Costo asintótico

Sebastian Mestre

¿Por qué nos importa el costo?

- En programación competitiva nos exigen que un programa termine en **menos de cierto tiempo** (típicamente 1 segundo).
- Para saber si nuestro programa va a entrar en tiempo, necesitamos estimar **cuánto tarda** antes de implementarlo completo.
- Implementar una idea y recién después descubrir que es demasiado lenta significa **perder mucho tiempo** en concurso.
- Queremos poder razonar sobre el **tiempo de ejecución** solo pensando en el algoritmo.

Tiempo exacto vs aproximación

- La cantidad exacta de tiempo que tarda un programa es imposible de conocer de antemano:
  - Depende de la máquina, compilador, caché, sistema operativo, qué otro software está corriendo, etc.
- Datos típicos:
  - Límite de tiempo ≈ 1 segundo.
  - Una CPU moderna ejecuta del orden de $3 \cdot 10^9$ operaciones por segundo.
- **Idea:** alcanza con garantizar que nuestro programa hace **menos de** $3 \cdot 10^9$ operaciones.

Dependencia en el tamaño de la entrada

- El tiempo de ejecución depende del **tamaño de la entrada**, denotado por N
- **Idea:** buscamos una **cota superior** del número de operaciones en función de N.
- Como $3 \cdot 10^9$ es un número grande, podemos permitirnos **aproximaciones asintóticas** (qué pasa cuando $N \to \infty$).
- Ejemplo: cuando N es grande
$$N^2 + N \sim N^2.$$
- En práctica competitiva el límite de tiempo suele ser bastante holgado, incluso **20x más de lo necesario**.
  - Nos importa diferenciar N de N^2.
  - Pero no distinguir entre $0{,}5N$ y $10N$.
- Para esto usamos la **notación O grande**.

Definición formal de notación O

Notación O

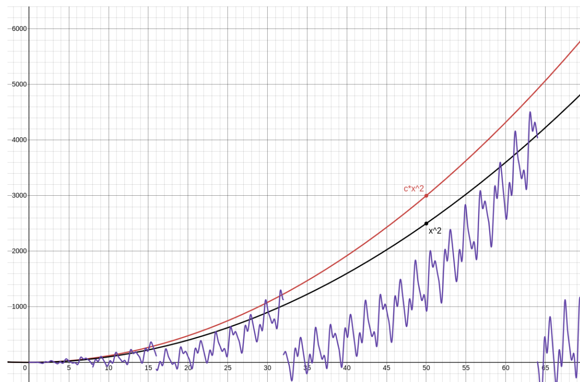
Matemáticamente, si $f : \mathbb{N} \to \mathbb{R}^+$ representa el tiempo de ejecución en función del tamaño de la entrada N, entonces:

- $O(f(N))$ es el conjunto de todas las funciones $g : \mathbb{N} \to \mathbb{R}^+$ tales que
- existen una constante $C > 0$ y un natural N_0 tal que para todo $N \geq N_0$,

$$g(N) \leq C \cdot f(N).$$

Cuando decimos que un programa tiene **costo** $O(f)$, significa que la función que representa el **peor caso** de tiempo de ejecución para entradas de tamaño N pertenece a $O(f)$.

Definición informal de notación O



Una funcion es $O(f)$ si, para valores grandes de N, sus picos mas altos no superan a f multiplicada por alguna constante.

Operaciones básicas: $O(1)$

- Muchas operaciones simples (y no tan simples) tienen **costo constante** $O(1)$.

Ejemplo

```
int x = 0;
x += 1; // O(1)
x *= 5; // O(1)
x %= 3; // O(1)
x += sqrt(x * 10); // O(1)

```

Bucles lineales: $O(N)$

- Un bucle típico de N pasos tiene costo $O(N)$.

Ejemplo

```
int sum = 0;
for (int i = 0; i < N; ++i) { // O(N) iteraciones
  sum += i; // O(1)
}
// total: O(N)

```

Bucles anidados: multiplicar costos

- Si tenemos algo dentro de un bucle, el costo se **multiplica** por la cantidad de operaciones.

Ejemplo

```
for (int i = 0; i < N; ++i) {    // O(N) iteraciones
  for (int j = 0; j < M; ++j) { // O(M)
    ...
  }
}
// total: O(N * M)

```

Sumar costos: solo el término dominante

- Código en líneas consecutivas se **suma**.
- Pero en notación O solo nos importa el **término superior**.

Ejemplo

```
for (int i = 0; i < N*N; ++i) { ... }      // O(N^2)
for (int i = 0; i < 3*N+15; ++i) { ... }   // O(N)
for (int i = 0; i < N*N-5; ++i) { ... }    // O(N^2)
// total: O(N^2 + N + N^2) = O(N^2)

```

Algoritmos de dos punteros

- Hay patrones que **rompen** la intuición simple de "bucle dentro de bucle $= O(N^2)$".
- Un ejemplo clásico son los **algoritmos de dos punteros**.

Ejemplo

```
int j = 0;
for (int i = N; i > 0; --i) {              // O(N) iteraciones
  while (j < N && a[j] + a[i-1] < x) {     // O(N)
    j++;
  }
}
// total: O(N)
// aunque a primera vista parezca O(N^2)

```

Patrones logarítmicos

- Cuando un número se va **multiplicando o dividiendo por 2** hasta alcanzar una cota, suele aparecer un $\log N$.

Ejemplo

```
1 for (int x = 1; x < N; x *= 2) { ... }
2 for (int x = N; x > 1; x /= 2) { ... }
3 // total: O(log N)
4
```

Suma armónica: $O(N \log N)$

- Aparece a veces (e.g. Criba de Eratostenes, algunas DPs) un patrón donde el número de iteraciones del bucle interno es N/i.
- La suma resultante se comporta como $N \log N$.

Ejemplo

```
1 for (int i = 1; i < N; ++i) {          // O(N) iteraciones
2   for (int j = i; j < N; j += i) {     // O(N / i) iteraciones
3     ... // O(1)
4   }
5 }
6 // total: O(N/1 + N/2 + ... + N/N) = O(N log N)
7
```

¿Qué costo es aceptable para cada N?

- Asumimos límite de tiempo ≈ 1 segundo y una implementación razonable en C++.

N	Costo
≤ 100	$O(N^4)$
≤ 500	$O(N^3)$
$\leq 10^4$	$O(N^2)$
$\leq 10^5$	$O(N\sqrt{N})$
$\leq 10^6$	$O(N \log N)$
$\leq 10^7$	$O(N)$
$\leq 10^{18}$	$O(\log N)$

Ideas clave

- No podemos conocer el **tiempo exacto**, pero sí su **orden de magnitud**, y con eso alcanza.
- La notación $O(\cdot)$ nos permite razonar sobre el **peor caso** de tiempo en función de N.
- Aprender a reconocer patrones típicos ($O(N)$, $O(N \log N)$, $O(N^2)$, etc.) es crucial para programar bajo límite de tiempo.

Q&A