

## IL TEMPO

NOTA: In entrambi i casi, ALGORITMI PARALLELI E ALGORITMI DISTRIBUITI, la risorsa Tempo è cruciale

Come si misura il Tempo?

Ricordiamo il concetto come lo avete visto nel CAPO SEQUENZIALE

Definizione formale:

$T(x)$  = numero di operazioni elementari su  $x$   
( $x$  è l'istanza)

$$t(n) = \max \{ T(x) \mid x \in \Sigma^n \}$$

è una funzione in  $n$ , dove  $n$  è la lunghezza dell'input

## Osservazione

Spesso non saremo interessati ad una valutazione precisa di  $t(n)$ , ma al suo tasso di crescita



useremo le funzioni asintotiche  $\Theta, \Omega, \mathcal{H}$

## Definizione

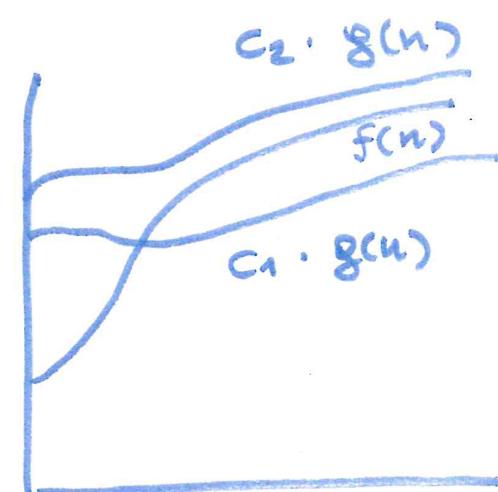
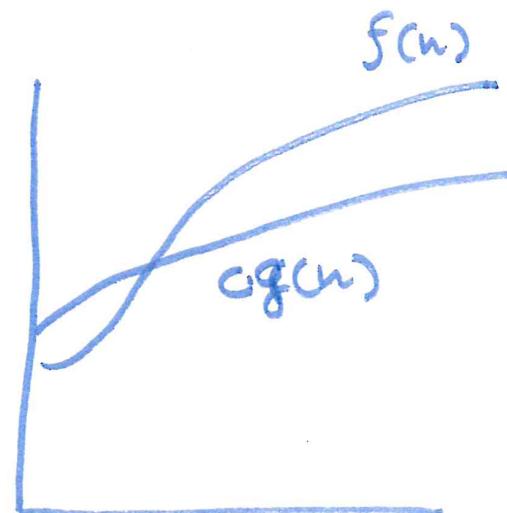
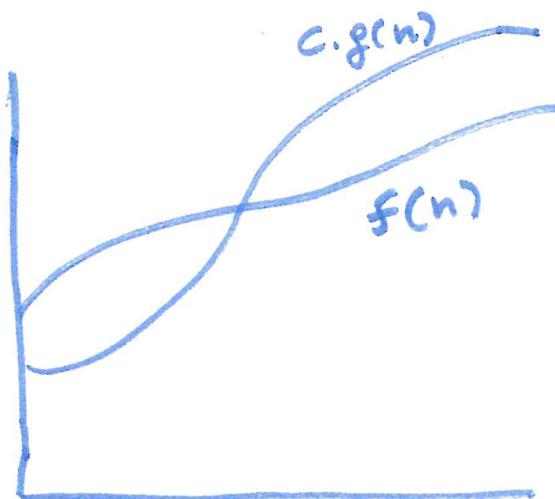
Siano  $f, g: \mathbb{N} \rightarrow \mathbb{N}$  due funzioni sui naturali. Diciamo che:

-  $f(n) = O(g(n))$  se  $\exists$  una costante  $c > 0$  e  $n_0 \in \mathbb{N}$   
t.c.  $f(n) \leq c \cdot g(n)$   $\forall n \geq n_0$ .

-  $f(n) = \Omega(g(n))$  se  $\exists$  una costante  $c > 0$  e  $n_0 \in \mathbb{N}$   
t.c.  $f(n) \geq c \cdot g(n)$   $\forall n \geq n_0$ .

-  $f(n) = \Theta(g(n))$  se esiste  $\exists$  due costanti  $c_1, c_2 > 0$  e  $n_0 \in \mathbb{N}$   
 t.c.  $c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$ .

Graficamente:



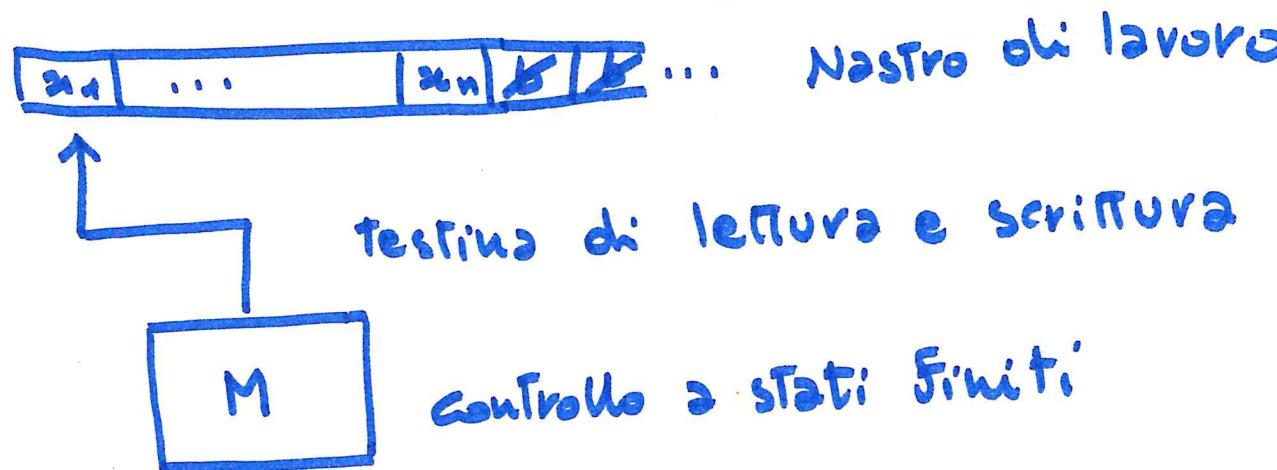
$$f(n) = O(g(n))$$

$$f(n) = \Omega(g(n))$$

$$f(n) = \Theta(g(n))$$

Nel seguito servirà il concetto di DTM (1936)

DTM = macchina di Turing



Def. formale:  $M = (Q, \Sigma, T, \delta, q_0, F)$

$Q$  = insieme finito di stati

$\Sigma$  = alfabeto di input

$T$  = alfabeto di lavoro,  $\Sigma \subset T$

$q_0 \in Q$  stato iniziale

$F \subseteq Q$  stati finali

$\delta: Q \times T \rightarrow Q \times T \times \{-1, 0, +1\}$

## Avvertenze sulla valutazione di $t(n)$

- ①  $t(n)$  è valutata su un particolare modello di calcolo
  - ② va scelto il criterio di costo: uniforme o logaritmico
- 

- ③ Le operazioni che contiamo sono le primitive messe a disposizione dal modello di calcolo

Ese.: PALINDROME

Input:  $x \in \{0,1\}^*$

Output:  $x$  è palindromo

Soluzione:

```
input ( $x \in \{0,1\}^*$ )
for ( $i=1$ ,  $j=|x|$ ;  $i < j \wedge x_i = x_j$ ;  $i++$ ,  $j--$ );
    return  $i \geq j$ ;
```

Tale programma: su RAM (memoria ad accesso casuale)  
ha  $T(n) = O(n)$

su DTM (abbiamo una sola Testina di lettura)  
ha  $T(n) = O(n^2)$

② Attenzione alla dimensione dei dati in gioco!

Ese.: FATTORIALE

Input:  $n \in \mathbb{N}$

Output:  $n!$

Soluzione: input ( $x \in \{0,1\}^*$ )

$k = 1;$

for ( $i = 1; i \leq x; k = k * i, i++$ );

return  $k$ ;

- Tale programma :
- considerando la moltiplicazione come operazione elementare su una RAM
- $$t(\log n) \leq n$$
- su DTM, doveroso scrivere in binario il risultato  $n! \approx n^n$  abbiamo
- $$t(\log n) \geq \log n^n = n \log n$$
- criterio di costo UNIFORME  
le operazioni elementari richiedono una unità di tempo
- criterio di costo LOGARITMICO:  
ogni operazione elementare ha un costo che dipende dal numero di bit degli operandi

# TEORIA DELLA COMPLESSITÀ

possibili Tassi di crescita di  $t(n)$

$t(n)$  si dirà:

- logaritmico quando è  $O(\log n)$
- polilogaritmico quando è  $O(\log^k n)$  per una cost.  $k > 0$
- lineare quando è  $O(n)$
- polinomiale quando è  $O(n^k)$  per una cost.  $k > 0$
- esponenziale quando NON è  $O(n^k)$  per ogni cost.  $k > 0$

## CONCETTO DI EFFICIENZA (CASO SEQUENZIALE)

### Definizione

Un problema è risolto efficientemente in tempo  $\text{P} \leq$   
è risolto da una DTM in tempo polinomiale -  
 $P =$  classe dei problemi di decisione risolti efficienti in tempo

$FP =$  " " " generali " "

oppure  
 $P =$  classe dei problemi di decisione risolti in tempo polinomiale  
 $FP =$  " " " generali " "

Tempo polinomiale = efficienza  
 $\Rightarrow$  perché?

Rispondiamo col seguente esempio:

Consideriamo un processore da 4 GHz ( $\neq$  da una DTM) su una istanza di 4'000 caratteri (metà pagina)

<u>complessità in tempo: <math>t(n)</math></u>	<u>tempo di attesa per l'output</u>
$n$	1 μs (micro = $10^{-6}$ )
$n^2$	4 ms (milli = $10^{-3}$ )
$2^n$	> 1 secolo
$3^n$	~ 2 secoli e mezzo

NOTA: Tempi esponenziali esplodono anche su istanze piccole

Quindi P e FP hanno una definizione robusta  
che non cambia con il modello di calcolo

Inoltre P e FP contengono molti problemi  
fondamentali: ordinamento, prodotto di matrici,  
alcuni problemi di ottimizzazione, ...

Purtroppo altri problemi interessanti non hanno  
ancora soluzioni efficienti e stanno in

NP = problemi di decisione risolti in tempo  
polinomiale su una Macchina di Turing  
non deterministica

Si congettura  $P \neq NP$ , ma non è ancora noto!

## OSSERVAZIONE

All'atto pratico siamo interessati ad una sottoclasse di P

- ① La valutazione asintotica dei tempi può nascondere costanti o termini che all'atto pratico fanno la differenza

es.: a merge-sort che lavora in tempo  $O(n \log n)$   
a volte si preferisce quick-sort ( $O(n^2)$ ).

- ② Da un punto di vista pratico, il grado del polinomio deve essere basso

es.:  $n^{1000}$  su piccole istanze risulta peggiore  
di un algoritmo esponenziale

-  $O(n^6)$  temp per il test di primalità ottenuto nel 2002  
il grado è troppo alto e si preferisce un algoritmo probabilistico con possibilità di errore

# ALGORITMI PARALLELI

Problematiche :

- 1) SINTESI
  - 2) VALUTAZIONE
  - 3) UNIVERSALITÀ
- 

① SINTESI :

Problema di sintesi : Come costruire algoritmi paralleli ?

Ci possiamo ispirare a buoni algoritmi sequenziali ?

Risposta : Ispirarsi sì ! Ma riciclare l'intero algoritmo quasi mai !

Esempio 1: sommare due numeri binari

Algoritmo sequenziale ottimo: somma i bit per colonna



Mi porta ad un buon algoritmo parallelo?

$$\begin{array}{r} 1001 \\ + 111 \\ \hline \dots \dots 1 \end{array}$$

ogni processore si può occupare  
di una colonna.

Problema?

Si! C'è il risotto che c'è tornare  
ad un algoritmo sequenziale  
con in più uno spreco di hardware  
ingiustificato!!

MA invece ...

Esistono algoritmi paralleli efficienti per sommare  
due numeri binari

Esempio 2: ordinamento

Algoritmo sequenziale ottimo: merge-sort



Mi porta ad un buon algoritmo parallelo?

No, MA invece esistono alg. paralleli efficienti  
per l'ordinamento.

« Per la sintesi di algoritmi paralleli servono  
idee nuove, tecniche e punti di vista totalmente  
diverse da quelle usate per il sequenziale »

## ② VALUTAZIONE DELLE PRESTAZIONI

- Problema della valutazione: come misurate le prestazioni degli algoritmi paralleli?
  - Tempo
  - hardware = n° di processori
- Problema della valutazione: cos'è efficiente nel caso parallelo?
  - parametro  $E = \text{efficienza}$   
Tiene conto sia del tempo, sia del numero di processori e deve dire se "il gioco vale la candela"!

### ③ UNIVERSALITÀ

Problematica teorica: riesco a caratterizzare la classe dei problemi che ammettono algoritmi paralleli efficienti?

Definizione nel caso sequenziale: FP

Definizione nel caso parallelo: NC

NC = problemi risolti da algoritmi paralleli veloci  
con - tempo polilogaritmico  
- hardware polinomiale

Esempi di problemi in NC:  
- somma di due numeri binari  
- ordinamento

Attenzione: In realtà per noi è ragionevole una sottoclasse di NC

Come per P, se il grado del polinomio o le costanti nasconde sono alte  $\Rightarrow$  il tempo è sì polinomiale ma poco pratico

Fatto:  $NC \subseteq FP$

olim. Per ottenere un algoritmo sequenziale partendo da quello parallelo, basta simulare "con un processore" in sequenza il lavoro dei processori coinvolti. Si ottiene un algoritmo sequenziale polinomiale in tempo.

- $NC = FP$  ?      Problema aperto! Si pensa di no!
  - La questione  $NC \subseteq FP$  ripropone il problema: "ogni algoritmo sequenziale efficiente è parallelizzabile?"
  - Se  $NC \neq FP \Rightarrow FP \setminus NC =$  problemi P-completi  
cioè quei problemi tali che la loro appartenenza a  $NC$  fa collassare  $NC$  e  $FP$  (ovvero  $NC = FP$ )  
"Pertanto problemi che non riusciremo a parallelizzare sono i problemi P-completi -"

Noi non ci occuperemo di UNIVERSALITÀ  
 E' difficile e rimane uno studio per esperti teorici

Riassumiamo le tappe nel modello parallelo:

- Fissare un modello di calcolo Teorico  
che "poi" si potrà mappare su architetture reali
- Fissare le risorse computazionali:  
tempo, n° di processori
- Fissare il "parametro" di efficienza  
(ci dice dove se la nostra soluzione parallela  
è preferibile ad una efficiente soluzione sequenziale)