3. Para cada una de estas sublistas, realizar los pasos (1) y (2), e igualmente para cada sublista dentro de las sublistas, etc.
4. Eventualmente las sublistas tendrán solamente 1 elemento o ninguno, y estas sublistas estarán ordenadas por definición.

$\Rightarrow$ Algoritmo *recursivo*.

Quicksort



Verificando el algoritmo

- Ahora deberiamos averiguar que el algoritmo funciona en todos casos.
- Típicamente es difícil demsotrar conclusivamente que un algoritmo siempre funcionaría.
- Hay un método de demostración que se llama *el método de inducción* que frecuentemente se puede aplicar a algoritmos, en particular si procesan datos en trozos discretos.
- Para nosotros, como científicos usando computadores (y no informáticos) solamente tenemos que averiguar que el algoritmo funciona en las circunstancias relevantes a nuestro problema.

Eligiendo el mejor algoritmo

- ¿Cómo sabemos que el algoritmo que hemos diseñado es la "mejor" solución?
- ¿Qué significaría "mejor" en este caso?
- Típicamente significa el más preciso y el más rápido.
- Precisión es normalmente el primer aspecto de un algoritmo que verificamos.
- Calcular la rapidez de un algoritmo es un problema desafiante. ¿Cómo podemos comparar la rapidez de dos algoritmos?

Determinando la rapidez de un algoritmo

- Recuerde que un algoritmo es independiente de los medios utilizados para implementarlo.
- Este significa que el algoritmo es independiente del computador que usamos y el lenguaje de programación.
- Por lo tanto, tenemos que caracterizar la "rapidez" de un algoritmo en una manera que no refiere a la velocidad del procesador, el número de procesadores, si usamos Python o C, etc.

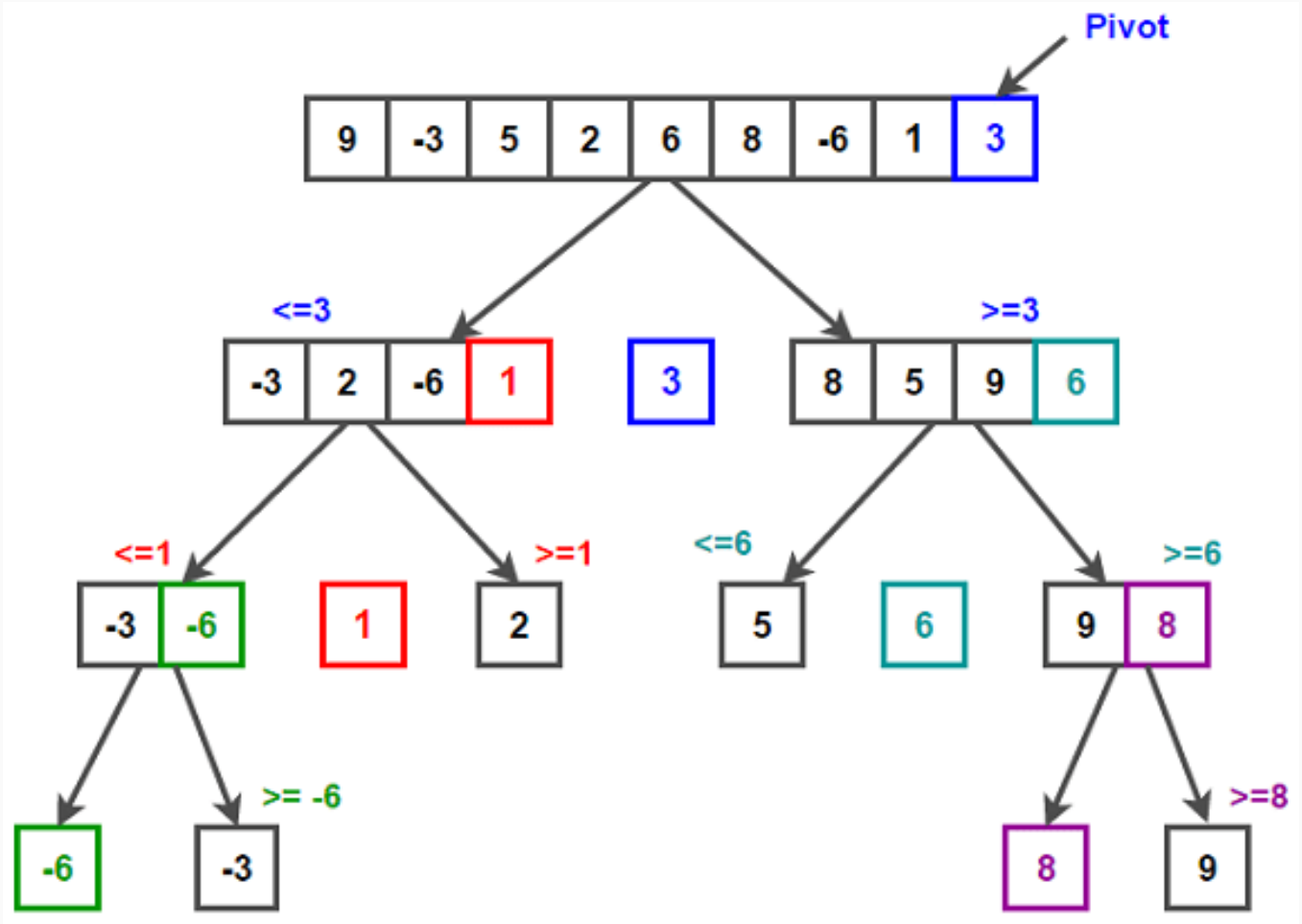
Complejidad algorítmica

- En la computación la rapidez de un algoritmo se define en terminos de su “complejidad”.
- Determinamos la complejidad de un algoritmo por la consideración del número de pasos que el algoritmo tiene que ejecutar para completar su tarea.

Complejidad algorítmica

- Pero, ¿qué es un "paso" para un algoritmo?
- La definición exacta de lo que constituye un "paso" depende del algoritmo.
- NO es algo al nivel del procesador y sus ciclos de operación.
- En el análisis de algoritmos, estamos pensando al nivel de los cálculos necesarios en el algoritmo.

Quicksort

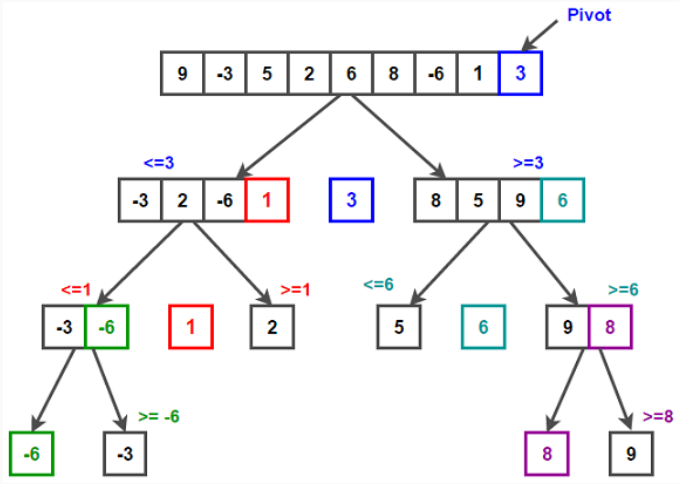


Complejidad algorítmica

- De hecho, estamos más interesados, cuando hablamos de complejidad, en como el algoritmo se *escala* cuando lo aplicamos a más datos.
- En el caso de "quicksort" queremos saber que tan rápido o lento es el algoritmo cuando lo aplicamos a una lista de 100 elementos, comparado a 10, o 1000000 elementos comparado a 1000, etc.

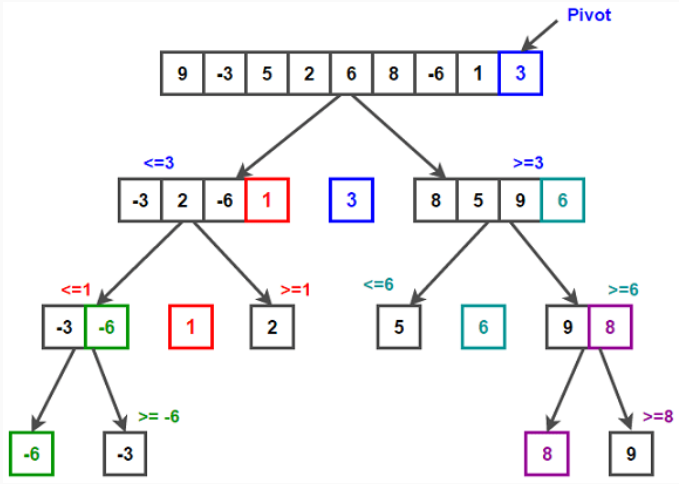
Complejidad algorítmica: N-cuerpos

- ¿Cómo describimos la complejidad de un algoritmo?



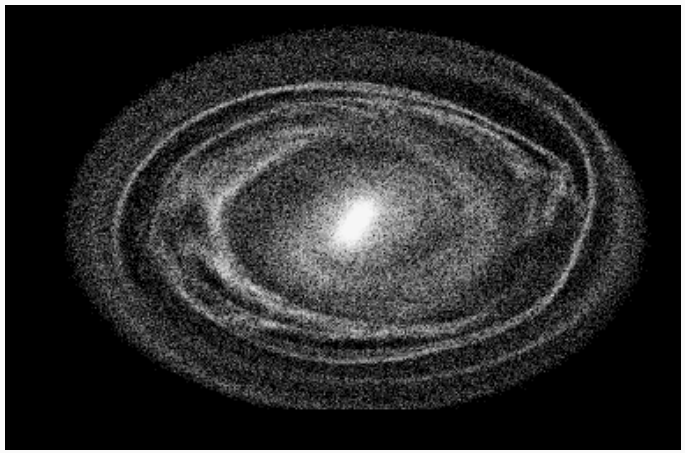
Complejidad algorítmica: N-cuerpos

- ¿Cómo describimos la complejidad de un algoritmo?
- Vamos a usar un algoritmo más simple que “quicksort”.



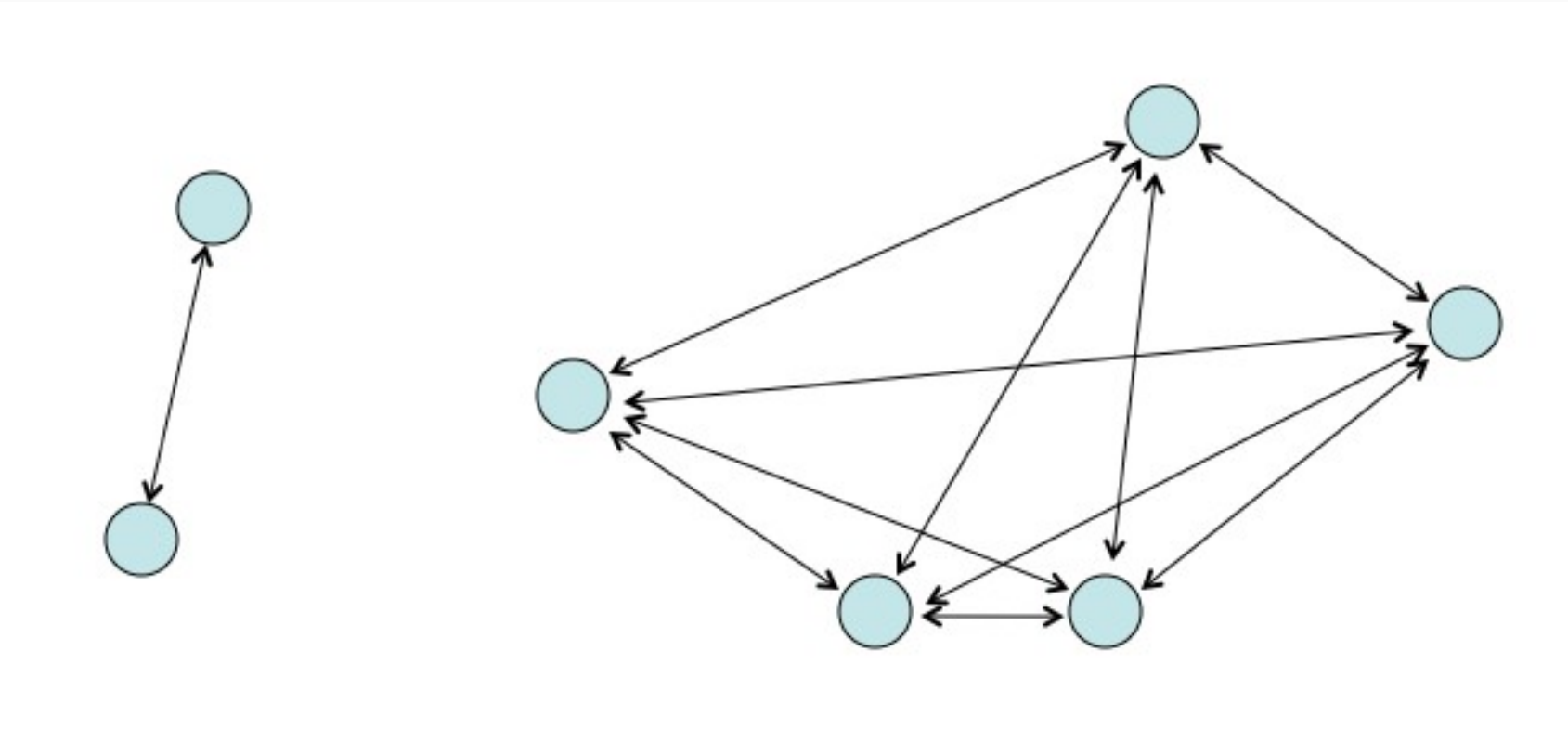
Complejidad algorítmica: N-cuerpos

- ¿Cómo describimos la complejidad de un algoritmo?
- Vamos a usar un algoritmo más simple que "quicksort".
- Analizaremos un tipo de simulación computacional que se usa mucho en la astrofísica: simulaciones de N-cuerpos.



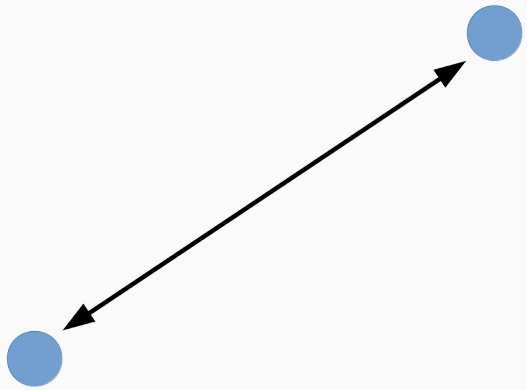
Complejidad algorítmica: N-cuerpos

- Fuerzas $\Rightarrow$ aceleraciones $\Rightarrow$ velocidades.
- Se puede actualizar las posiciones de las partículas con sus velocidades: el sistema se evoluciona en el tiempo.



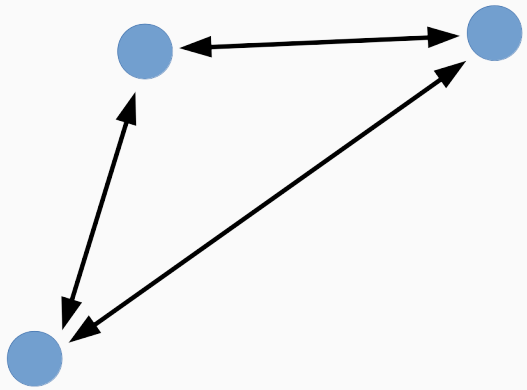
Complejidad algorítmica: N-cuerpos

- El "paso" básico en este algoritmo es el calculo de las fuerzas entre cada par de partículas.
- Con solo 2 partículas tenemos que calcular solamente la fuerza entre un par y listo! (1 paso).



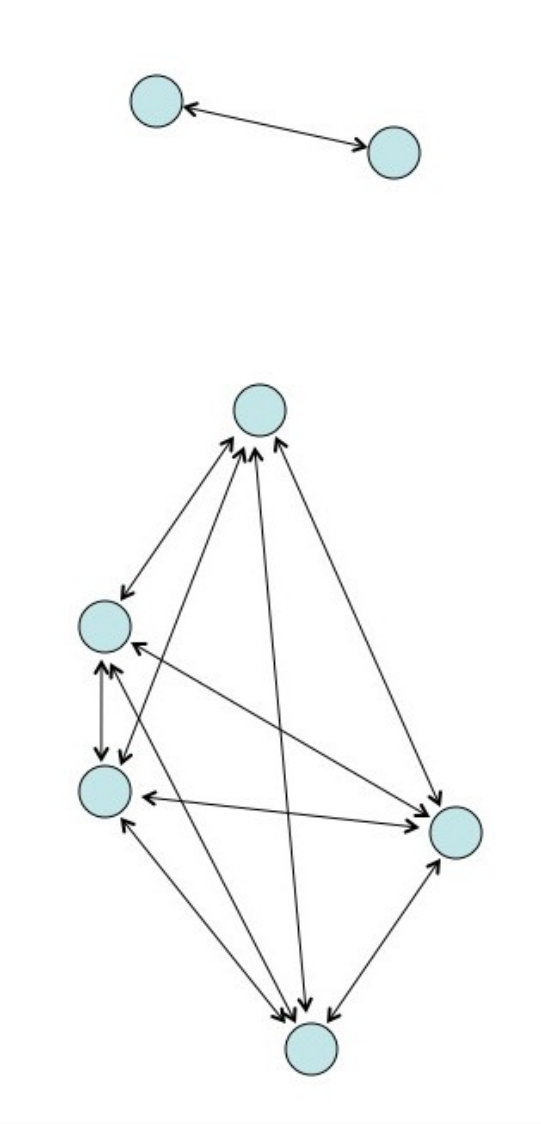
Complejidad algorítmica: N-cuerpos

- 3 partículas: 3 pares posibles $(1, 2)$, $(1, 3)$, $(2, 3)$. (3 'pasos' en el algorítmo)
- ¿n partículas?
- La pregunta es, ¿cuántos pares únicos podemos formar de n partículas?



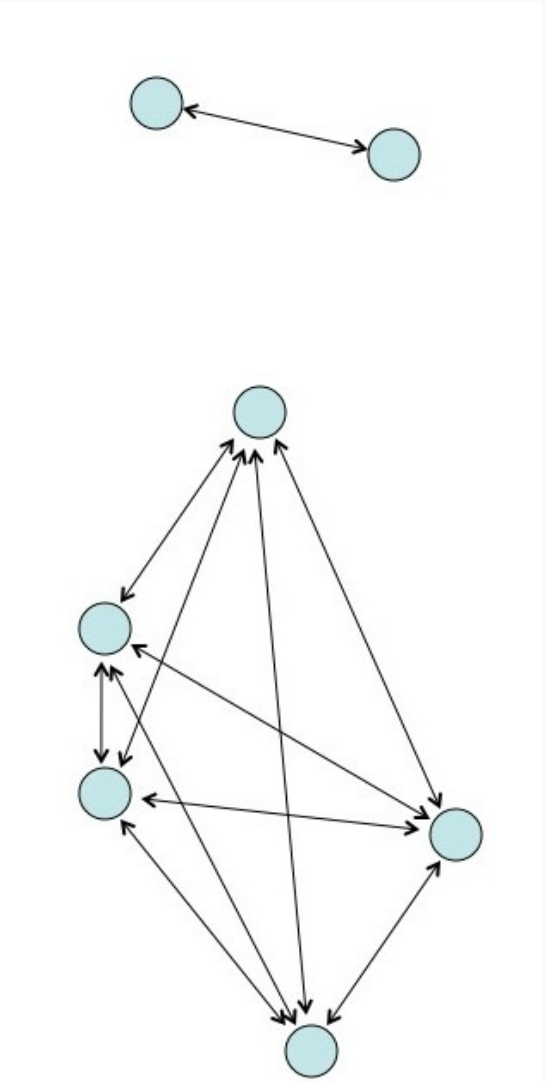
Complejidad algorítmica: N-cuerpos

- Hay n pares para cada partícula (incluyendo un "par" de la partícula con si misma).
- Hay n partículas $\Rightarrow$ número total de pares es $n \times n = n^2$.
- Para eliminar los pares falsos (de una partícula con si misma), de los cuales hay n, podemos restar n de n^2.



Complejidad algorítmica: N-cuerpos

- Pero, estamos contando los mismos pares dos veces: el par $(3, 5)$ es el mismo par como $(5, 3)$.
- Multiplicamos por $1/2$, para obtener el número final de pares únicos: $\frac{1}{2}(n^2 - n)$.

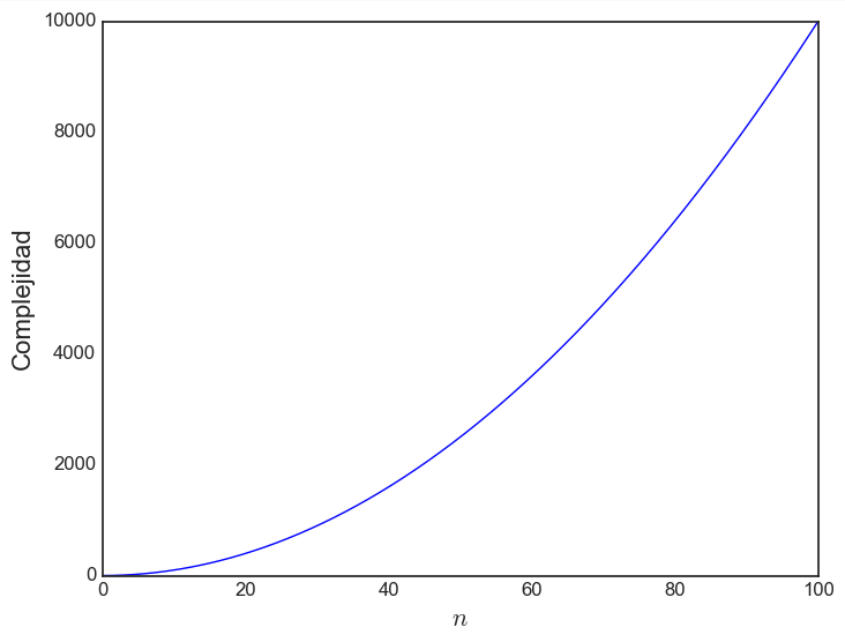


Complejidad algorítmica: notación O grande

- La complejidad del algoritmo está dado simplemente por contar el número de pares únicos!
- Expresamos la complejidad en notación "Big-O" (O grande): $\mathcal{O}$.

Complejidad algorítmica: notación O grande

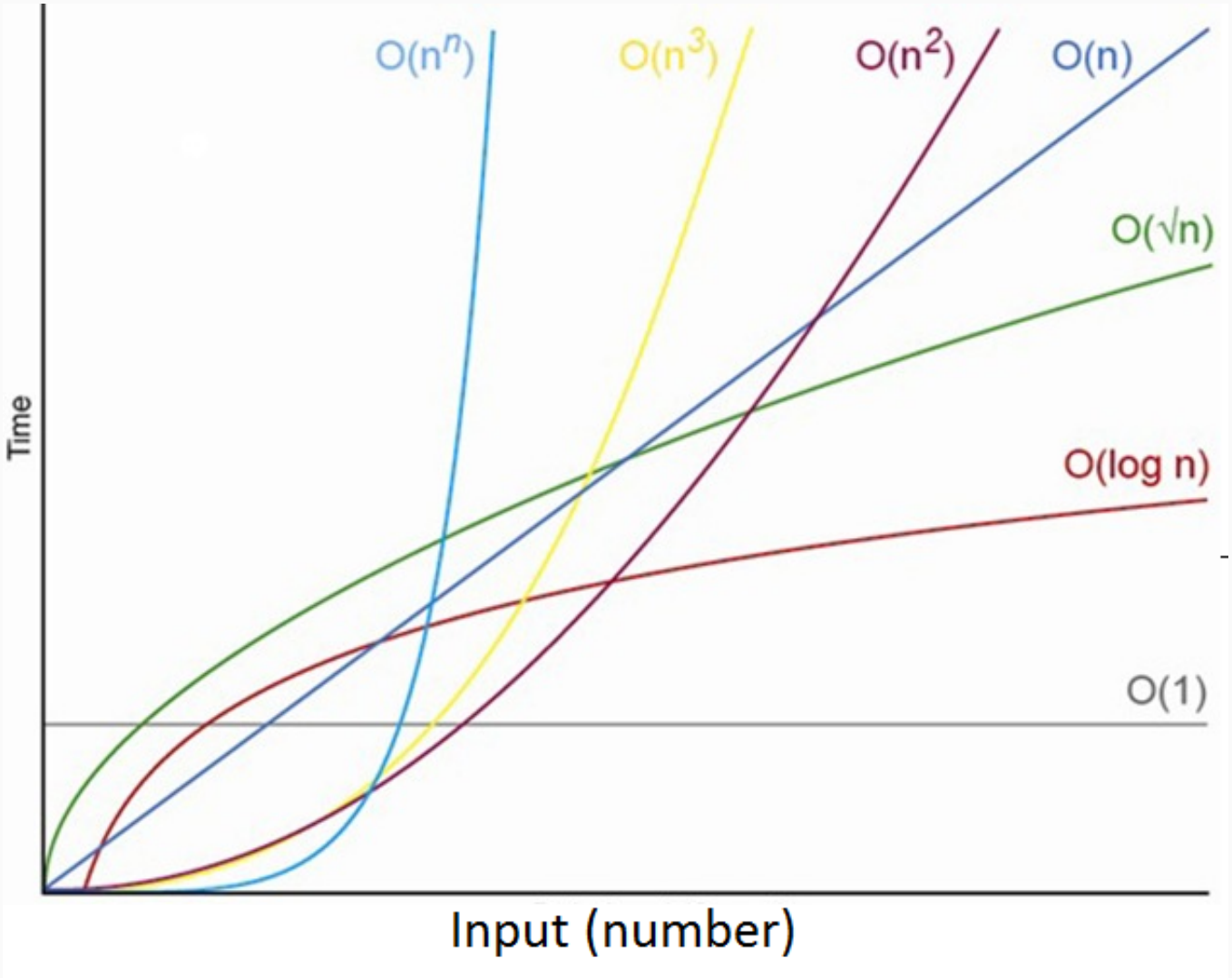
- Número de pares de n partículas es $\frac{1}{2}(n^2 - n)$. Usamos el término del grado más alto del polinomial (sin preocuparnos por los coeficientes): $\mathcal{O}(n^2)$.
- Este es porque este término crece más rápido con n mayor que los otros, y el factor de $1/2$ no es muy importante.



Complejidad algorítmica: notación O grande

- El algoritmo de N-cuerpos tiene una complejidad de $O(n^2)$.
- Si la implementación del algorítmo (i.e. el programa) demora 2 segundos para 10 partículas (i.e. $n = 10$),
  - $t = cn^2$, $2 = c(10^2) \Rightarrow c = 0,02$.
- El programa demorará aproximadamente 200 segundos para $n = 100$ partículas.
  - $t = cn^2$, $t = (0,02) \times (100^2) = 200$.
- Este algoritmo, por lo tanto, se *escala* muy mal: demora mucho más tiempo cuando aumentamos el número de partículas un poco.

Diferentes complejidades



¿Por qué es importante saber la complejidad de un algoritmo?

- Ejemplo: si uno tiene un *software* para simulaciones de N-cuerpos, pero sin información de su complejidad algoritmica.

¿Por qué es importante saber la complejidad de un algoritmo?

- Ejemplo: si uno tiene un *software* para simulaciones de N-cuerpos, pero sin información de su complejidad algoritmica.
- Quizás uno lo usa para simular un cúmulo de estrellas muy pequeña con 20 partículas, y tarda 1 hora para completarse.

¿Por qué es importante saber la complejidad de un algoritmo?

- Ejemplo: si uno tiene un *software* para simulaciones de N-cuerpos, pero sin información de su complejidad algoritmica.
- Quizás uno lo usa para simular un cúmulo de estrellas muy pequeña con 20 partículas, y tarda 1 hora para completarse.
- Si el algoritmo tiene complejidad $\mathcal{O}(n^2)$ y uno intenta simular una galaxia con 200.000 partículas, se demorará...

¿Por qué es importante saber la complejidad de un algoritmo?

- Ejemplo: si uno tiene un *software* para simulaciones de N-cuerpos, pero sin información de su complejidad algoritmica.
- Quizás uno lo usa para simular un cúmulo de estrellas muy pequeña con 20 partículas, y tarda 1 hora para completarse.
- Si el algoritmo tiene complejidad $\mathcal{O}(n^2)$ y uno intenta simular una galaxia con 200.000 partículas, se demorará...

  11.000 años!

¿Por qué es importante saber la complejidad de un algoritmo?

- Ejemplo: si uno tiene un *software* para simulaciones de N-cuerpos, pero sin información de su complejidad algoritmica.
- Quizás uno lo usa para simular un cúmulo de estrellas muy pequeña con 20 partículas, y tarda 1 hora para completarse.
- Si el algoritmo tiene complejidad $\mathcal{O}(n^2)$ y uno intenta simular una galaxia con 200.000 partículas, se demorará...

  11.000 años!
- En qué momento se deja de esperar..?

Contenido

Precisión en la computación

- Estamos todos acostumbrados a calcular cosas en una calculadora (o en el telefono) y confiar en el resultado. Pero, de hecho, los computadores nunca son perfectamente precisos.

  $\Rightarrow$ Tienen una memoria finíta

  $\Rightarrow$ Utilizan números binarios.

Memoria finita

- Los computadores no pueden almacenar numeros a una precisión arbitrariamente alta.
- Por ejemplo: el número π es un número irracional (de hecho, trascendente).

Memoria finita

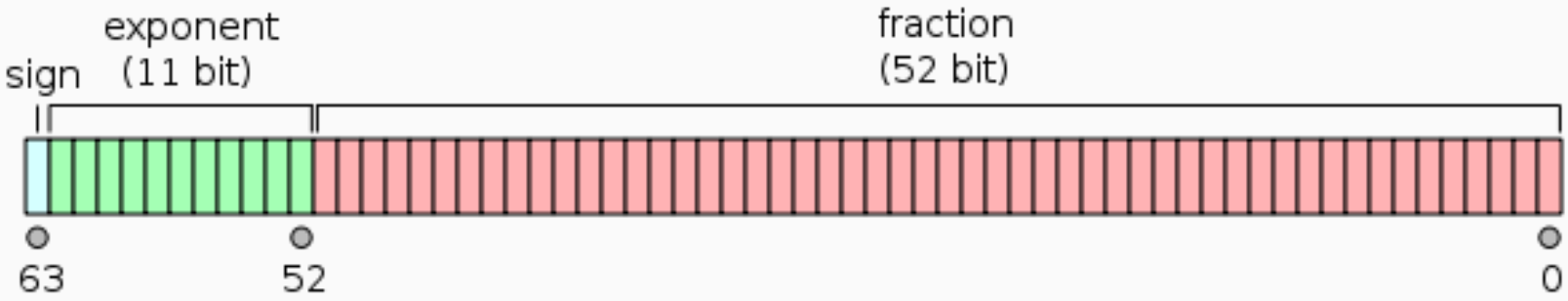
- No se puede escribir π en ningún sistema de números (base-10, base-20, lo que sea) a una precisión arbitraria porque requeriría un número infinito de dígitos.
- Por lo tanto, cualquier cálculo en un computador que involucra π tiene que ser (hasta cierto punto) inexacto porque el número está truncado.

Límites en tamaños de variables

- De hecho, la mayoría de los lenguajes de programación ponen límites en los tamaños de los números que se puede guardar en una variable.
- En Python un *float* normal no puede ser mayor que 10^{308}. ¿Por qué? Tiene que ver con como los computadores guardan números en binario.

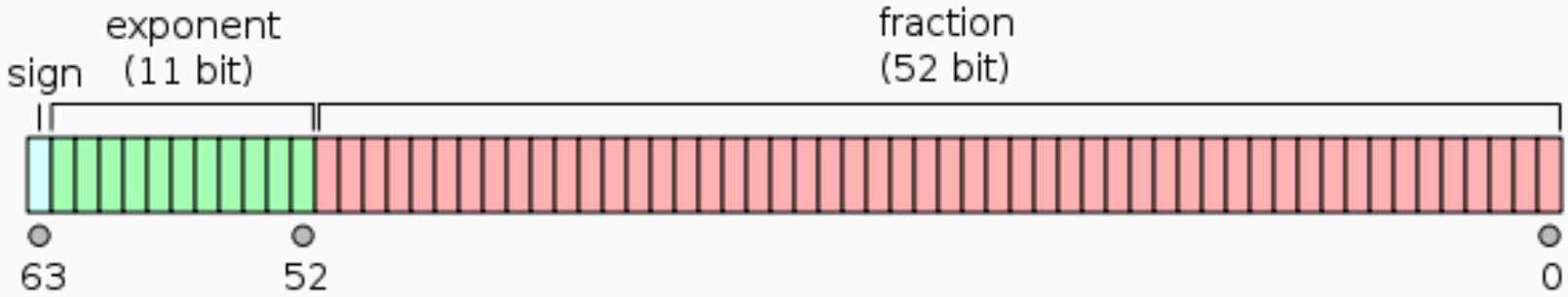
Estándar para números de punto flotante

- Hay un estandar en la informática, que se llama IEEE754, que define como guardar un número de punto flotante en la memoria de un computador.
- En Python hay un total de 64 bits para guardar un *float*. De estos, 1 bit es para el signo (+/-), 11 bits son para el exponente, y 52 bits son para el significando (mantisa, coeficiente).



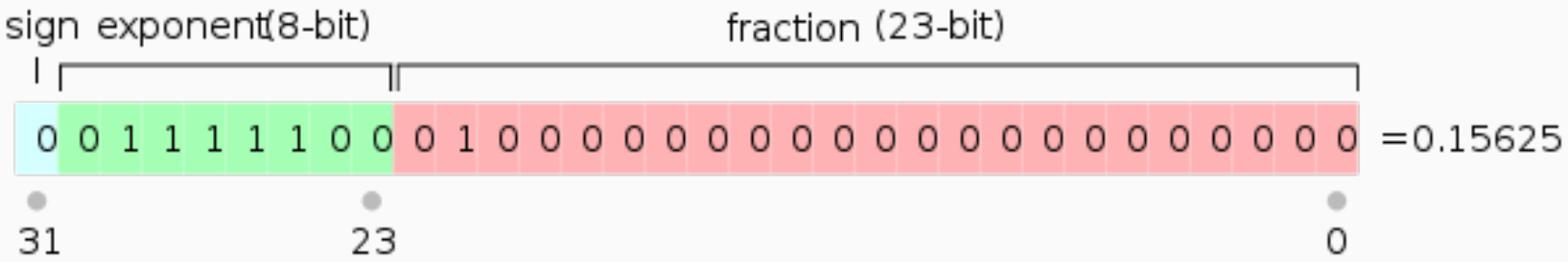
Estándar para números de punto flotante

- Con 11 bits en el exponente, el rango de números decimales que se puede guardar es de -1024 a 1023.
- 2^{1023} es aproximadamente 10^{308}.



Precisión de numeros de punto flotante

- Entonces, todos los números de punto flotante en un computador están truncados.
- En la mayoría de los lenguajes hay *floats* de precisión-simple (*single-precision*) y precisión-doble (*double-precision*).
- Precisión-simple: 32 bits. Precisión-doble: 64 bits.
- Es importante saber cual se ocupa, porque determina la precisión numérica del algoritmo.



Precisión de la máquina

- El grado de precisión que uno puede obtener en una variable se llama la precisión de la máquina (*machine precision*), ϵ.
- No importa el programa que escribimos, ni el algoritmo que usamos, NUNCA calcularemos valores a una precisión mayor que la precisión de la máquina.
- Notese que algunos lenguajes permiten control en la cantidad de memoria asignada a variables de distintos tipos, y por eso podemos obtener valores muy pequeños para ϵ (es decir, precisión extremadamente alta).

Precisión doble: $\epsilon = 2^{-53} \approx 1,11 \times 10^{-16}$.

El problema de precisión finita

- Ejemplo: multiplicamos un número muy pequeño por un número muy grande, usando precisión-simple: el *significando* del número tendrá 23 bits.
- Estos 23 bits corresponden a $7 - 8$ dígitos decimales (depende del número).
- El septimo u octavo dígito en la representación decimal del número será incorrecto (por el truncamiento del número).

El problema de precisión finita

- La constante gravitacional en unidades SI: $6,67408 \times 10^{-11}$ $\mathrm{m}^3\ \mathrm{kg}^{-1}\ s^{-2}$.
- La masa solar en unidades SI: 2×10^{30} kg.
- El septimo dígito del resultado de la multiplicación será un número de magnitud 10^{13}!
- Este número podría ser suficientemente grande para modificar otros números en el cálculo y introducir errores.

Números que no se puede representar exactamente en binario

- Otro problema: hay algunos números que no podemos representar exactamente en el sistema binario.
- En el sistema decimal, no se puede representar $1/3$ exactamente (es igual a $0.3333333\ldots$).

Números que no se puede representar exactamente en binario

- Por las mismas razones, hay fracciones que no se puede representar exactamente en binario.

Números que no se puede representar exactamente en binario

- Por las mismas razones, hay fracciones que no se puede representar exactamente en binario.
- Un ejemplo común es el número 0.1 (una representación *exacta* de 1/10 en el sistema decimal).

Números que no se puede representar exactamente en binario

- Por las mismas razones, hay fracciones que no se puede representar exactamente en binario.
- Un ejemplo común es el número 0.1 (una representación *exacta* de 1/10 en el sistema decimal).
- En binario, este tiene la representación $0.0001100110011001100110011\ldots$. Por lo tanto, NUNCA tendremos el valor 0.1 representado EXACTAMENTE en el computador - siempre habrá un error del tamaño de la precisión de la máquina.

Resumen

- Los algoritmos son las longanizas del choripan que es la computación científica.

Resumen

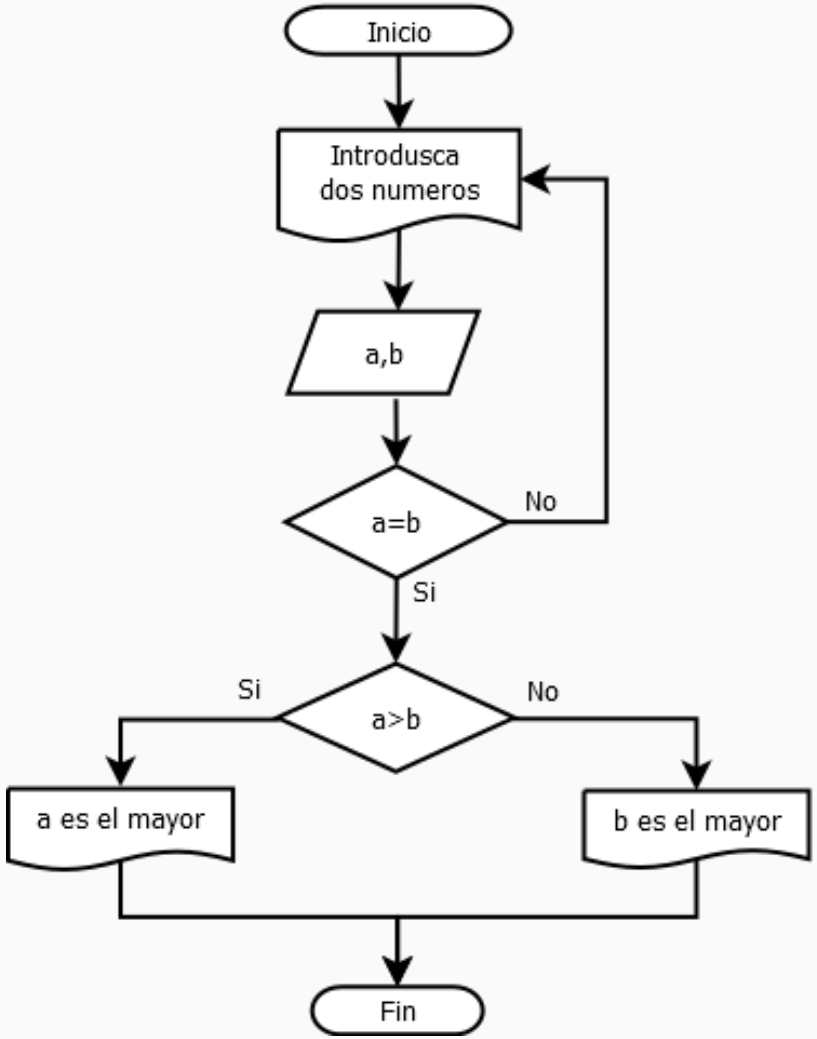
- Los algoritmos son las longanizas del choripan que es la computación científica.
- Algoritmos son procesos estructurados diseñados para resolver un problema especifico.

Resumen

- Los algoritmos son las longanizas del choripan que es la computación científica.
- Algoritmos son procesos estructurados diseñados para resolver un problema especifico.
- El primer paso, en cualquier tarea computacional, es pensar sobre el algoritmo, diseñarlo y planificarlo.

Resumen

- Planificar:
  - Lista de instrucciones
  - Diagrama de flujo
  - Pseudo-código



Resumen

- Planificar:
  - Lista de instrucciones
  - Diagrama de flujo
  - Pseudo-código

```
1   Proceso NUMERO_MAYOR
2       Escribir 'Ingresa A: ';
3       Leer A;
4       Escribir 'Ingresa B: ';
5       Leer B;
6       Si A > B Entonces
7           Escribir 'El mayor es A';
8       Sino
9           Escribir 'El mayor es B';
10      FinSi
11  FinProceso
12
```

Resumen

- La implementación de un algorítmo es un programa.
- Un algoritmo bueno es *rápido* y *preciso*.
- La complejidad algorítmica: como se escala su tiempo de ejecución con la cantidad de datos que usamos como entrada al algoritmo.

$\Rightarrow$ Notación O-grande (Big-O), $\mathcal{O}$.

Resumen

- La exactitúd de los algoritmos está afectado por la precisión de las variables.
- Casi todos los números que están guardados en el computador tienen un cierto nivel de imprecisión.
- Siempre debemos estar consciente de eso cuando escribimos programas (aunque en la práctica es raro que es un gran problema...)