Merge Insertion Sort

$$T(n,k)=\begin{cases} O(n^2) & n \le k \\ 2T\left(\frac{n}{2}\right)+\Theta(n) & n > k \end{cases}$$

$$T(n,k)=\Theta\left(\underbrace{n \cdot k}_{\text{Costo dell'I.S.}} + \underbrace{n \cdot \log \frac{n}{k}}_{\text{Costo del Merge}}\right)$$

si ~~esegue~~ esegue I.S. $\frac{n}{k}$ volte, su segmenti lunghi k

Costo: $\frac{n}{k} \cdot O(k^2) = O(n \cdot k)$

fusioni di sequenze

$k \to 2k \to 4k \to \dots \underbrace{2^i}\, k = n$

$i = \log_2 \frac{n}{k}$

1) # livelli dell'albero di ricorsione
2) ogni livello "costa" $\Theta(n)$

Titolo nota 26/02/2016

3) massimo valore asintotico di k t.c. MergeInsertionSort ha lo stesso tempo di esecuzione di Merge Sort

$$T_{MIS}(n,k) = \Theta\left(nk + n\log\frac{n}{k}\right) \qquad T_{MS}(n) = \Theta(n\log n)$$

$$\Downarrow$$

$$k = O(\log n)$$

proviamo con $k = \Theta(\log n)$

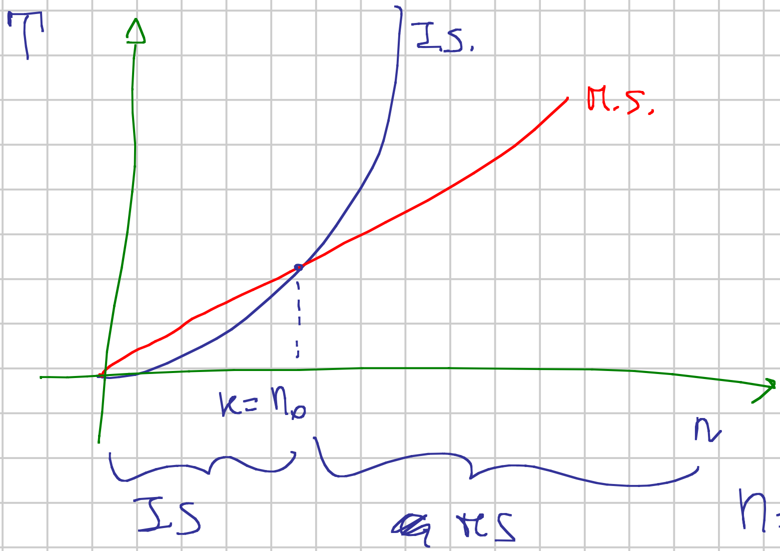
$$T_{MIS}(n, k=\log n) = \Theta\left(n \cdot \log n + n \log\frac{n}{\log n}\right) =$$

$$= \Theta\left(n\log n + n\log n - n\log(\log n)\right) = \Theta(n\log n)$$

4) Come scegliere k in pratica?

k → dim. massima delle ~~seq~~ sequenze su cui I.S. è più veloce del M.S.

$$T_{IS}(n) \overset{?}{\leq} T_{MS}(n)$$



$$\boxed{T_{IS}(n) = 2 \cdot n^2 \qquad T_{MS}(n) = 64\, n \log n}$$

$$\cancel{2} n^{\cancel{2}} \leq \cancel{64}^{\,32}\, \cancel{n} \log n$$

$$n = 32 \log n$$

$n = 2^7$ $\quad 2^7 \overset{?}{=} 32 \cdot 7$ $\quad 2^7 < 32 \cdot 7$ $\quad \underline{\underline{IS}}$

$n = 2^8$ $\quad 2^8 \overset{?}{\leq} 32 \cdot 8 = 2^8$ $\quad$ IS e MS hanno pari costo

$n_0 = 2^8$

$n = 257 = 2 n_0 + 1$

$257 \quad \frac{ol}{2} \quad 32 \cdot \log 257$

$\cancel{4}$

Limiti inferiori di complessità

Problema Π

$L(n)$ è un limite inferiore per il problema Π

se ogni algoritmo A che risolve Π

$$T_A(n) = \Omega(L(n))$$ nel caso peggiore

$L(n)$ operazioni sono NECESSARIE per risolvere Π nel caso peggiore

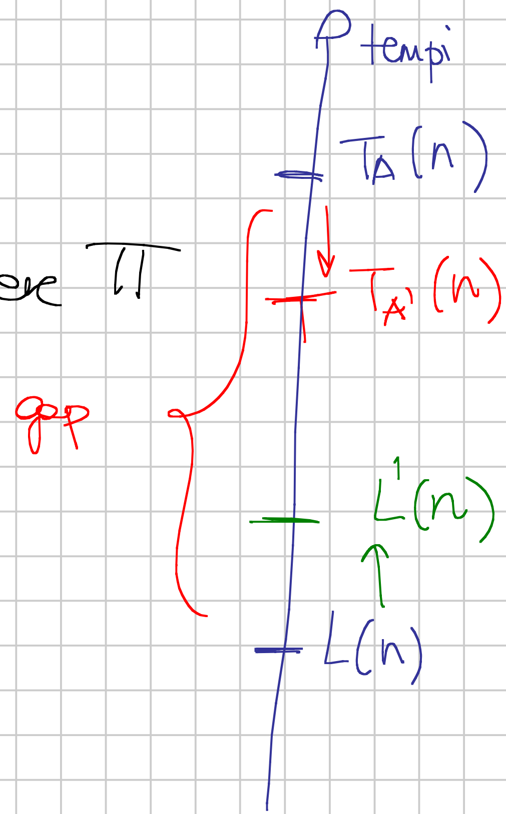
A algoritmo che risolve Π

$T_A(n)$ è un limite superiore alla complessità

$T_A(n)$ operazioni sono sufficienti per risolvere Π

A è un algoritmo ottimo per Π se

$$T_A(n) = \Theta(L(n))$$



1 TECNICA della "DIMENSIONE dell'INPUT"

Se un problema Π richiede l'esame di tutti i dati di input

$$\Rightarrow \quad L_{\Pi}(n) = \Omega(n) \qquad n = \text{dim. dell'input}$$

Π: ricerca del minimo $\quad L_{\Pi}(n) = \Omega(n)$ significativo

Π: Somma di 2 matrici $n \times n$

dim. input $= \Theta(n^2)$

$$\begin{pmatrix} 1 & 2 \\ 4 & 6 \end{pmatrix} + \begin{pmatrix} 2 & 8 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 10 \\ 6 & 7 \end{pmatrix}$$

A $\qquad$ B $\qquad$ C

2. tecnica dell'albero di decisione

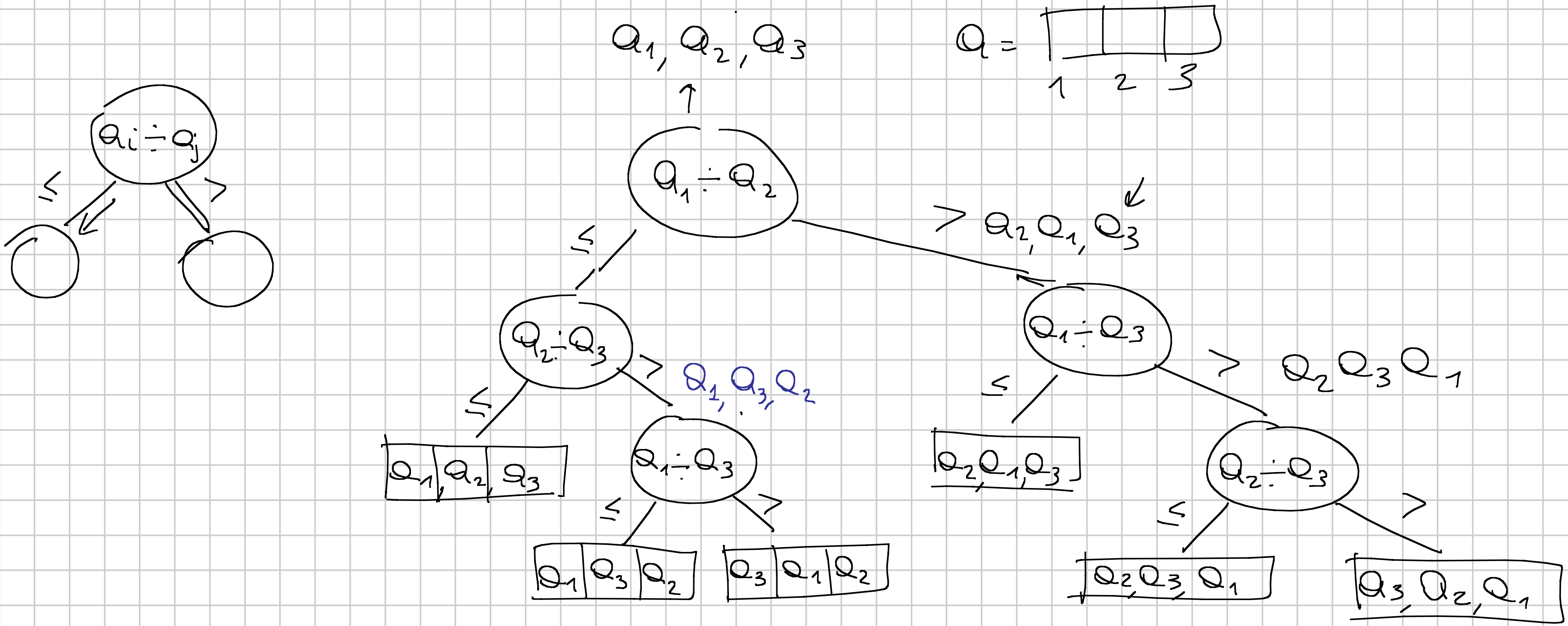
si applica ai problemi che si possono risolvere per mezzo di una sequenza di "decisioni" o "confronti"

AdD → struttura matematica che si usa per rappresentare algoritmi che risolvono un problema attraverso confronti

costo in tempo e # confronti effettuati sono dello stesso ordine di grandezza

ESEMPIO

AdD che rappresenta InsertionSort per $n = 3$



AdD

nodi interni ⟶ confronti

foglie ⟶ soluzioni

cammini radice → foglie : esecuzioni dell'algoritmo su una particolare istanza di input

lunghezza cammino radice → foglia ⟶ # confronti effettuati dall'algoritmo

⟶ stima del costo in tempo

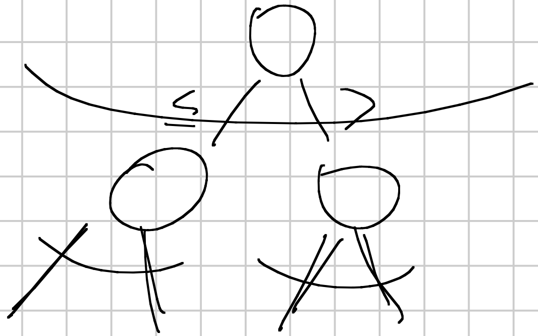
Altezza : lunghezza del percorso più lungo radice → foglie

↳ # confronti al caso pessimo

(1) Altezza AdD che rappresenta l'algoritmo A ⟷ costo in tempo al caso pessimo di A $T_A(n)$

(2) # foglie dell'AdD ≥ # soluzioni del problema Π
∀ AdD relativo al problema Π

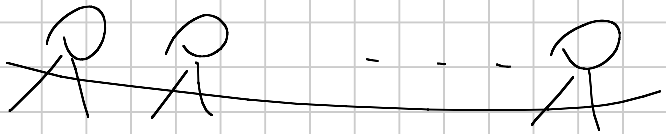
$\Rightarrow$ Un limite inferiore per l'altezza di un generico AdD relativo a un problema Π equivale a un limite inferiore per il # di confronti <u>necessari</u> per risolvere il problema Π



1 confronto : 2 casi

2 confronti : 4 casi

i confronti :



i confronti $\Rightarrow 2^i$ casi

Con i confronti, un generico algoritmo può discriminare al più 2^i casi

x essere corretto, i deve essere t.c. $2^i \geq \underbrace{n!}_{\#\text{soluzioni}} \Rightarrow i \geq \log n!$

Teorema Qualsiasi algoritmo di ordinamento per confronti richiede $\Omega(n \log n)$ confronti al caso pessimo.

Dim lo dimostriamo cercando un limite inferiore per l'altezza di un AdD dove ogni permutazione compare in una foglia

(**ES** $\lceil a_3, a_2, a_1$ mai presente nelle foglie

$\Rightarrow$ l'algoritmo associato all'albero è errato ad esempio sull'input $\lceil 5, 4, 3$)

T: generico AdD con f foglie

1) $\boxed{f \geq n!}$

2) AdD è un albero binario completo (ogni nodo interno ha esattamente due figli)

Proprietà Un albero binario completo di altezza h contiene al più 2^h foglie.

Dim (per induzione sull'altezza h)

BASE $\boxed{h=0}$ # foglie $= 1 = 2^0 = 2^h$

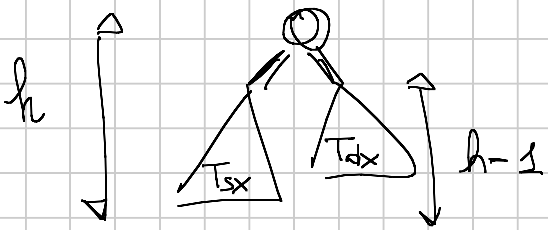
ipotesi induttiva:

$\forall$ ABC di altezza $k \leq h-1$

$\#$ foglie $\leq 2^k$

Passo

T sia un ABC di altezza h



$f = \#$ foglie di T

$f_{sx} = \#$ foglie di T_{sx}, $f_{dx} = \#$ foglie T_{dx}

altezza massima dei sottoalberi è $h-1$

$$f = f_{sx} + f_{dx} \underset{i.i}{\leq} 2^{h-1} + 2^{h-1} = 2^h \quad \square$$

$$\boxed{\begin{array}{l} f \geq n! \\ f \leq 2^h \end{array}}$$

h = # confronti al caso peggiore

$$2^h \geq f \geq n!$$

$$\Rightarrow \quad 2^h \geq n! \quad \Rightarrow \quad \boxed{h \geq \log n!}$$

Lim. inferiore

$$n! = \underbrace{n \cdot (n-1) \cdot (n-2) \cdots \frac{n}{2}}_{\geq \frac{n}{2}} \cdot \underbrace{\left(\frac{n}{2}-1\right) \cdots 3 \cdot 2 \cdot 1}_{\geq 1} > \left(\frac{n}{2}\right)^{n/2} (1)^{n/2} = \left(\frac{n}{2}\right)^{n/2}$$

$$n! > \left(\frac{n}{2}\right)^{n/2}$$

$$L(n) = h \geq \log n! > \log\left(\frac{n}{2}\right)^{n/2} = \frac{n}{2}\log\frac{n}{2} = \Omega(n\log n)$$

$\square$

$\Rightarrow$ MergeSort è ottimo in tempo.