RIPASSO SU COMPLESSITÀ COMPUTAZIONALE

Cosa è la complessità computazionale di un algoritmo / programma?

"È una funzione che restituisce la quantità di risorse utilizzate durante l'esecuzione del programma, in funzione della dimensione dell'input."

RISORSE UTILIZZATE

- $\underbrace{\text{TEMPO DI ESECUZIONE / N° OPERAZIONI ELEMENTARI}}_{\text{COMPLESSITÀ IN TEMPO}}$
- $\underbrace{\text{QUANTITÀ DI MEMORIA UTILIZZATA}}_{\text{COMPLESSITÀ IN SPAZIO}}$

DIMENSIONE DELL'INPUT

Una qualunque misura di quanto occupa in memoria l'input dato all'algoritmo.

Tipicamente, se l'input comprende un array, la dimensione dell'input è la lunghezza dell'array.

Esempio

```
public static boolean ricerca (int[] a, int x)
```

ALGORITMO: RICERCA LINEARE

INPUT: ARRAY a, l'elemento da cercare x

La lunghezza dell'input è la lunghezza di a.

Complessità in tempo potrebbe essere $C(n) = 2n + 7$

dove n è la lunghezza dell'array a.

Se non si dice esplicitamente, si intende la complessità nel caso pessimo.

Complessità
- CASO MEDIO
- $\boxed{\text{CASO PESSIMO}}$
- CASO OTTIMO

COMPLESSITÀ ASINTOTICA: ordine di grandezza (infinito) della complessità computazionale

Usiamo 3 notazioni per specificare gli ordini di grandezza.

Potete pensarli quasi come se fossero confronti con $\leq$ e $\geq$

talvolta si scrive impropriamente $=$ invece di $\in$

$(C(n) \leq f(n))$

$C(n) \in O(f(n))$ $\quad \lim_{n \to \infty} \frac{C(n)}{f(n)} \in \mathbb{R}$ $\quad$ "C cresce come f o meno di f"

$(C(n) \geq f(n))$

$C(n) \in \Omega(f(n))$ $\quad \lim_{n \to \infty} \frac{f(n)}{C(n)} \in \mathbb{R}$ $\quad$ "C cresce come f o più di f"

$(C(n) = f(n))$

$C(n) \in \Theta(f(n))$ quando $C(n) \in O(f(n))$ e $C(n) \in \Omega(f(n))$ $\quad$ "C cresce come f"

oppure $\lim_{n \to +\infty} \frac{C(n)}{f(n)} \in \mathbb{R} \setminus \{0\}$

Ricerca lineare (riprende l'esempio precedente)

n = dimensione input, ovvero il numero di elementi dell'array in cui devo effettuare la ricerca

CASO PESSIMO.

quando l'elemento che cerco non sta nell'array e.

Complessità in tempo: $\Theta(n)$ (devo guardare tutte le posiz. dell'array, che sono n, per capire che l'elemento che cerco non ne fa parte)

Complessità in spazio: 32 bit $\quad \Theta(1)$ (indipendentemente da quanto è grande l'array, serve solo una variabile intera per scorrere l'array e memorizzare la posizione corrente)

$f(n) = 1$

$g(n) = 2$

$\Rightarrow$ son entrambe $\Theta(1)$ $\quad$ numeri reali diversi da zero

$$\lim_{n \to \infty} \frac{f(n)}{1} = \lim_{n \to +\infty} \frac{1}{1} = 1$$

$$\lim_{n \to +\infty} \frac{g(n)}{1} = \lim_{n \to +\infty} \frac{2}{1} = 2$$

CASO OTTIMO

quando l'elemento che stiamo cercando è il primo dell'array.

Complessità in tempo : $\Theta(1)$ — siccome si ferma sul primo elemento, la lunghezza dell'array non influisce sul tempo di esecuzione.

Complessità in spazio : $\Theta(1)$ — usa una sola variabile indice per scorrere l'array (anche se si ferma subito)

In realtà la notazione $\Theta(-)$ si usa molto (moltissimo) poco.
Si usa quasi sempre $O(-)$.

Esempio : ricerca lineare

CASO PESSIMO

queste sono sbagliate

Complessità in tempo : $\Theta(n)$ $\cancel{\Theta(n^2)}$ $\cancel{\Theta(n^3)}$

$\underbrace{\boxed{O(n)},\ O(n^2),\ O(n^3),\ O(2^n)}$

Sono tutti corretti, ma ovviamente si preferisce dare la stima migliore possibile, e quindi $O(n)$

$3n+5 \in O(n)$

$3n+5 \in O(n^2)$

$3n+5 \in O(n^{40})$

$3n+5 \in O(2^n)$

queste sono sbagliate

Complessità in spazio: $\Theta(1)$, $\cancel{\Theta(n)}$, $\cancel{\Theta(n^2)}$

$\underbrace{\boxed{O(1)},\ O(n),\ O(n^2),\ \ldots\ O(2^n) \ldots}$

Sebbene tutte corrette, quando esprimo la complessità con la notazione $O(\ )$, cerco di dare la stima migliore possibile, quella più piccola $O(1)$

DEFINIZIONE ALTERNATIVA DELLE NOTAZIONI O, Ω, Θ

Date due funzioni $C: \mathbb{N} \to \mathbb{R}$, ed $f: \mathbb{N} \to \mathbb{R}$, diciamo che

$C(n) \in O(f(n))$ quando esistono due costanti positive c ed m tali che

$$C(n) \leq c \cdot f(n) \text{ per tutti gli } n \geq m.$$

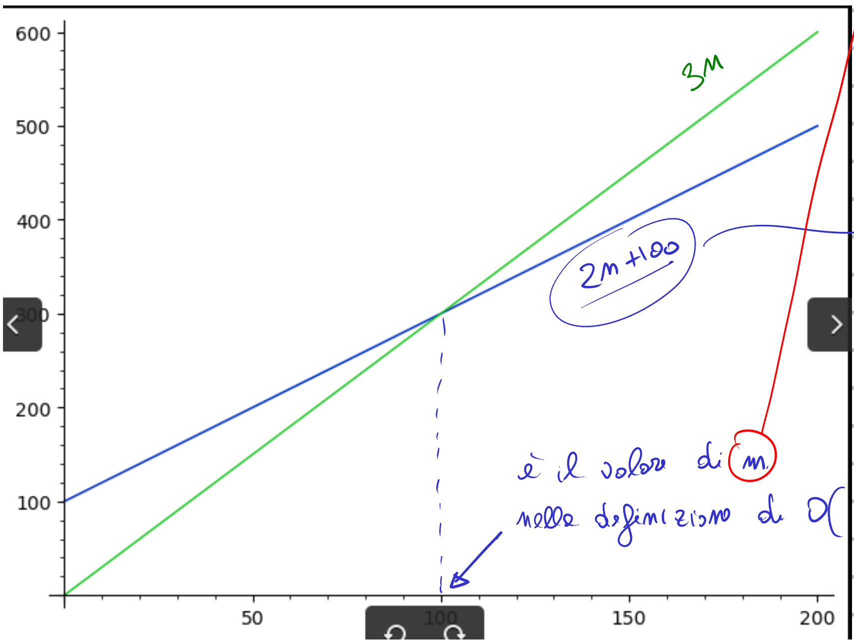
Esempio

Verifichiamo che $2n + 100 \in O(n)$

- Con la definizione data le volte scorse, calcoliamo $\lim_{n \to +\infty} \frac{2n+100}{n} = 2 \in \mathbb{R}$
- Con la nuova definizione dobbiamo invece far vedere che da un certo punto in poi la funzione $2n+100$ è $\leq$ di un qualche multiplo di n.

In altre parole, dobbiamo trovare c ed m che soddisfano la condizione di sopra.

Vi faccio vedere con un grafico che se prendo $c=3$, da un certo punto in poi $2n + 100 \leq 3n$.

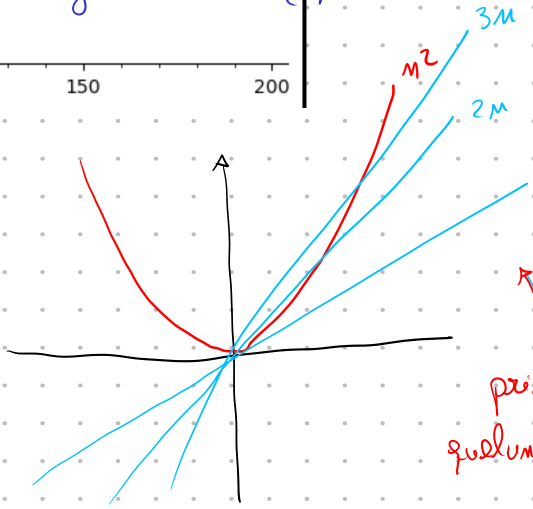


è il valore di m nella definizione di $O(\ )$

$2n+100 \in O(n)$ perché troviamo un multiplo di n che, da un certo punto in poi, è più grande della funzione $2n+100$

Esempio 2

$n^2 \notin O(n)$



Siccome la parabola prima o poi supera qualunque multiplo di n, È vero esattamente il contrario, non è vero che $n^2 \in O(n)$.

$n, 2n, 3n, 150.000\,n \in O(n^2)$

$C(n) \in \Omega(f(n))$ quando esistono due costanti positive c ed m tali che $C(n) \geq c\, f(n)$ per tutti gli $n \geq m$.

$C(n) \in \Theta(f(n))$ quando esistono tre costanti positive c_1, c_2 ed m tali che $c_1 f(n) \leq C(n) \leq c_2 f(n)$ per tutti gli $n \geq m$.

In linea di massima, le definizioni con i limiti e quelle date oggi coincidono, ma ci sono casi in cui con la definizione di oggi si può dimostrare che $C(n)$ ha una certa ordine di crescita, ma la stessa cosa non si riesce a fare nella definizione con i limiti.

Complessità in tempo MERGE SORT

Complessità di split è $O(n)$ dove n è la lunghezza del vettore e (quello da dividere)

Complessità di merge è $O(n)$ dove n è la lunghezza del vettore e (quello dove va il risultato)

Complessità di mergeSort?

```
int half = a.length / 2;
// Ceo due array destinati a contenere la prima e la seconda parte di a.
// pari, i due array hanno esattamente la stessa lunghezza (la metà di a.
// altrimenti il secondo avrà un elemento in più del primo (provare con a
// con a.length = 9 per convincersi).
int[] first = new int[half];
int[] second = new int[a.length - half];
// Usando il metodo split, divide a tra gli array first e second
split(a, first, second);
// Prima chiamata ricorsiva: ordina first
mergeSort(first);
// Seconda chiamata ricorsiva: ordina second
mergeSort(second);
// Ricopia gli array ordinati first e second in a, mantenendo l'ordine
merge(a, first, second);
```

$O(n)$

$O(n)$

??

$O(n)$

Se chiamiamo $T(n)$ il tempo che impiega il merge sort su un array e di lunghezza n:

$T(1) = c$

le somma del tempo perso da split, merge e creazione array

$T(n) =$ una cosa che è $O(n) + 2\,T(n/2)$

le dimensioni degli array first e second

perché deve ordinare sia first che second

$\leq dn + 2\,T(n/2)$

per definizione di $O(n)$, c'è questa costante d che però non sappiamo quanto vale, ma non ci interessa

Relazione di ricorrenza

$T(8) \leq d\,8 + 2 \cdot T(4)$

$\leq d \cdot 8 + 2 \cdot \left(d \cdot 4 + 2\,T(2) \right) = d \cdot 8 + d \cdot 8 + 4 \cdot T(2)$

$\leq d \cdot 8 + d \cdot 8 + 4\left(d \cdot 2 + 2\,T(1) \right) = d \cdot 8 + d \cdot 8 + d \cdot 8 + 8 \cdot T(1)$

$= d \cdot 8 + d \cdot 8 + d \cdot 8 + 8 \cdot c$

ho 3 somme

$T(n) =$

$\vdots$

$= d \cdot n + d \cdot n + d \cdot n + \dots + n \cdot c$

quante ne ho di queste somme? $\log_2 n$

$\leq d \cdot n \cdot \log_2 n + n \cdot c$ Qual è l'ordine di grandezza?

$\leq O(n \log_2 n)$

$\lim_{n \to +\infty} \frac{n \log n}{\cancel{n}} = \lim_{n \to +\infty} \log n = +\infty$