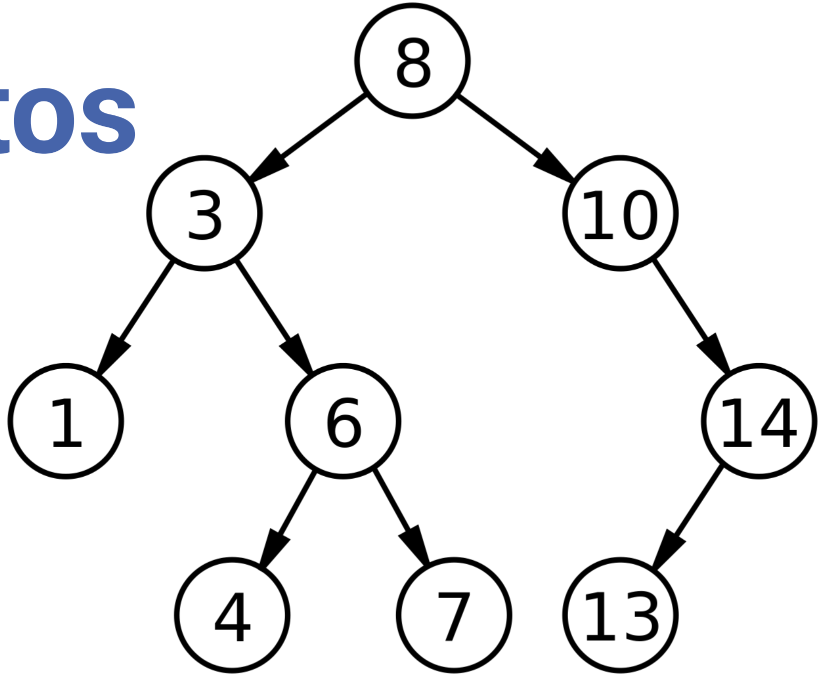
Estructuras de datos



Análisis asintotico

Notación Big-O, propiedades y reglas de los limites

Franco Cerda

Créditos: Sebastián Torrealba

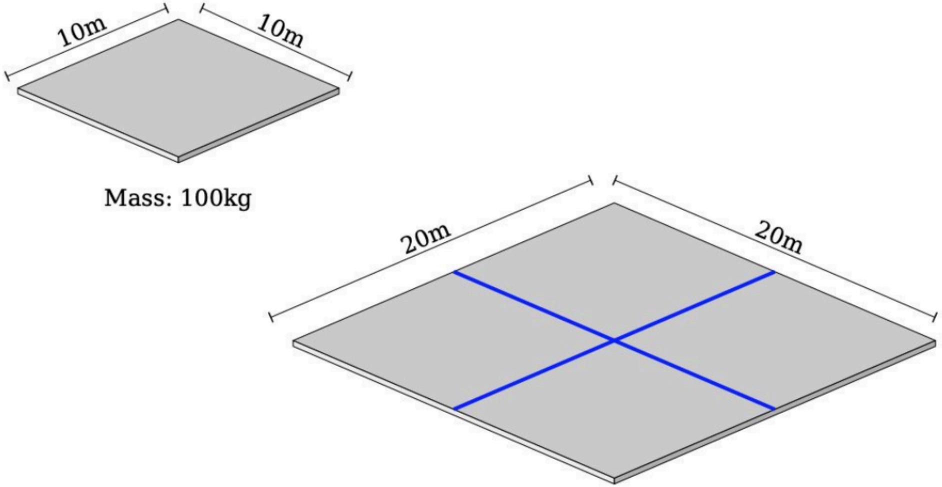
Motivación

Preguntas frecuentes:

- ¿Por qué existen algoritmos más rapidos que otros?
- ¿Cómo medir la eficiencia de un algoritmo?
- ¿Cómo formalizamos una notación para definir la eficiencia de los algoritmos?
- ¿Basta solo usar el tiempo de ejecución de un programa para compararlos?

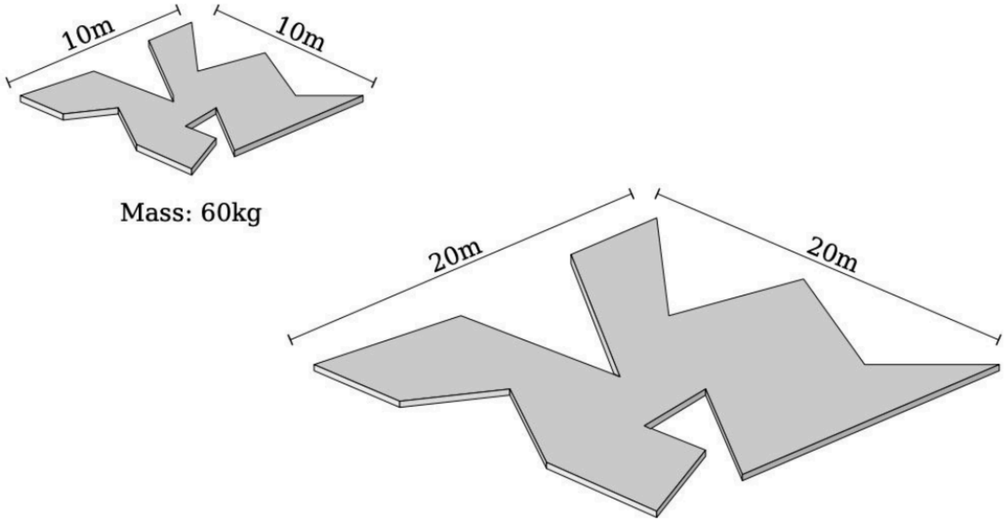
Estimar cantidades

¿Cual sería una estimacion correcta de la masa del siguiente ejemplo?



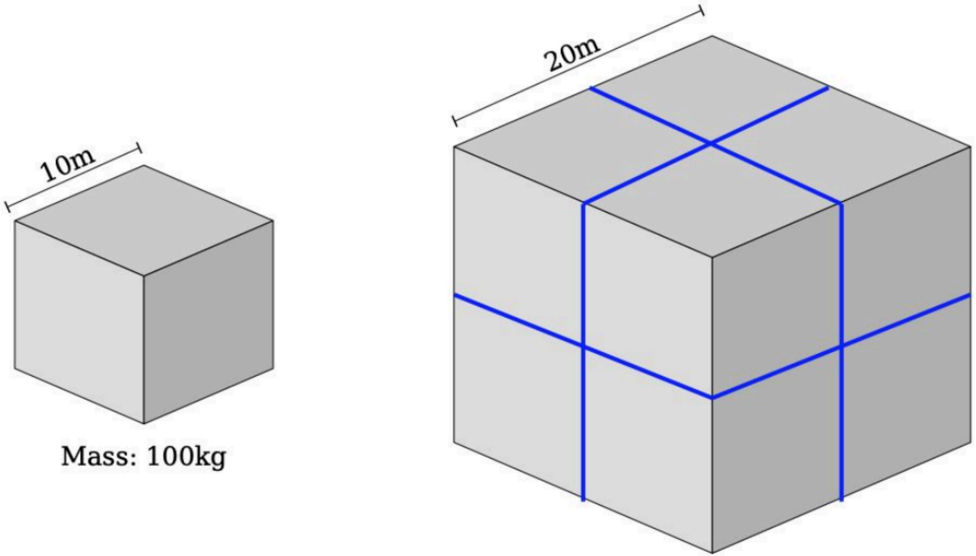
Estimar cantidades

¿Podemos seguir ocupando la misma estrategia al estimar la masa?



Estimar cantidades

¿Cuál sería la estimación de la masa en este caso?



¿Qué aprendimos?

Conocer la velocidad a la que escala una cantidad permite predecir su valor en el valor en el futuro, aunque no tengamos una fórmula exacta.

La pregunta es: ¿Cómo definimos que tan velozmente escala una cantidad formalmente?

Notación Big-O

Es una forma de cuantificar la velocidad a la que crece una cantidad. En el caso de los algoritmos, le decimos complejidad algoritmica.

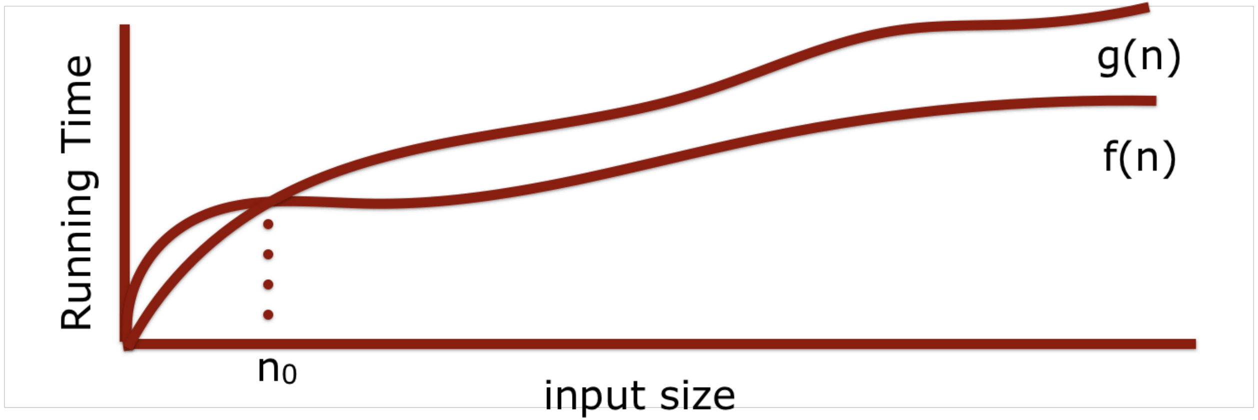
Se ocupan funciones para cuantificar la velocidad.

Definición matematica

Las funciones $f(n)$ y $g(n)$ son funciones que asignan números reales a números enteros no negativos. Decimos que $f(n)$ es $O(g(n))$ si existe una constante real $c > 0$ y una constante entera $n_0 \geq 1$, tal que $f(n) \leq c \cdot g(n)$ para cada entero $n \geq n_0$. Esta definición se conoce como la notación “big-Oh” o “notación O grande”. También podemos decir que “$f(n)$ es del orden de $g(n)$”.

Notación Big-O

En palabras simples, podemos decir que $f(n)$ es del orden de $g(n)$ si en un punto en adelante la función $c \cdot g(n)$ siempre es mayor que la función $f(n)$.



Notación Big-O

Para denotar que una función $f(n)$ crece a un ritmo $g(n)$ decimos que $f(n) \in O(g(n))$

Ejemplos:

- Un cuadrado de largo L su area crece a un ritmo de $O(L^2)$

  El area de un cuadrado simplemente es multiplicar sus lados

- Un circulo de radio r su area crece a un ritmo de $O(r^2)$

  El area de un circulo es πr^2, no nos interesa incluir las constantes en la notación Big-O

Propiedades de la notación Big-O

- Regla de la suma

$$f_1 \in O(g), f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\max(g, h))$$

En otras palabras, nos quedamos con la más “grande”.

- Regla del producto

$$f_1 \in O(g), f_2 \in O(h) \Rightarrow f_1 \cdot f_2 \in O(g \cdot h)$$

En otras palabras, podemos multiplicar dos funciones, sin ningún problema

¿Cómo podemos saber que función es “más” grande?

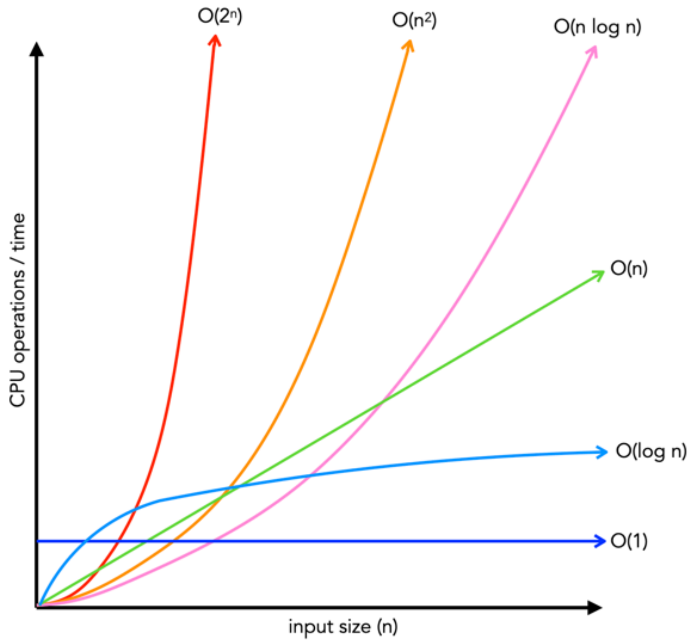
Para esto existe algo llamado *regla del limite*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = k$$

El valor de k puede ser uno de las tres opciones en el siguiente **orden**:

- Si $k = 0$, implica que, $O(f(n)) \in O(g(n))$
- Si $k = \infty$, implica que, $O(g(n)) \in O(f(n))$
- Si $k \neq 0$, implica que $O(g(n)) \in O(f(n))$, como también, $O(f(n)) \in O(g(n))$

Clasificación de cotas asintoticas



Funciones más comunes en la notación Big-O 🥑

- Polinomicas de grado k: $O(n^k)$
- Logaritmicas: $O(\log n)$
- Exponenciales: $O(k^n)$
- Raiz cuadrada: $O(\sqrt{n})$

Y combinaciones de estas mismas usando las propiedades mencionadas anteriormente

¡Formulario! 🥑

- Suma de los primeros n terminos:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Cambio de base para logaritmos

$$\log_b a = \frac{\log_c a}{\log_c b}$$

Ejercicio de ejemplo

- Demostrar que $\sum_{i=0}^{n} i = O(n^2)$
- Demostrar que los logaritmos son proporcionales en la notación Big-O, es decir:

$$O(\log_a n) \in O(\log_b n) \wedge O(\log_b n) \in O(\log_a n)$$

- Demostrar que $O(1) \in O(\lfloor \pi \rfloor)$

¿Cómo podemos identificar la complejidad en código?

Tenemos que definir una operación elemental, esta operación tendrá complejidad $O(1)$, es decir, constante.

Operaciones elementales **pueden** ser:

- Operaciones logicas y aritmeticas
- Mostrar datos por pantalla

Las operaciones elementales **suelen** ser las que no dependen del tamaño de la entrada del programa.

Ejemplo código 1

¿Cuál sería la complejidad de este código?

```
int suma = 0;
for (int i = 0; i < n; i++)
  suma += 5;
```

*Recuerda que n es nuestra variable**

Ejemplo código 2

¿Cuál sería la complejidad de este código?

```
int suma = 0
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    suma += 2;
```

*Recuerda que n es nuestra variable**

Ejemplo código 3

¿Cuál sería la complejidad de este código?

```
int suma = 0;
for (int i = 0; i < n; i++)
  for (int j = 0; j < i; j++)
    suma += 1;
```

*Recuerda que n es nuestra variable**

Ejemplo código 4

¿Cuál sería la complejidad de este código?

```
int suma = 0;
for (int i = 0; i < n; i++)
  for (int j = 0; j < m; j++)
    suma += n;
```

*Recuerda que n y m son las variables**

Ejemplo código 5

¿Cuál sería la complejidad de este código?

```
int suma = 0;
for (int i = 0; i*i < n; i++)
  suma += 1
```

Ten cuidado con la condición del ciclo 👀

Ejemplo código 5

¿Cuál sería la complejidad de este código?

```
for (int i = n; i > 0; i /= 2)
  cout << "Hola" << endl;
```

Ejemplo código 6

¿Cuál sería la complejidad de este código?

```
for (int i = 0; i < (n % 10); i++)
  cout << "Cuidadito!" << endl;
```

¡Fin!