Analisi di mergeSort ed equazioni di ricorrenza

corso di laurea in **Matematica**

Informatica Generale, Lezione **9.2**

Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Analisi di mergeSort

Analizziamo la complessità dell'algoritmo **ricorsivo**, versione base, vedremo poi ottimizzazioni che **non migliorano la complessità asintotica** (anche se possono essere **molto più efficienti**).

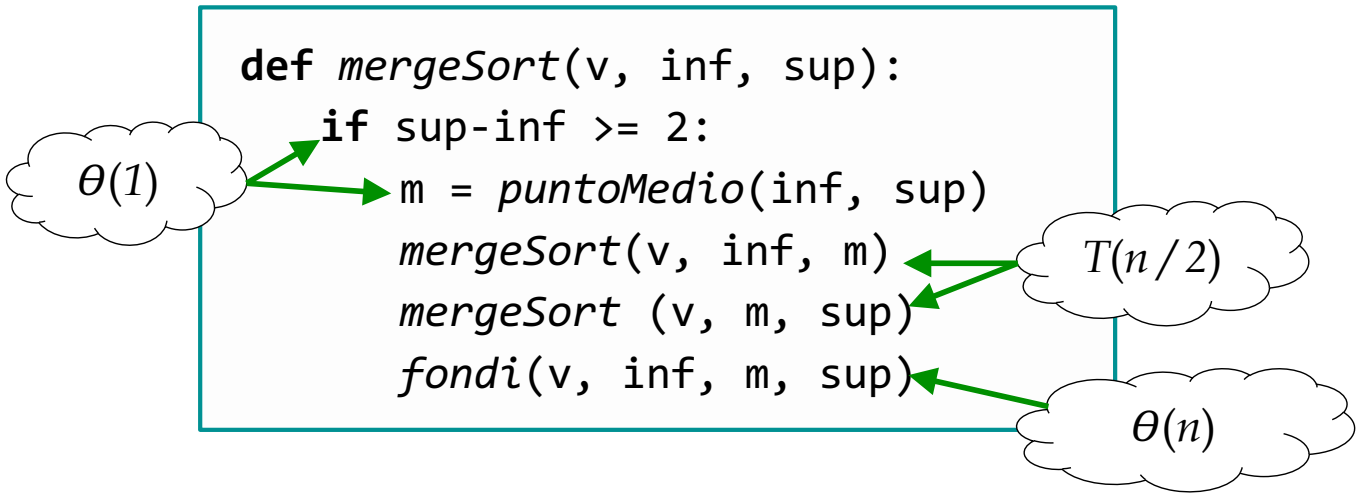
Divisione e caso base sono chiaramente costanti.

Sappiamo che la **fusione è lineare** e chiamando $n = sup - inf$…

Infine, detta $T(n)$ la complessità di `mergeSort`, le due **chiamate ricorsive** avranno ovviamente complessità $T(n/2)$.

Abbiamo quindi la **relazione di ricorrenza**:

$$T(n) = 2T(n/2) + \Theta(n) \qquad T(1) = \Theta(1)$$



Review: metodo iterativo

Risolviamo l'equazione con il **metodo iterativo**:

$$T(n) = 2\,T(n/2) + \theta(n) =$$
$$= 2[2T(n/2^2) + \theta(n/2^1)] + \theta(n/2^0) =$$
$$= 2[2[2T(n/2^3) + \theta(n/2^2)] + \theta(n/2^1)] + \theta(n/2^0) =$$
$$= 2^3T(n/2^3) + 2^2\theta(n/2^2) + 2^1\,\theta(n/2^1) + 2^0\,\theta(n/2^0) =$$

… (si "**srotola**" fino a una **forma generale**)

$$= 2^k\,T(n/2^k) + \sum_{i=0}^{k-1} 2^i\theta\left(\frac{n}{2^i}\right)$$

fino a $2^k = n$, cioè $k = \log_2 n$. Sapendo che $2^{\log n} = n$ ottenendo:

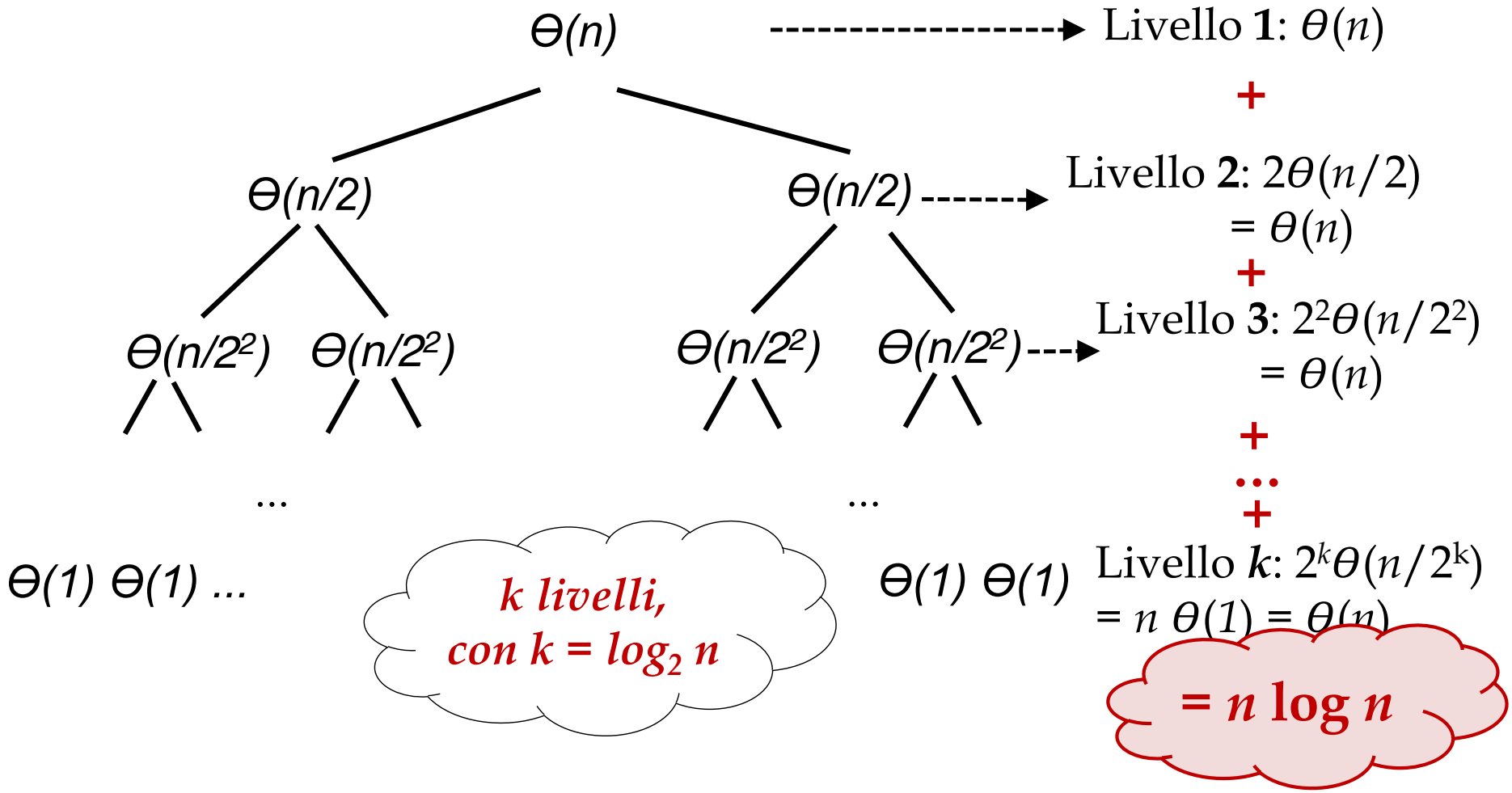
$$T(n) = n\,\theta(1) + \sum_{i=0}^{\log n-1} 2^i\theta\left(\frac{n}{2^i}\right) =$$
$$= \theta(n) + n\sum_{i=0}^{\log n-1} \theta\left(\frac{2^i}{2^i}\right) =$$
$$= \theta(n) + n\sum_{i=0}^{\log n-1} \theta(1) =$$
$$= \theta(n) + \theta(n \log n) = \theta(n \log n).$$

"pericolosi" movimenti di costanti dentro e fuori θ!!!

Metodo dell'albero (iterativo plus ☺)

Il **metodo dell'albero** una versione "migliorata" del metodo iterativo, utile soprattutto per procedure Divide et Impera.

Consiste nel **visualizzare i livelli** dell'**albero di ricorsione**.



Metodo di sostituzione

Questo è il metodo **più potente**, ma **più pericoloso** ☺ e direi, destinato a un **pubblico adulto**.

Idea: **ipotizzare una soluzione** e **verificare** che soddisfi l'equazione.

Difficoltà:

- formulare una **buona ipotesi** (usualmente serve **esperienza**)
- evitare i tranelli della notazione asintotica (occorre eliminare Θ, $\mathcal{O}$, e $\boldsymbol{\Omega}$ e considerare **esplicitamente le costanti**)

Vediamo nel nostro caso.

a) si **elimina la notazione asintotica**, in quanto le costanti nascoste potrebbero essere diverse. Otteniamo quindi: $T(n)=2\cdot T(n/2) + c\cdot n$ e $T(1) = d$ con $c, d > 0$.

b) **Formuliamo l'ipotesi** $T(n) = \Theta(n \log n)$ e

c) **mostriamo separatamente** per **induzione su *n*** che $T(n)=\mathcal{O}(n \log n)$ e $T(n) = \Omega(n \log n)$.

$\mathcal{O}(n \log n)$ e $\Omega(n \log n)$ a loro volta **lasciano la libertà di fissare delle costanti**: sono nella forma $k\, n \log n + h$ per **costanti arbitrarie** k e h.

Metodo di sostituzione: esempio

- $T(n) = \mathcal{O}(n \log n)$: significa trovare opportune costanti h, k per cui:

Caso Base: $T(1) = d \leq k\,1 \log 1 + h$. Ciò è vero per opportuna scelta di $h > d$ (h deriva dall'eliminazione della notazione asintotica!)

Passo Induttivo:

$$\begin{aligned} T(n) &= 2\,T(n/2) + cn && = \{\textbf{induzione}\} \\ &\leq 2\,(k\,n/2 \log n/2 + h) + cn && = \{\textbf{svolgo}\} \\ &= k\,n \log n/2 + 2h + cn && = \{\textbf{logaritmo}\} \\ &= k\,n\,(\log n - 1) + 2h + cn && = \{\textbf{svolgo}\} \\ &= k\,n \log n - k\,n + 2h + cn = k\,n\,log\,n + h + h + cn - kn \end{aligned}$$

che è minore di $k\,n\,log\,n + h$ a patto che $h + cn - kn$ sia negativa, cioè se $cn + h \leq k\,n$ e posso scegliere semplicemente $k \geq c$ a questo fine.

- $T(n) = \Omega(n \log n)$. Troviamo opportune costanti h', k' per cui:

Caso Base: $T(1) = d \geq k\,1 \log 1 + h'$ (esempio $h' = 0$).

Passo Induttivo: (scelgo subito $h' = 0$).

$$\begin{aligned} T(n) &\geq \{\textbf{induzione}\}\ 2\,k'\,(n/2 \log n/2) + cn = \{\textbf{svolgo \& log}\} \\ &= k'\,n \log n - k'\,n + cn \end{aligned}$$

che è maggiore di $k'\,n\,log\,n$ a patto che $cn - k'n$ sia **positiva**, cioè per $cn \geq k'n$ e quindi per $c \geq k'$.

Teorema Principale

Quando **divido** in il problema originale in b **istanze** e ne **risolvo** a, si può trovare (**quasi sempre**) la **soluzione generale** dell'equazione di ricorrenza:

$$T(n) = a\,T(n/b) + \theta(f(n)) \qquad T(1) = \theta(1)$$

Esempi:

- in **`mergeSort`** abbiamo $a = b = 2$ e $f(n) = n$.
- nella **ricerca binaria**, invece $a = 1$, $b = 2$, e $\theta(f(n)) = 1$.

Teorema [Principale o dell'**Esperto o Master Theorem]**. *Dati* $a \geq 1$ *e* $b > 1$ *e una funzione asintoticamente positiva* $f(n)$ *e un'equazione di ricorrenza nella forma* $T(n) = a\,T(n/b) + \theta(f(n))$ *e* $T(n)=\theta(1)$*, vale che:*

1. *se* $f(n) = \mathcal{O}(n^{\log_b a - \varepsilon})$ *per qualche* $\varepsilon > 0$ *allora* $T(n) = n^{\log_b a}$

2. *se* $f(n) = \theta(n^{\log_b a})$ *allora* $T(n) = n^{\log_b a}.\log n$

3. *se* $f(n) = \Omega(n^{\log_b a+\varepsilon})$ *per qualche* $\varepsilon > 0$ *e se* $a \cdot f(n/b) \leq c \cdot f(n)$ *per qualche* $c < 1$ *(per n sufficientemente grande) allora* $T(n) = \theta(f(n))$

Difficoltà del Teorema Principale

Il teorema principale prescrive, per risolvere un'equazione di ricorrenza nella forma:

$$T(n) = a\,T(n/b) + \theta(f(n)) \qquad\qquad T(1) = \theta(1)$$

di **confrontare** tra loro, le funzioni:

$$f(n) \qquad n^{\log_b a}$$

e (in prima approssimazione) la complessità è **governata** dalla **maggiore delle due**.

Attenzione: alle condizioni $f(n) = \mathcal{O}(n^{\log_b a - \varepsilon})$ e $f(n) = \boldsymbol{\Omega}(n^{\log_b a + \varepsilon})$ che significano che $f(n)$ deve essere **polinomialmente più piccola** oppure **polinomialmente più grande**.

Ciò significa che tra i casi **1** e **2** e tra i casi **2** e **3** c'è una **zona grigia** in cui il Teorema Principale **non ci assiste**.

Esempio: $T(n) = 2\,T(n/2) + \theta(n \log n)$ e $T(1) = \theta(1)$. In questo caso ho che $a = b = 2$ e $\log_2 2 = 1$, $n^1 = n$ e $n = \mathcal{O}(n \log n)$, ma **non c'è nessun ε per cui $n \log n > n^{1-\varepsilon}$** (in quanto asintoticamente ho che $\log n < n^{\varepsilon}$ per ogni $\varepsilon > 0$, calcolate ad esempio $\lim_{n\to\infty} \frac{\log n}{n^{\varepsilon}} = 0$).

Applicazione del Teorema Principale

Vediamo un po' di esempi (abbiamo sempre $T(1) = \theta(1)$), cominciando dal calcolo di due algoritmi già visti:

Ricerca Binaria

In questo caso abbiamo $T(n) = T(n/2) + \theta(1)$.

Quindi $a = 1$, $b = 2$ e quindi $\log_2 1 = 0$.

Inoltre $f(n)$ è una costante, come anche $n^{log_b\, a} = n^0 = 1$.

Quindi $f(n)$ è $\theta(1)$, siamo nel caso **2**. del Teorema Principale e la soluzione è $T(n) = \theta(n^{log_b\, a} \log n) = \theta(\log n)$.

mergeSort

In questo caso abbiamo $T(n) = 2T(n/2) + \theta(n)$.

Quindi $a = 2$, $b = 2$ e quindi $\log_2 2 = 1$.

Inoltre $f(n) = n$, come anche $n^{log_b\, a} = n^1 = n$.

Siamo ancora nel caso **2**. del Teorema Principale e la soluzione è $T(n) = \theta(n^{log_b\, a} \log n) = \theta(n \log n)$.

Applicazione del Teorema Principale

Vediamo altri esempi (abbiamo sempre $T(1) = \theta(1)$), stavolta di algoritmi che **non vedrete** in questo corso:

Ricerca Binaria su matrice quadrata $n \times n$ ordinata

In questo caso abbiamo $T(n^2) = 3\,T(n^2/4) + \theta(1)$.

Quindi $a = 3$, $b = 4$ e $\log_4 3 = 0.79248\ldots$

Inoltre $f(n^2)$ è una costante, e quindi $f(n^2) = \mathcal{O}((n^2)^{0.79247\ldots})$.

Quindi siamo nel caso **1**. del Teorema Principale e la soluzione è $T(n^2) = \theta((n^2)^{log_4\,3}) = \theta(n^{1.58496\ldots})$.

Algoritmo di Strassen per il prodotto tra matrici

In questo caso abbiamo $T(n) = 7\,T(n/2) + \theta(n^2)$.

Quindi $a = 7$, $b = 2$ e quindi $\log_2 7 = 2.80735\ldots$

Inoltre $f(n) = n^2$, e quindi $f(n) = \mathcal{O}(n^{2.80735\ldots})$

Siamo ancora nel caso **1**. del Teorema Principale e la soluzione è $T(n) = \theta(n^{log_2\,7}) = \theta(n^{2.80735\ldots})$.

Applicazione del Teorema Principale

Vediamo un esempio (sempre $T(1) = \theta(1)$), di algoritmo misterioso che ha la relazione di ricorrenza:

$$T(n) = 4\,T(n/2) + n^{5/2}$$

Abbiamo a=4, b=2 e $\log_2 4 = 2$.

Inoltre $f(n) = n^{5/2} = \mathbf{\Omega}(n^{2+\varepsilon})$ per $0 < \varepsilon \leq ½$. Quindi siamo nel caso **3.** del Teorema Principale.

Occorre in questo caso verificare anche una condizione supplettiva, e cioè $a\,f(n/b) < c\,f(n)$ per qualche $c < 1$.

Nel nostro caso vale, in quanto a $4\,(n/2)^{5/2} < c\,n^{5/2}$, osservando che $2^{5/2}=4\sqrt{2}$, otteniamo che l'equazione è soddisfatta per $1/\sqrt{2} < c$, ed essendo $1/\sqrt{2} < 1$, esistono soluzioni.

Quindi, $T(n) = \theta(n^{5/2}) = \boldsymbol{\theta(n^2\sqrt{n})}$.

Max Divide et Impera: Torneo

Vediamo ora un altro algoritmo *Divide et Impera*: l'obiettivo è il calcolo del **massimo** (di un vettore).

Invece di procedere sequenzialmente, osserviamo che se conosciamo m_{sx}= **max**$(v[0, i))$ e m_{dx} = **max**$(v[i, n))$ il massimo del vettore sarà $m = \max\{m_{sx}, m_{dx}\}$.

Questo è vero $\forall i \in [0, n)$, **estremi inclusi**, assumendo in tal caso **max**$(v[i, i)) = -\infty$. Ma per fare progressi, devo suddividere il vettore in due **segmenti non vuoti**.

In vista di una **procedura ricorsiva**, è sufficiente trovare un **caso elementare** (=caso base) **non banale**, che è il vettore di un elemento, perché ovviamente è vero $\forall i \in [0, n)$. **max**$(v[i, i+1)) = v[i]$.

```
def torneo(v, inf, sup):
    if inf + 1 == sup:   # caso base
            return v[inf]
    m = choose(inf+1, sup) # m ∈ [inf+1, sup)
    return max(torneo(v, inf, m),
            torneo(v, m, sup))
```

```
def choose(a, b):
    # PREC: a ≤ b
    # POST: return m, a ≤ m < b
```

fondamentale per la terminazione di torneo!

Massimo Div&Imp: Torneo

Con questa definizione, se `choose` **scegliesse sempre** *inf*+1, ci riduciamo alla ricerca **sequenziale**.

Probabilmente, per sfruttare il *Divide et Impera*, si può pensare sia meglio scegliere il **punto medio** per suddividere il lavoro.

Così facendo, ritroviamo l'**usuale "algoritmo"** per "**determinare il vincitore**" di molte **competizioni sportive**.

Pregio: tutti fanno al più log *n* confronti!

Complessità del Torneo

Ma in **totale**, **quante partite** si fanno?

Applichiamo il **Teorema Principale**! La relazione di ricorrenza è:

$$T(n) = 2\,T(n/2) + \theta(1) \qquad T(1) = \theta(1)$$

Abbiamo quindi $a = 2$, $b = 2$, quindi $\log_2 2 = 1$, e il costo di ricombinazione è $\theta(1)$, e chiaramente una costante è $\mathcal{O}(1)$. Quindi siamo nel caso **1.** del Teorema Principale, la complessità è $\boldsymbol{\theta(n)}$.

Alternativamente con il metodo **iterativo**…

$$\begin{aligned} T(n) &= 2\,T(n/2) + \theta(1) \\ &= 2(2(T(n/4)+ \theta(1))+ \theta(1) \\ &= \ldots \\ &= 2^{k-1}\theta(1)+ \textstyle\sum_{i=0,\ldots,k-2} 2^i\theta(1) \text{ con } k = \log_2 n \\ &= 2^{\log n}\theta(1) + 2^{\log n}\,\theta(1) = \\ &= n\,\theta(1) + n\,\theta(1) = \theta(n) \end{aligned}$$

$1+2+4+\ldots+2^{n-1}$ ***è il numero binario*** $111\ldots1 = 2^n - 1$

Complessità del Torneo

In questo caso, il *Divide et Impera* **non ha avuto successo**.

Non è sorprendente, in quanto il problema si risolveva in modo lineare anche senza il *Divide et Impera*, ed è **arduo** pensare di **fare progressi avendo già una soluzione lineare**.

Tuttavia, vedremo che l'**informazione** che si **può raccogliere** durante un torneo (grazie alla **transitività** di ≥) **è molto di più** di quella che si ottiene da una **scansione lineare** e può essere usata con successo in un algoritmo di ordinamento (`heapSort`).

Pensate ai seguenti problema: avendo il tabellone di un torneo, **quanti partite sono necessarie per conoscere il/la secondo/a concorrente più forte**? Ovviamente non è chi ha perso la finale…

E dopo aver determinato il massimo con una scansione lineare per il calcolo del minimo?

Analisi minimo ricorsivo

Vediamo l'algoritmo per calcolare il minimo di un vettore in forma **ricorsiva**.

Il caso base è costante così come il calcolo di del minimo tra due numeri (elemento corrente e risultato ricorsivo di `minV`).

La chiamata ricorsiva si applica a un vettore di lunghezza n – 1.

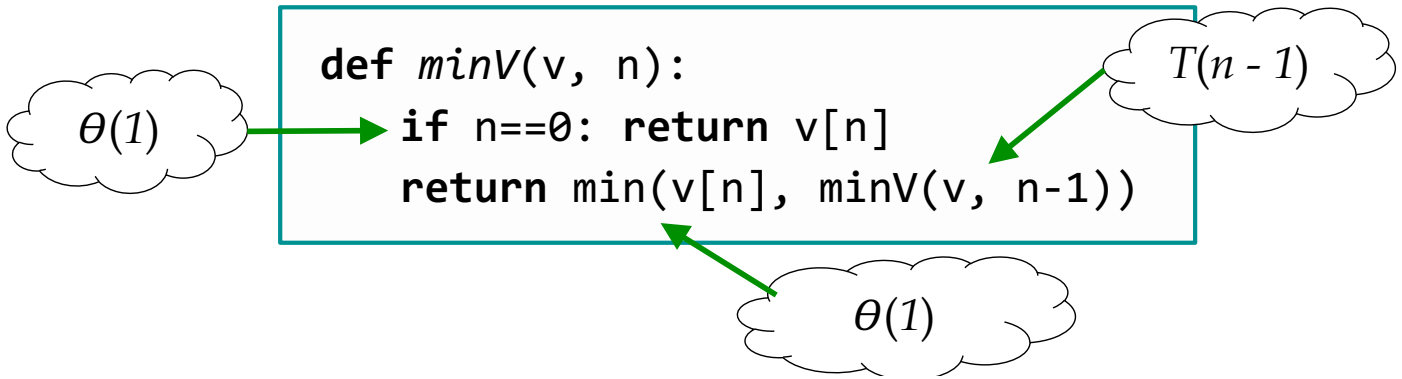
Abbiamo quindi la **relazione di ricorrenza**:

$$T(n) = T(n - 1) + \Theta(1) \qquad T(1) = \Theta(1)$$

Qui, è necessario (e sufficiente) applicare il metodo iterativo.

$T(n) = T(n - 1) + \Theta(1) = T(n - 2) + \Theta(1) + \Theta(1) = \ldots$

$= \sum_{i=1, \ldots, n} \Theta(1) = \Theta(n)$



insertionSort ricorsivo

Vediamo l'algoritmo insertionSort in forma **ricorsiva** (notate la concisione ☺)

Certo, bisogna scrivere **`insert`**. ➧ **Esercizio**.

Il caso base è costante mentre **`insert`** ha costo (pessimo) $\Theta(n)$.

La chiamata ricorsiva si applica a un vettore di lunghezza $n - 1$.

Abbiamo quindi la **relazione di ricorrenza**:

$$T(n) = T(n - 1) + \Theta(n) \qquad T(1) = \Theta(1)$$

Qui, è necessario (e sufficiente) applicare il metodo iterativo.

$T(n) = T(n - 1) + \Theta(n) = T(n - 2) + \Theta(n - 1) + \Theta(1) = \ldots$

$= \sum_{i=1, \ldots, n} \Theta(i) = \Theta(n^2)$ (Formula di Gauss)