Complejidad Asintótica e Introducción a C++

Complejidad en programación competitiva

- En programación competitiva hay **límites de tiempo** estrictos
- Un programa que exceda el límite dará el veredito **TLE** (Time Limit Exceeded)
- Necesitamos estimar que tan rápido será nuestro código antes de enviarlo

¿Cómo puedo calcular cuánto se demora mi programa?

- Medir con un cronómetro **depende del entorno** (hardware, lenguaje, etc.)
- Buscamos un **modelo teórico** que anticipe el rendimiento
- Hoy aprenderemos a **estimar** ese tiempo de forma práctica

¿Qué es un algoritmo?

Antes de comenzar: ¿Qué es un algoritmo?

- **Secuencia finita** de pasos bien definidos
- Transforman una entrada en una salida
- Deben ser **correctos** y, para competir, **eficientes**

¿Cómo sé cuál algoritmo es más rápido?

- Supongamos dos algoritmos que resuelven el **mismo problema**
- ¿Cuál elijo?

Factores externos que influyen

- Lenguaje de programación (C++, Python, Java…)
- Potencia del computador
- Sistema operativo y compilador

> Necesitamos una herramienta que **abstraiga** estos factores.

Hacia una métrica independiente

Queremos encontrar funciones:

$$T_1(n) = ? \qquad\qquad T_2(n) = ?$$

que indiquen el **tiempo** según el tamaño de entrada n.

Luego, simplemente **comparamos** $T_1(n)$ y $T_2(n)$.

El problema práctico

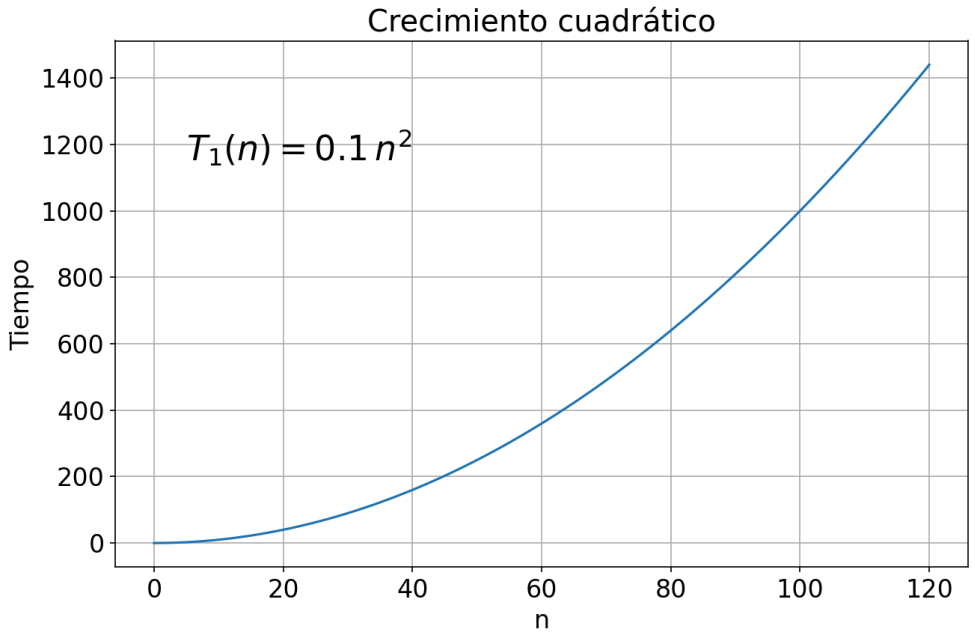
- Calcular $T(n)$ **exactamente** es casi imposible
- Depende de constantes ocultas y del hardware

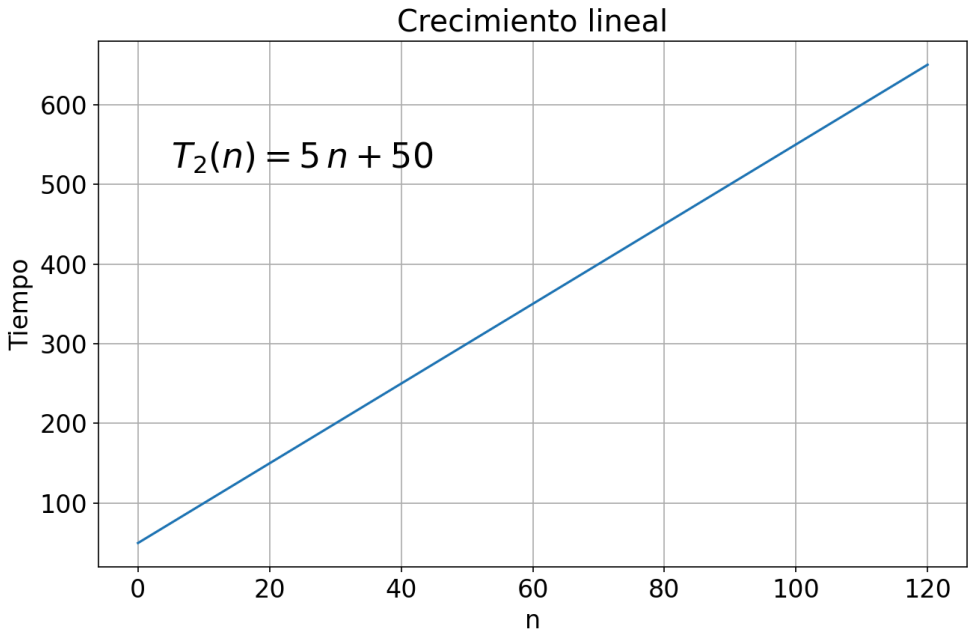
La solución

- Renunciamos a saber la constante exacta
- Nos enfocamos en **cómo crece** $T(n)$ cuando n aumenta

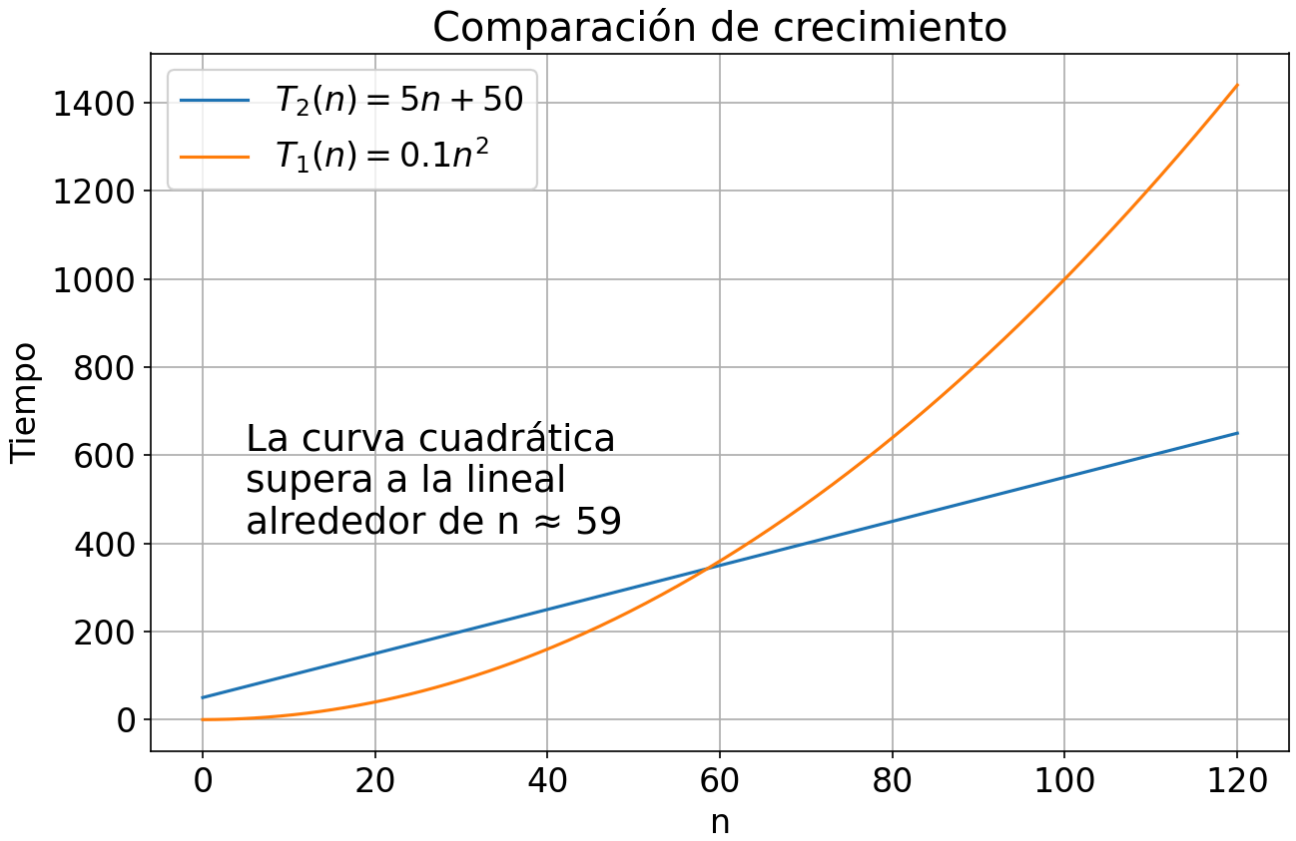
Crecimiento de funciones

Digamos por ejemplo que $T_1(n) = 0.1n^2$ y $T_2(n) = 5n + 50$





Crecimiento de funciones



- A medida que n crece, $T_1(n)$ **domina** a $T_2(n)$
- Para n suficientemente grande, $T_1(n)$ es **mucho mayor**

Notación asistótica O

- Describe **cómo crece** $T(n)$
- Ignora constantes y detalles de hardware

$$T_1(n) = 0.1n^2 \implies T_1(n) \in O(n^2)$$
$$T_2(n) = 5n + 50 \implies T_2(n) \in O(n)$$

> O(f) es el conjunto de funciones que crecen **como máximo** tan rápido como $f(n)$
>
> .

¿Cómo calcular la notación O?

- Estimaremos la cantidad de operaciones que realiza un algoritmo, dependiendo de la entrada n.
- Usaremos la notación O para expresar el crecimiento de estas operaciones.

¿Cómo calcular la notación O?

```python
def es_primo(x):
    for i in range(2, x):
        if x % i == 0:
            return False
    return True


n = int(input())

contador = 0
for i in range(2, n + 1):
    if es_primo(i):
        contador += 1

print(contador)
```

Ejemplo Nº1

- El algoritmo cuenta los números primos en el rango $[2, n]$
- ¿Cuántas operaciones hace?
- El bucle externo itera $n - 1$ veces
- La función `es_primo(x)` tiene un bucle que itera $n - 2$ veces
- Total de operaciones: $(n - 2) \cdot (n - 1) = n^2 - 3n + 2$
- Por lo tanto, $T(n) \in O(n^2)$

¿Cómo calcular la notación O?

```python
def es_primo(x):
    for i in range(2, x):
        if x % i == 0:
            return False
    return True


n = int(input())

contador = 1
for i in range(3, n + 1, 2):
    if es_primo(i):
        contador += 1

print(contador)
```

Ejemplo Nº2

- Optimización del programa anterior
- Ahora no revisa los números pares
- ¿Cuántas operaciones hace?
- El bucle externo itera $\frac{n-1}{2}$ veces
- La función `es_primo(x)` sigue iterando $n-2$ veces
- Total de operaciones: $\frac{(n-1)(n-2)}{2}$
- Por lo tanto, $T(n) \in O(n^2)$

```python
def es_primo(x):
    for i in range(2, int(x**0.5) + 1):
        if x % i == 0:
            return False
    return True


n = int(input())

contador = 0
for i in range(2, n + 1):
    if es_primo(i):
        contador += 1

print(contador)
```

Ejemplo Nº3

- Otra optimización diferente
- Ahora `es_primo(x)` solo itera hasta $\sqrt{x}$
- ¿Cuántas operaciones hace?
- El bucle externo itera $n - 1$ veces
- La función `es_primo(x)` itera hasta $\sqrt{n}$ en el peor de los casos
- Total de operaciones: $(n - 1)(\sqrt{n} - 2)$
- Por lo tanto, $T(n) \in O(n^{3/2})$

```python
n = int(input())

es_primo = [True] * (n + 1)
es_primo[0] = es_primo[1] = False

for i in range(2, n + 1):
    if es_primo[i]:
        for j in range(i * 2, n + 1, i):
            es_primo[j] = False

contador = 0
for i in range(2, n + 1):
    if es_primo[i]:
        contador += 1
print(contador)
```

Criba de Eratóstenes

- Implementación de la Criba
- ¿Cuántas operaciones hace?
- El bucle externo itera $n - 1$ veces
- El bucle interno itera $\frac{n}{i}$ veces en el peor de los casos
- Total de operaciones: $\sum_{i=2}^{n} \frac{n}{i}$
- Por lo tanto, $T(n) \in O(n \log n)$

¿Cómo estimo el tiempo de ejecución?

- Una vez que tenemos la complejidad, podemos estimar el tiempo de ejecución
- En programación competitiva, se suele usar el estimado de 10^8 operaciones por segundo
- Para un algoritmo con complejidad $T(n) \in O(f(n))$, el tiempo de ejecución se estima como:

$$\text{Tiempo estimado} \approx \frac{f(n)}{10^8}$$

¿Cómo estimo el tiempo de ejecución?

- Por ejemplo, si $T(n) \in O(n^2)$ y $n = 10^6$, entonces:
  - $(10^6)^2 \approx 10^{12}$ operaciones
  - El programa debería ejecutarse en **aproximadamente 10.000 segundos**
  - Esto es equivalente a **aproximadamente 2.8 horas**
- Por otro lado, si $T(n) \in O(n \log n)$, entonces:
  - $10^6 \log_2(10^6) \approx 10^6 \cdot 20 = 2 \cdot 10^7$ operaciones
  - El programa debería ejecutarse en **aproximadamente 0.2 segundos**

Introducción a C++

¿Por qué C++ en programación competitiva?

- Lenguaje de bajo nivel
- Gran **Velocidad**, comparable al C "puro"
- Biblioteca estándar muy completa (STL)
- Es el lenguaje dominante en la mayoría de los jueces en línea

¿Cómo instalar C++?

Sigue la guía recomendada para instalar el compilador de C++ en el editor de tu preferencia:

Compilación y editores — Apunte ProgComp UChile

- Instrucciones para Linux, Windows y Mac
- Recomendaciones de editores y entornos
- Ejemplos de comandos para compilar y ejecutar tus programas

Plantilla mínima

```cpp
#include <bits/stdc++.h>   // Toda la STL en un solo include

using namespace std;

int main() {

    /* — tu solución — */

    return 0;
}
```

- `bits/stdc++.h` no es parte del estándar, pero **todos los jueces** que usan GCC lo incluyen.

Tipos de variables básicos

Tipo	Rango aproximado / uso
`int`	$\pm 2 \cdot 10^9$
`long long`	$\pm 9 \cdot 10^{18}$
`double`	15 dígitos decimales
`char`	1 byte
`string`	Cadena de caracteres dinámicos
`vector<T>`	Arreglo dinámico

> Usa siempre **`long long`** cuando necesites enteros mayores a $\pm 2 \cdot 10^9$.

Entrada / salida

```cpp
int a;
int b;

cin >> a;
cin >> b;

cout << a + b;
cout << endl;
```

Input

```
3 7
```

Output

```
10
```

```cpp
int a, b;

cin >> a >> b;

cout << a + b << endl;
```

Input

```
3 7
```

Output

```
10
```

Condicionales

```
int a;
cin >> a;

if (a > 0) {
    cout << "Positivo" << endl;
} else if (a < 0) {
    cout << "Negativo" << endl;
} else {
    cout << "Cero" << endl;
}
```

Tipos principales

- **for**: Repetición con contador
- **while**: Repetición mientras se cumpla una condición

```
for (int i = 0; i < n; i++) {
    cout << i << " ";
}
```

Output

```
0 1 2 3 4
```

Tipos principales

- **for**: Repetición con contador
- **while**: Repetición mientras se cumpla una condición

```
int j = 0;
while (j < n) {
    cout << j << " ";
    j++;
}
```

Output

```
0 1 2 3 4
```

```cpp
int suma(int a, int b) {
    return a + b;
}
```

- El tipo va **antes** del nombre.
- Los argumentos se pasan por **valor** (copia).
- Ejemplo de uso: `cout << suma(3, 7) << endl;` muestra `10

```cpp
vector<int> v;           // vacío

v.push_back(42);         // añade al final
v.push_back(7);          // añade al final
v.push_back(13);         // añade al final

cout << v.size() << endl; // muestra 3

cout << v[1] << endl; // muestra 7
v.pop_back();         // elimina el último elemento

cout << v.back() << endl; // muestra 7
```

- Capacidad automática, memoria contigua.
- Métodos útiles: `push_back()`, `size()`, `clear()`, `back()`, `pop_back()`.

```cpp
int n;
cin >> n;
vector<int> v(n);

for (int i = 0; i < n; i++) {
    cin >> v[i];
}

v[1] = 42;

for (int i = 0; i < n; i++) {
    cout << v[i] << " ";
}
```

Input

```
5
9 4 8 1 3
```

Output

```
9 42 8 1 3
```

Resolviendo problemas en C++

- Problema "Watermelon" - Codeforces
- Problema "Vanya and Fence" - Codeforces