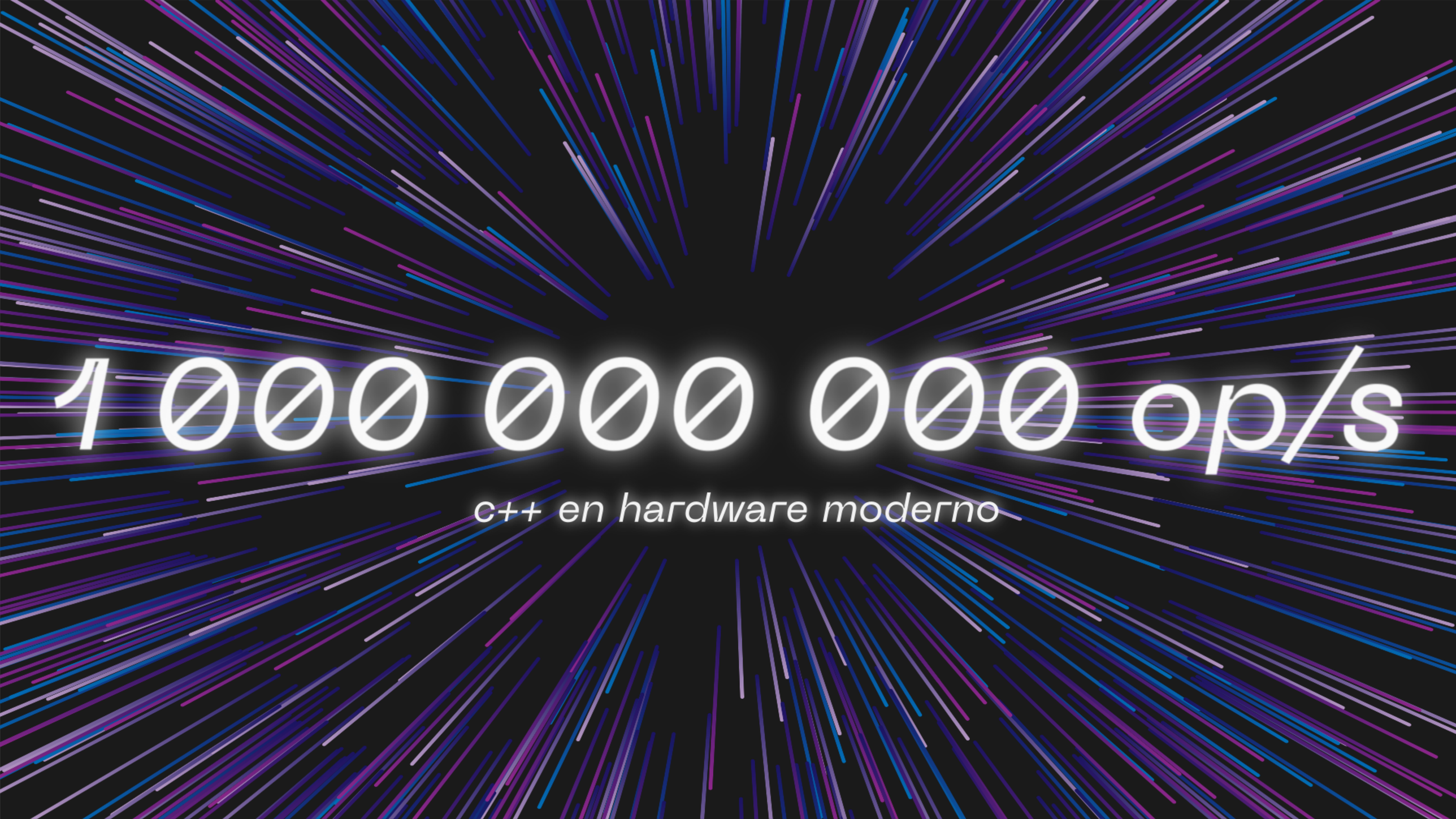
análisis de complejidad


1 000 000 000 op/s
c++ en hardware moderno

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

F. Forward and Backward

time limit per test: 0.5 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

A distant planetary system has a single sun and N–1 planets. Each planet is identified by a distinct integer from 2 to N. In planet b, numbers are represented using base b

A palindromic number is bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla

input
33
output
2 10 32

Título

Restricciones

Enunciado (con descripción de e/s y límites)

Ejemplos (opcional)

CLUB DE PROGRAMACIóN COMPETITIVA UADY

Aceptado

Wrong answer: Respuesta incorrecta

Time limit exceeded: Tiempo límite excedido

Memory limit exceeded: Memoria límite excedida

Presentation error: Produce salida en formato incorrecto (Tokens extra como espacios o saltos de linea)

Runtime error: Se ejecuta con errores (division entre cero, segfault)

Compilation error: El envío no compila

AC
WA
TLE
MLE
PE
RE
CE

CLUB DE PROGRAMACIóN COMPETITIVA
UADY

	O(1)	O(logn)	O(n)	O(nlogn)	$O(n^2)$	$O(2^n)$	O(n!)
10	<10ms	<10ms	<10ms	<10ms	<10ms	<10ms	<10ms
10^2	<10ms	<10ms	<10ms	<10ms	<10ms	10^{13} y	10^{141}y
10^3	<10ms	<10ms	<10ms	<10ms	<10ms	10^{284} y	10^{2551}y
10^4	<10ms	<10ms	<10ms	<10ms	10 ms	10^{2993}y	10^{35642}y
10^5	<10ms	<10ms	<10ms	<10ms	10 s	10^{30085}y	10^{10^6}y
10^6	<10ms	<10ms	10 ms	10 ms	16 m	10^{10^6}y	?
10^8	<10ms	<10ms	100 ms	2 s	115 d	10^{10^7}y	?
10^9	<10ms	<10ms	1 s	29 s	31 y	10^{10^8}y	?
10^{12}	<10ms	<10ms	16 m	11 h	10^5 y	?	?
10^{15}	<10ms	<10ms	11 d	576 d	10^{13} y	?	?
10^{18}	<10ms	<10ms	31 yr	1894 yr	10^{21} y	?	?

Incluso con la cantidad de operaciones por segundo que podemos realizar, el tamaño de la entrada y la complejidad de las técnicas que podríamos pretender emplear hace casi siempre necesario el uso de técnicas y algoritmos más sofisticados que atiendan a estas limitaciones.

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

Notación Big O

Expresa complejidad temporal de un algoritmo en el peor de los casos en función de n, para un n arbitrariamente grande, nos permite calcular de antemano si una solución cumplirá con las restricciones del problema.

La complejidad es un límite superior para el número de pasos que requiere un algoritmo como función del tamaño de entrada

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

Definición formal

$$f(x) = O(g(x)) \Leftrightarrow \exists C, n_0 \in \mathbb{N} | n \geq n_0 \Rightarrow f(n) \leq Cg(n)$$

Una función f(x) es O(g(x)) si existen C y n_0 tal que f(n) siempre es menor o igual a C · g(x) si n es mayor o igual a n_0

En la práctica es posible (y más rápido) calcular complejidad con métodos empíricos sin necesidad de emplear la definición formal.

CLUB DE PROGRAMACIóN COMPETITIVA UADY

Calculando complejidad

```
1  int a = 5;
2  int b = 7;
3  int c = 4;
4  int d = a + b + c + 153;
```

Las operaciones que se ejecutan en un tiempo que no depende de n son de complejidad constante u O(1). Algunas operaciones con esta complejidad son las operaciones aritméticas, asignaciones, comparaciones, acceso a contenedores secuenciales como std::vector o std::array, y acceso a contenedores basados en tablas de hash como std::unordered_map o std::unordered_set

```java
for (int i = 1; i <= n; i++) {
    // constant time code here
}
```

```java
int i = 0;
while (i < n) {
    // constant time code here
    i++;
}
```

```java
for (int i = 1; i <= 5 * n + 17; i++) {
    // constant time code here
}
```

```java
for (int i = 1; i <= n + 457737; i++) {
    // constant time code here
}
```

Las operaciones que dependen linealmente de n tienen complejidad lineal u O(n). El ejemplo más básico es iterar desde una constante hasta n

Cuando usamos Big O ignoramos constantes y términos de orden menor.

$$O(n) * O(1) + O(1) = O(n)$$

```java
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        // constant time code here
    }
}
```

Podemos encontrar la complejidad de ejecutar ciclos anidados multiplicando la complejidad de cada ciclo.

$$O(n) * O(m) = O(nm)$$

```java
for (int i = 1; i <= n; i++) {
    for (int j = i; j <= n; j++) {
        // constant time code here
    }
}
```

$$O(n) * O(n) = O(n^2)$$

CLUB DE PROGRAMACIóN COMPETITIVA UADY

```java
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        // constant time code here
    }
}
for (int i = 1; i <= n + 58834; i++) {
    // more constant time code here
}
```

De nuevo, ignoramos constantes y términos de orden menor, esto debido a que cuando n se hace suficientemente grande, el termino que domina el comportamiento de la función es el de grado mayor

$$O(n) * O(n) + O(n) + O(58834) = O(n^2 + n + 58834) = O(n^2)$$

Tomemos como ejemplo la siguiente expresión:

$$n^3 + 10n^2 + 100$$

para n = 1000 tenemos:

$$1000000000 + 10000000 + 100 = 1010000100$$

Podemos ver que el primer término contribuyó en mas del 99% al resultado final, por esto usualmente no tiene sentido considerar terminos de orden menor

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

Complejidades comunes

Expresiones algebráicas: O(1)

Busqueda binaria: O(log n)

Insercion y búsqueda en std::map Y std::set: O(logn)

Test de primalidad: O($\sqrt{n}$)

Busqueda lineal: O(n)

std::sort : O(n log n)

Iterar sobre todos los subsets de un arreglo: O(2^n)

Iterar sobre todas las permutaciones de un arreglo: O(n!)

CLUB DE PROGRAMACIÓN COMPETITIVA UADY

Tamaños de entrada y complejidades máximas

Considerando $5x10^8$ ops/s

$n \leq 11$: $O(n!)$

$n \leq 28$: $O(2^n)$

$n \leq 120$: $O(n^4)$

$n \leq 700$: $O(n^3)$

$n \leq 2x10^4$: $O(n^2)$

$n \leq 2x10^7$: $O(n \log n)$

$n \leq 5x10^8$: $O(n)$

$n \leq 2x10^{17}$: $O(\sqrt{n})$

$n \leq 2^{5x10^8}$: $O(\log n)$

$n \leq \infty$: $O(1)$

```java
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        // constant time code here
    }
}
for (int i = 1; i <= m; i++) {
    // more constant time code here
}
```

Algúnas veces usamos más de una variable para calcular la complejidad, como en este caso.

$$O(n) * O(n) + O(m) = O(n^2 + m)$$

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

Factor constante

En ciertos casos poco comunes puede que una solución que teóricamente está dentro de los límites de complejidad mostrados no sea aceptada.
Esto puede deberse a que los límites son muy "apretados" y los factores constantes que ignoramos terminan haciendo una pequeña diferencia.
Esto suele ser arreglado con optimizaciones que no modifican la complejidad temporal de la solución