

Software crítico para sistemas espaciales

2026/27

Master Universitario en Enxeñaría Aeroespacial

Arno Formella
Miguel Ramón González Castro
Manuel Pérez Cota

Departamento de Informática
Universidade de Vigo

26/27

- secuencia finita (*receta*) de pasos (o instrucciones) bien definidos para lograr una tarea (con bucles o recursión)
- los pasos, en principio, deben ser ejecutables por un humano
- las instrucciones deben realizar una transformación finita del estado (o configuración) del sistema (alteraciones finitas de bits de la configuración del sistema)
- lograr una tarea significa que se puede verificar una propiedad del estado (final) del sistema (posiblemente parcialmente) mediante un algoritmo
- un programa (software) es un algoritmo formulado en un lenguaje formal de tal forma que (después de posibles transformaciones automáticas) se puede ejecutar en un ordenador (hardware), el estado se encuentra en la memoria

complejidad

- algoritmos se caracterizan por su complejidad tanto en **tiempo** (medido en instrucciones o pasos ejecutados) como en **memoria** (tamaño de las configuraciones intermedias en memoria)
- para una cota asintótica superior respecto a una entrada de tamaño n , decimos: la función f es de orden de una función g , siempre cuando:

$$\exists c > 0 \exists n_0 \forall n > n_0 : |f(n)| \leq c \cdot g(n)$$

escribimos: $f(n) = \mathcal{O}(g(n))$.

- ejemplo: si $f(n) = \mathcal{O}(n^3)$ entonces f no crece más rápido que cúbico
- hay más notaciones para otras caracterizaciones asintóticas: o , ω , Ω , Θ .
- si queréis, mirad el **complexity zoo**:

https://complexityzoo.net/Complexity_Zoo



- existen *problemas* que no son computables
o en otras palabras: hay cosas que no tienen solución
- ejemplos: verificar equivalencia entre gramáticas formales,
averiguar si un programa ejecuta todas sus líneas
- hay problemas de los cuales se sabe que son computables
(es decir, existe solución) en la teoría (eficiente), pero **no se sabe como calcularla** en la práctica (eficiente); mirad en la teoría de grafos los *menores prohibidos*
- hay problemas para los cuales hay algoritmos que los resuelven, pero aún no se sabe la *clase más pequeña a la cual pertenecen*, p.ej., deshacer un no-nudo; mirad la *teoría de los nudos*

<https://en.wikipedia.org/wiki/Unknot> o la factorización de números https://en.wikipedia.org/wiki/Integer_factorization

ejemplos de tiempo de cálculo según complejidad

asumimos que podemos *tratar* con 1 millón de cosas en un **segundo** usando un algoritmo **lineal** (es decir, procesar con un megahercio):

función	$\mathcal{O}$ -notación	tiempo
tiempo lineal	$\mathcal{O}(n)$	1 segundo
tiempo casi-lineal	$\mathcal{O}(n \log n)$	20 segundos
tiempo cuadrático	$\mathcal{O}(n^2)$	11.6 días
tiempo cúbico	$\mathcal{O}(n^3)$	31710 años
tiempo cuártico	$\mathcal{O}(n^4)$	32 mil millones de años (2.3 veces edad del universo)
tiempo exponencial	$\mathcal{O}(2^n)$	eterno
tiempo factorial	$\mathcal{O}(n!)$	ya no hay más palabras

ejemplos de tiempo de cálculo según complejidad

asumimos que podemos tratar con 1 millón de cosas en un **milisegundo** usando un algoritmo **lineal** (es decir, procesar con un gigahercio):

función $\mathcal{O}$ -notación	tiempo	
tiempo lineal	$\mathcal{O}(n)$	1 milisegundo
tiempo casi-lineal	$\mathcal{O}(n \log n)$	20 milisegundos
tiempo cuadrático	$\mathcal{O}(n^2)$	16 minutos
tiempo cúbico	$\mathcal{O}(n^3)$	31.7 años
tiempo cuártico	$\mathcal{O}(n^4)$	32 millones de años
tiempo exponencial	$\mathcal{O}(2^n)$	eternal
tiempo factorial	$\mathcal{O}(n!)$	ya no hay más palabras

Observa: una **paralelización** con una máquina de p procesadores nos da como mucho una aceleración lineal en p (y en la mayoría de los casos ni eso); **Computación cuántica** ofrece a veces (algoritmo de Grover) una aceleración cuadrática respecto a n

tamaños de problemas tratables respecto a complejidad

asumimos **1 nanosegundo** de tiempo para procesar **una** cosa (es decir, frecuencia de operación de 1 gigahercio).

tamaños de problemas tratables dentro de una hora:

función	$\mathcal{O}$ -notación	tamaño
tiempo lineal	$\mathcal{O}(n)$	3.6 billones
tiempo casi-lineal	$\mathcal{O}(n \log n)$	96.6 mil millones
tiempo cuadrático	$\mathcal{O}(n^2)$	1.9 millones
tiempo cúbico	$\mathcal{O}(n^3)$	15.3 miles
tiempo cuártico	$\mathcal{O}(n^4)$	1377
tiempo exponencial	$\mathcal{O}(2^n)$	41
tiempo factorial	$\mathcal{O}(n!)$	15

entonces:

- para el problema dado o aplicación por desarrollar
- se debe averiguar
 - si existe un algoritmo para computarlo
 - cual es el mejor/adecuado paradigma de implementación
 - cual es la mejor/adecuada implementación para los datos esperados en los casos de su aplicación real
- se debe pensar sobre un algoritmo de certificación, es decir, otro algoritmo que compruebe—emitiendo un certificado—que el algoritmo original ha calculado correctamente
- se debe validar y verificar que el algoritmo cumple con todos los requisitos y todas las especificaciones

representación de enteros

- recuerda: el invento del cero junto con anotar números con cifras en posiciones fue un gran logro (mira historia del 0)
- basándose en dígitos (p.ej., de 0 a 9) se escribe un valor teniendo en cuenta la posición del dígito (nosotros desde la derecha a la izquierda, como la escritura árabe)
- asumiendo una base b (p.ej., 2 o 10 o 16) se calcula el valor v de una secuencia de dígitos $d_n d_{n-1} \dots d_2 d_1 d_0$ (con $d_i \in [0, b)$):

$$v = (d_n d_{n-1} \dots d_2 d_1 d_0)_b = \sum_{i=0}^n d_i \cdot b^i$$

representación de enteros negativos

- con signo, es decir, se añade un dígito como prefijo cuando es cero, el número es positivo, sino negativo:

$$v = (d_n d_{n-1} \dots d_2 d_1 d_0)_2 = \begin{cases} + \sum_{i=0}^{n-1} 2^i \cdot b^i & \text{si } d_n = 0 \\ - \sum_{i=0}^{n-1} 2^i \cdot b^i & \text{si } d_n = 1 \end{cases}$$

- complemento de dos:

$$v = (d_n d_{n-1} \dots d_2 d_1 d_0)_2 = \begin{cases} -2^n + \sum_{i=0}^{n-1} 2^i \cdot b^i & \text{si } d_n = 1 \\ \sum_{i=0}^n d_i \cdot b^i & \text{si } d_n = 0 \end{cases}$$
$$= (d_n d_{n-1} \dots d_2 d_1 d_0)_2$$

error de desbordamiento

intervalos de números enteros en los sistemas de uso actual

8 bits	unsigned char	0
		255
	char	-128
		127
16 bits	unsigned short	0
		65 535
	short int	-32 768
		32 767
32 bits	unsigned long int	0
		4 294 967 295
	long int	-2 147 483 648
		2 147 483 647
64 bits	unsigned long long int	0
		18 446 744 073 709 551 615
	long long int	-9 223 372 036 854 775 808
		9 223 372 036 854 775 807

int depende del sistema, a veces short, a veces long



calcular factorial con `int` (32 bits)

```
1  #include <iostream>
2
3  void fact(int n) {
4      int p=1;
5      for(int i=1; i<=n; ++i)
6          p *= i;
7      std::cout << p;
8  }
9
10 int main() {
11     for(int i=1; i<=35; ++i) {
12         std::cout << i << "! = ";
13         fact(i);
14         std::cout << std::endl;
15     }
16     return 0;
17 }
18
```

calcular factorial con `int` (32 bits)

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrire/cpp $ g++ -O3 fac_int.cpp
formella@hpformella ~/tex/teach/scrire/cpp $ ./a.out
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
14! = 1278945280
15! = 2004310016
16! = 2004189184
17! = -288522240
18! = -898433024
19! = 109641728
20! = -2102132736
21! = -1195114496
22! = -522715136
23! = 862453760
24! = -775946240
25! = 2076180480
26! = -1853882368
27! = 1484783616
28! = -1375731712
29! = -1241513984
30! = 1409286144
31! = 738197504
32! = -2147483648
33! = -2147483648
34! = 0
35! = 0
formella@hpformella ~/tex/teach/scrire/cpp $
```

calcular factorial con long (64 bits)

```
1  #include <iostream>
2
3  void fact(int n) {
4      long p=1;
5      for(int i=1; i<=n; ++i)
6          p *= i;
7      std::cout << p;
8  }
9
10 int main() {
11     for(int i=1; i<=35; ++i) {
12         std::cout << i << "! = ";
13         fact(i);
14         std::cout << std::endl;
15     }
16     return 0;
17 }
18
```

calcular factorial con long (64 bits)

File Edit View Search Terminal Help

```
formella@hpformella ~/tex/teach/scribe/cpp $ g++ -O3 fac_long.cpp
```

```
formella@hpformella ~/tex/teach/scribe/cpp $ ./a.out
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = -4249290049419214848
22! = -1250660718674968576
23! = 8128291617894825984
24! = -7835185981329244160
25! = 7034535277573963776
26! = -1569523520172457984
27! = -5483646897237262336
28! = -5968160532966932480
29! = -7055958792655077376
30! = -8764578968847253504
31! = 4999213071378415616
32! = -6045878379276664832
33! = 3400198294675128320
34! = 4926277576697053184
35! = 6399018521010896896
```

```
formella@hpformella ~/tex/teach/scribe/cpp $ █
```

calcular factorial con python (tamaño arbitrario)

```
1  def fact(n):
2      p=1
3      for i in range(1,n+1):
4          p*=i
5      return p
6
7  for i in range(1,31):
8      print("{}!".format(i), "= {}".format(fact(i)))
9
10 fact(1000000)
```

calcular factorial con python (tamaño arbitrario)

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrise/python $
formella@hpformella ~/tex/teach/scrise/python $ python3 fac_simple.py
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 112400072777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 1033314796638614492966651337523200000000
formella@hpformella ~/tex/teach/scrise/python $
```

calcular factorial con gmp (tamaño arbitrario)

```
1  #include <iostream>
2  #include <gmp.h>
3
4  void fact(int n, bool print=true) {
5      mpz_t p;
6      mpz_init_set_ui(p,1);  // p = 1
7      for(int i=1; i<=n; ++i)
8          mpz_mul_ui(p,p,i);  // p = p * i
9      if(print) mpz_out_str(stdout,10,p);
10     mpz_clear(p);
11 }
12
13 int main() {
14     for(int i=1; i<=30; ++i) {
15         std::cout << i << "! = ";
16         fact(i);
17         std::cout << std::endl;
18     }
19     fact(1000000,false);
20     return 0;
21 }
```



calcular factorial con gmp (tamaño arbitrario)

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrise/cpp $ g++ -O3 fac_simple.cpp -lgmp
formella@hpformella ~/tex/teach/scrise/cpp $ ./a.out
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 112400072777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 1033314796638614492966651337523200000000
formella@hpformella ~/tex/teach/scrise/cpp $
```

factorial de biblioteca con python (tamaño arbitrario)

```
1  import math
2
3  for i in range (1,31):
4      print(
5          "{}!".format(i),
6          "= {}".format(math.factorial(i))
7      )
8
9  math.factorial(1000000)
```

factorial de biblioteca con python (tamaño arbitrario)

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrise/python $
formella@hpformella ~/tex/teach/scrise/python $
formella@hpformella ~/tex/teach/scrise/python $ time python3 fac_fac.py
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000

real    0m6.619s
user    0m6.609s
sys      0m0.009s
formella@hpformella ~/tex/teach/scrise/python $
```

factorial de biblioteca con gmp (tamaño arbitrario)

```
1  #include <iostream>
2  #include <gmp.h>
3
4  void fact(int n, bool print=true) {
5      mpz_t p;
6      mpz_init_set_ui(p,1);    // p = 1
7      mpz_fac_ui(p,n);         // p = n!
8      if(print) mpz_out_str(stdout,10,p);
9      mpz_clear(p);
10 }
11
12 int main() {
13     for(int i=1; i<=30; ++i) {
14         std::cout << i << "! = ";
15         fact(i);
16         std::cout << std::endl;
17     }
18     fact(1000000,false);
19     return 0;
20 }
21
```



factorial de biblioteca con gmp (tamaño arbitrario)

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrise/cpp $
formella@hpformella ~/tex/teach/scrise/cpp $ g++ -O3 fac fac.cpp -lgmp
formella@hpformella ~/tex/teach/scrise/cpp $ time ./a.out
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000

real    0m2.659s
user    0m2.623s
sys      0m0.036s
formella@hpformella ~/tex/teach/scrise/cpp $
```

diferencias en tiempo de ejecución

python	simple	331.12 seconds
	biblioteca	6.62 seconds
C++	simple	94.60 seconds
	biblioteca	0.16 seconds

- simple: python es 2.5 veces más lento que C++
- biblioteca: python es 40 veces más lento que C++
- en python la versión propia simple es 50 veces más lenta que la versión en biblioteca
- en C++ la versión propia simple es 590 veces más lenta que la versión en biblioteca
- **amenaza:** python naïf es 2000 veces más lento que C++ bien usado

- **solución:** controlar siempre el intervalo en el cual se realizan los cálculos considerando también todos los resultados intermedios o usar formatos de rango arbitrariamente grande
(¡cuales necesitan un análisis de la memoria requerida!)