# Appunti Analisi e Progettazione Algoritmi

**Lorenzo Vaccarecci**

**A.A. 2023/2024**

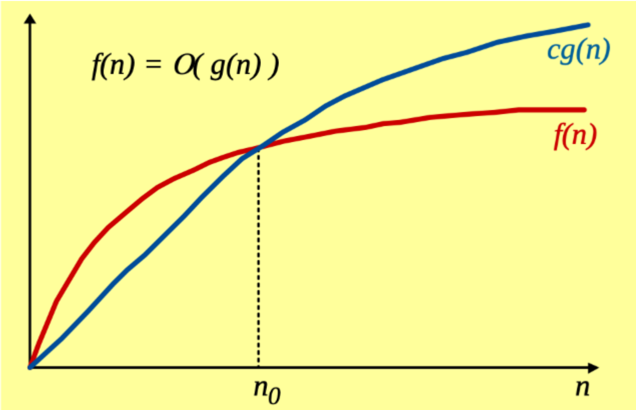
# Indice

# Capitolo 1

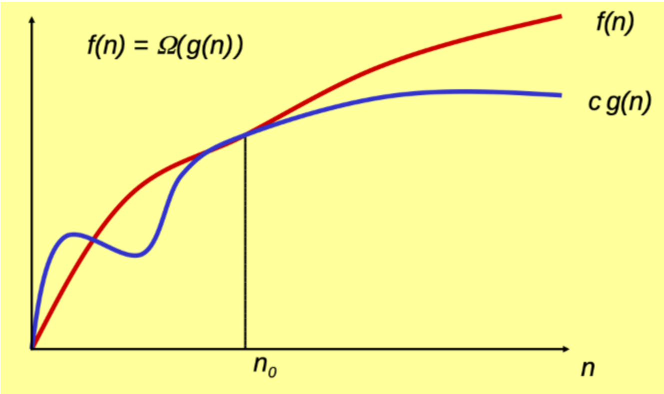
# Analisi della correttezza e complessità degli algoritmi

## 1.1 Notazioni asintotiche

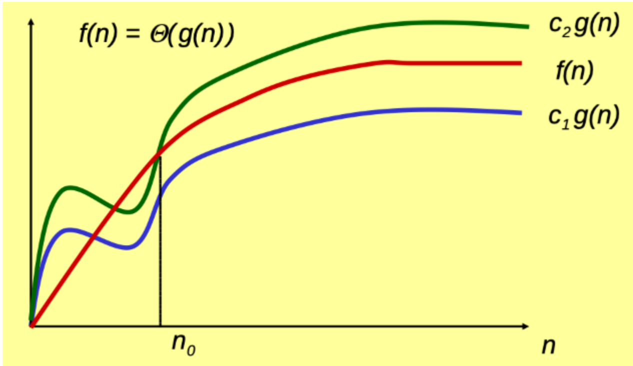
- Una funzione $f(n)$ appartiene all'insieme $O(g(n))$ se da un certo punto $n_0$ in poi, $f$ sta sotto una funzione "multipla" di $g$




- Una funzione $f(n)$ appartiene all'insieme $\Omega(g(n))$ se da un certo punto $n_0$ in poi, $f$ sta sopra una funzione "sottomultipla" di $g$




- Una funzione $f(n)$ appartiene all'insieme $\Theta(g(n))$ se da un certo punto $n_0$ in poi, $f$ è compresa tra un "multiplo" di $g$ e un sottomultiplo di $g$




### 1.1.1 Proprietà della notazione asintotica

**Transitiva**

- $f(n)$ è $O(g(n))$ e $g(n)$ è $O(h(n))$ implica $f(n)$ è $O(h(n))$
- $f(n)$ è $\Omega(g(n))$ e $g(n)$ è $\Omega(h(n))$ implica $f(n)$ è $\Omega(h(n))$
- $f(n)$ è $\Theta(g(n))$ e $g(n)$ è $\Theta(h(n))$ implica $f(n)$ è $\Theta(h(n))$

**Riflessiva**

- $f(n)$ è $O(f(n))$
- $f(n)$ è $\Omega(f(n))$
- $f(n)$ è $\Theta(f(n))$

**Simmetrica**

$f(n)$ è $\Theta(g(n))$ se e solo se $g(n)$ è $\Theta(f(n))$

**Simmetrica trasposta**

$f(n)$ è $O(g(n))$ se e solo se $g(n)$ è $O(f(n))$

**Somma**

$f(n) + g(n)$ è $O(\max\{f(n), g(n)\})$, analogamente per $\Omega$ e $\Theta$

**Comportamenti notevoli intrattabili**

- $\Theta(2^n)$ esponenziale
- $\Theta(n!)$ fattoriale
- $\Theta(n^n)$ esponenziale in base $n$

## 1.2 Complessità di algoritmi e problemi

**Legenda:**

- $n$: dimensione dell'input
- $i$: un input
- $t(i)$: tempo di esecuzione dell'algoritmo per l'input i
- $s(i)$: spazio di memoria necessario per l'esecuzione dell'algoritmo per l'input i

**Complessità temporale**

- **Caso peggiore** $T_{worst}(n) = \max\{t(i)|i \text{ ha dimensione } n\}$
- **Caso migliore** $T_{best}(n) = \min\{t(i)|i \text{ ha dimensione } n\}$
- **Caso medio** $T_{avg}(n) = \text{avg}\{t(i)|i \text{ ha dimensione } n\}$

Per il caso medio, per ogni $n$ si considerano tutti i possibili input di dimensione $n$, siano $i_1, \dots, i_N$ e si fa la media aritmetica dei tempi:
$T_{avg}(n) = \frac{t(i_1)+\ldots+t(i_N)}{N}$
Ovviamente, se i possibili casi di input non sono equiprobabili la media deve essere una media pesata:
$T_{avg}(n) = p_1 \cdot t(i_1) + \ldots + p_N \cdot t(i_N)$
Definizioni:

- Un algoritmo ha complessità $O(f(n))$ se $T_{worst}(n) = O(f(n))$
- Un algoritmo ha complessità $\Omega(f(n))$ se $T_{best}(n) = \Omega(f(n))$
- Un algoritmo ha complessità $\Theta(f(n))$ se ha complessità $O(f(n))$ e $\Omega(f(n))$

Per gli algoritmi randomizzati il caso peggiore è definito come:
$T_{exp}(i) = \mathbb{E}[t(i,c)]$ Da completare la spiegazione di questa parte
$T_{exp\_worst}(n) = \max(T_{exp}(i)|i \text{ ha dimensione } n)$
**Problema $\neq$ Algoritmo**

- **Delimitazione superiore**: Un problema ha complessità $O(f(n))$ se esiste un algoritmo di complessità $O(f(n))$ che lo risolve
- **Delimitazione inferiore**: Un problema ha complessità $\Omega(f(n))$ se tutti i possibili algoritmi risolventi hanno complessità $\Omega(f(n))$

Quindi per trovare una delimitazione superiore è sufficiente (e necessario) trovare un algoritmo che risolva il problema in un tempo $O(f(n))$, mentre, per trovare una delimitazione inferiore è necessario dimostrare che qualunque possibile algoritmo deve impiegare un tempo $\Omega(f(n))$.
**Un problema è chiuso se si conoscono limite superiore e inferiore coincidenti**:

- Esiste un algoritmo risolvente di complessità $O(f(n))$
- Si è dimostrato che qualunque algoritmo risolvente deve avere complessità $\Omega(f(n))$, ossia non può esistere un algoritmo di complessità inferiore a $\Omega(f(n))$

Si è quindi dimostrato che l'algoritmo risolvente è ottimo. **Un problema è aperto se (tutte le) delimitazioni inferiori e superiori differiscono** e di conseguenza ha un gap algoritmo. Un gap algoritmo può essere chiuso in due modi:

- **Dal di sopra**: si trova un algoritmo migliore, abbassando così il limite superiore
- **Dal di sotto**: si riesce a dimostrare un limite inferiore più alto

I due modi non sono mutuamente esclusivi, ossia può succedere che si riesca a trovare un algoritmo migliore, e contemporaneamente si riesca a dimostrare un limite inferiore più alto.

## 1.3 Corretteza di algoritmi ricorsivi

E' basata sul principio di induzione.
Un insieme definito induttivamente è un insieme i cui elementi sono tutti e soli quelli che si ottengono applicando ripetutamente un insieme di regole, a partire da quelle con premessa vuota che costituiscono la base della definizione induttiva.

### Esempio: Serie geometrica

Proviamo per induzione aritmetica che $P(n) = \sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$, per tutti gli $n \in \mathbb{N}$.
**Base** $P(0) = q^0 = 1 = \frac{q^1-1}{q-1}$
**Passo induttivo** Assumiamo $P(n)$ e dimostriamo $P(n+1)$:

$$P(n+1) = \sum_{i=0}^{n+1} q^i \rightarrow q^0 + \ldots + q^n + q^{n+1} \rightarrow \left(\sum_{i=0}^{n} q^i\right) + q^{n+1}$$

Svolgendo i calcoli si riesce a determinare che viene

$$\frac{q^{(n+1)+1} - 1}{q - 1}$$