

ALGORITMO : PROCEDURA COMPUTAZIONALE BEN
DEFINITA CHE PRODUCE UNO O PIÙ VALORI
(OUTPUT) IN FUNZIONE DI UNO O PIÙ ALTRI
VALORI (INPUT) PER RISOLVERE PROBLEMI
COMPUTAZIONALI



PROBLEMA COMPUTAZIONALE : SPECIFICATO DA UNA
DATA RELAZIONE TRA INPUT/OUTPUT

ESEMPIO: PROBLEMA DELL'ORDINAMENTO (SORTING)

INPUT: SEQUENZA (a_1, a_2, \dots, a_n) DI n NUMERI

OUTPUT: PERMUTAZIONE $(a_{i_1}, a_{i_2}, \dots, a_{i_n})$ DI (a_1, a_2, \dots, a_n)

TALE CHE $a_{i_1} \leq a_{i_2} \leq \dots \leq a_{i_n}$
(ORDINAMENTO NON-DECRESCENTE)

- OGNI SPECIFICO INPUT (ES. $(4, 5, 1, 2, 4, 3)$)
SI DIRA' ISTANZA DEL PROBLEMA

ALTRI ESEMPI DI PROBLEMI COMPUTAZIONALI:

- PROGETTO HUMAN GENOME
- ACCESSO VELOCE ALL'INFORMAZIONE (WEB, DATABASE)
- COMMERCIO ELETTRONICO (CRITTOGRAFIA, FIRME DIGITALI)
- PROBLEMI DI OTTIMIZZAZIONE (PRODUZIONE, TRASPORTO)
- ECC.

ALGORITMI COME TECNOLOGIA (AL PARI DEL HARDWARE)

- SI CONSIDERINO DUE ALGORITMI a_1 ED a_2 PER UN MEDESIMO PROBLEMA COMPUTAZIONALE
(ES. a_1 - INSERTION SORT, a_2 - MERGESORT)

- SUPPONIAMO CHE $T_{a_1}(n) = c_1 n^2$
 $T_{a_2}(n) = c_2 n \lg n$

- PER VALORI SUFFICIENTEMENTE GRANDI DI n
SI HA: $T_{a_2}(n) < T_{a_1}(n)$, DATO CHE

$$\lim_{n \rightarrow \infty} \frac{c_2 n \lg n}{c_1 n^2} = 0$$

- CONSIDERIAMO DUE COMPUTER C_1 (VELOCE) E C_2 (LENTO)

VELOCITA'

C_1	10^9 ISTRUZ/SEC
C_2	10^7 ISTRUZ/SEC

- SUPPONIAMO DI AVERE DELLE IMPLEMENTAZIONI DI

a_1 SU C_1 E DI a_2 SU C_2 TALI CHE

$$T_{a_1}(n) = 2n^2 \quad \text{E}$$

$$T_{a_2}(n) = 50n \lg n$$

- CONFRONTIAMO I TEMPI DI ESECUZIONE DI a_1
SU C_1 E DI a_2 SU C_2 SU UN INPUT DI
DIMENSIONE $n = 10^6$

COMPUTER C_1 :
$$\frac{2 \cdot (10^6)^2 \text{ ISTRUZ}}{10^9 \text{ ISTRUZ/SEC}} = 2000 \text{ SEC}$$

COMPUTER C_2 :
$$\frac{50 \cdot 10^6 \cdot \lg 10^6 \text{ ISTRUZ}}{10^7 \text{ ISTRUZ/SEC}} \simeq 100 \text{ SEC}$$

SEBBENE C_1 SIA 100 VOLTE PIÙ VELOCE DI C_2 ,
L'ESECUZIONE DI a_1 (DA PARTE DI C_1) RISULTA
20 VOLTE PIÙ LENTA DI QUELLA DI a_2 (DA PARTE
DI C_2).

1.2-2

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n , insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

$$8n^2 < 64n \lg n$$

$$\frac{n}{\lg n} < 8 \quad \longrightarrow \quad n \leq 43$$

1.2-3

What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

$$100n^2 < 2^n \quad \longrightarrow \quad n \geq 15$$

1-1 Comparison of running times

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t , assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
\sqrt{n}							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

UN PRIMO CASO DI STUDIO: **INSERTION SORT**
(CORRETTEZZA E COMPLESSITA')

STABILE

INPUT: UN ARRAY $A[1..m]$ DI INTERI

OUTPUT: UNA PERMUTAZIONE DI A ORDINATA IN SENSO NON-DECRESCENTE

INSERTION SORT(A)

for $j := 2$ to $\text{length}[A]$

do $\text{key} := A[j]$

{ INSERISCE $A[j]$ NELLA SEQUENZA ORDINATA $A[1..j-1]$ }

$i := j - 1$

while $i > 0$ and $A[i] > \text{key}$

do $A[i+1] := A[i]$

$i := i - 1$

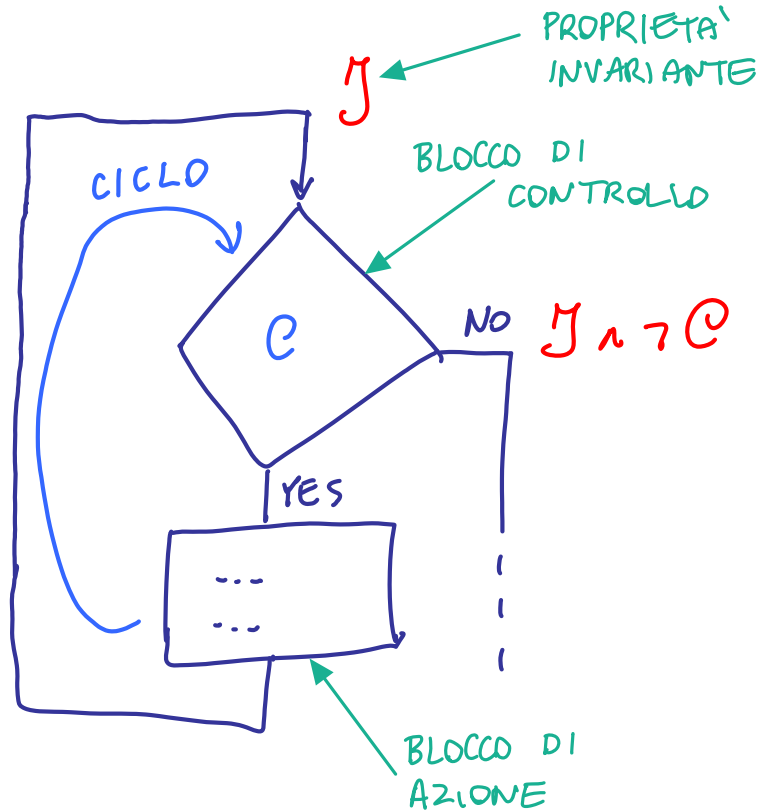
$A[i+1] := \text{key}$

B	D	F	H	E	A	C	G
2	4	4	4	7	8	8	9

--	--	--	--	--	--	--	--

CORRETTEZZA DI INSERTION SORT

- SI UTILIZZA LA TECNICA DELLE (PROPRIETA') INVARIANTI
DI CICLO



1. INIZIALIZZAZIONE
 \mathcal{I} E' VERA PRIMA DELLA PRIMA ITERAZIONE
2. MANTENIMENTO
SE \mathcal{I} E' VERA PRIMA DELLA ESECUZIONE DI UNA ITERAZIONE DEL CICLO, RIMANE VERA PRIMA DELLA SUCCESSIVA ESECUZIONE
3. CONCLUSIONE
QUANDO IL CICLO TERMINA VALE $\mathcal{I} \wedge \neg C$

CORRETTEZZA DEL CICLO-FOR

- SUPPORREMO CHE IL CORPO DEL ^(CICLO-FOR) SIA CORRETTO, CIOE' CHE INSERISCA CORRETTAMENTE L'ELEMENTO $A[j]$ NEL SOTTOARRAY (ORDINATO) $A[1..j-1]$

INSERTION SORT(A)

for $j := 2$ to $\text{length}[A]$

do $\text{key} := A[j]$

$i := j-1$

while $i > 0$ and $A[i] > \text{key}$

do $A[i+1] := A[i]$

$i := i-1$

$A[i+1] := \text{key}$

$j \equiv$ IL SOTTOARRAY $A[1..j-1]$ E' FORMATO DAGLI ELEMENTI ORDINATI IN $A^{(0)}[1..j-1]$
& $1 \leq j-1 \leq \text{length}[A]$

$C \equiv ??$

NOTA: $A^{(0)}$ INDICA L'ARRAY A NELLE SUE CONDIZIONI INIZIALI

CORRETTEZZA DEL CICLO-FOR

- SUPPORREMO CHE IL CORPO DEL ^(CICLO-FOR) SIA CORRETTO, CIOE' CHE INSERISCA CORRETTAMENTE L'ELEMENTO $A[j]$ NEL SOTTOARRAY (ORDINATO) $A[1..j-1]$

INSERTION SORT(A)

for $j := 2$ to $\text{length}[A]$

do $\text{key} := A[j]$

$i := j-1$

while $i > 0$ and $A[i] > \text{key}$

do $A[i+1] := A[i]$

$i := i-1$

$A[i+1] := \text{key}$

$j \equiv$ IL SOTTOARRAY $A[1..j-1]$ E' FORMATO DAGLI ELEMENTI ORDINATI IN $A^{(0)}[1..j-1]$
& $1 \leq j-1 \leq \text{length}[A]$

$C \equiv j \leq \text{length}[A]$

INIZIALIZZAZIONE ($j=2$)

$j \equiv$ IL SOTTOARRAY $A[1..1]$ E' FORMATO DAGLI ELEMENTI
ORDINATI IN $A^{(0)}[1..1]$

• BANALMENTE VERO !!

INSERTION SORT(A)

for $j := 2$ to $\text{length}[A]$

do $\text{key} := A[j]$

$i := j-1$

while $i > 0$ and $A[i] > \text{key}$

do $A[i+1] := A[i]$

$i := i-1$

$A[i+1] := \text{key}$

$j \equiv$ IL SOTTOARRAY $A[1..j-1]$ E'
FORMATO DAGLI ELEMENTI
ORDINATI IN $A^{(0)}[1..j-1]$
& $1 \leq j-1 \leq \text{length}[A]$

$C \equiv j \leq \text{length}[A]$

MANTENIMENTO ($j \leq \text{length}[A]$)

SE IL SOTTOARRAY $A[1..j-1]$ E' FORMATO DAGLI ELEMENTI ORDINATI DI $A^{(0)}[1..j-1]$, DOPO L'ESECUZIONE DEL CORPO DEL CICLO-FOR $A[j]$ E' INSERITO CORRETTAMENTE IN $A[1..j-1]$ E DUNQUE $A[1..j]$ E' FORMATO DAGLI ELEMENTI ORDINATI DI $A^{(0)}[1..j]$

INSERTION SORT(A)

for $j := 2$ to $\text{length}[A]$

do $\text{key} := A[j]$

$i := j-1$

while $i > 0$ and $A[i] > \text{key}$

do $A[i+1] := A[i]$

$i := i-1$

$A[i+1] := \text{key}$

$j \equiv$ IL SOTTOARRAY $A[1..j-1]$ E' FORMATO DAGLI ELEMENTI ORDINATI IN $A^{(0)}[1..j-1]$
& $1 \leq j-1 \leq \text{len } A$

$C \equiv j \leq \text{length}[A]$

CONCLUSIONE

($j > \text{length}[A]$)

A CONCLUSIONE DELL'ESECUZIONE DEL CICLO-FOR, VALE $j \wedge \neg C$.
DUNQUE $1 \leq j-1 \leq \text{length}[A]$ E $j > \text{length}[A]$, DA CUI $j = \text{length}[A] + 1$.
PERTANTO: IL SOTTOARRAY $A[1.. \text{length}[A] + 1 - 1] \equiv A$ E' FORMATO DAGLI
ELEMENTI ORDINATI IN $A^{(0)}$!

INSERTION SORT(A)

for $j := 2$ to $\text{length}[A]$

do $\text{key} := A[j]$

$i := j - 1$

while $i > 0$ and $A[i] > \text{key}$

do $A[i+1] := A[i]$

$i := i - 1$

$A[i+1] := \text{key}$

$J \equiv$ IL SOTTOARRAY $A[1.. j-1]$ E'
FORMATO DAGLI ELEMENTI
ORDINATI IN $A^{(0)}[1.. j-1]$
& $1 \leq j-1 \leq \text{length}[A]$

$C \equiv j \leq \text{length}[A]$

ANALISI DI COMPLESSITA' DEGLI ALGORITMI

- MISURA DELLE RISORSE RICHIESTE DALL'ESECUZIONE DI UN ALGORITMO, QUALI
 - TEMPO DI ELABORAZIONE
 - MEMORIA
 - LARGHEZZA DI BANDA NELLE COMUNICAZIONI
 - HARDWARE
- FAREMO RIFERIMENTO AL MODELLO DI CALCOLO A UN PROCESSORE RANDOM-ACCESS MACHINE (RAM) IN CUI LE ISTRUZIONI SONO ESEGUITE UNA ALLA VOLTA
- PER UN DATO ALGORITMO, SI CERCA UNA RELAZIONE TRA LA DIMENSIONE DELL'INPUT E IL TEMPO DI ELABORAZIONE.

DIMENSIONE DELL'INPUT

- PUO' ESSERE:
- NUMERO DEGLI ELEMENTI DELL'INPUT
 - NUMERO DI BIT PER RAPPRESENTARE L'INPUT
 - COPPIA DI NUMERI (ES. GRAFI)

TEMPO DI ELABORAZIONE

- NUMERO DI OPERAZIONI PRIMITIVE ESEGUITE
- FAREMO L'IPOTESI CHE OGNI ISTRUZIONE COINVOLGA UN CERTO NUMERO DI OPERAZIONI PRIMITIVE E CHE QUINDI ABBA COSTO COSTANTE

INSERTION SORT(A)

$$1 \leq t_j \leq j$$

COSTO

ESECUZIONI

```

1. for j := 2 to length[A]
2.   do key := A[j]
3.     { INSERTSCE A[j] IN A[1..j-1] }
4.     i := j-1
5.     while i > 0 and A[i] > key
6.       do A[i+1] := A[i]
7.         i := i-1
8.     A[i+1] := key
  
```

C_1

n

C_2

$n-1$

O

$n-1$

C_4

$n-1$

C_5

$$\sum_{j=2}^n t_j$$

C_6

$$\sum_{j=2}^n (t_j - 1)$$

C_7

$$\sum_{j=2}^n (t_j - 1)$$

C_8

$n-1$

DOVE $n := \text{length}[A]$, $t_j := \#$ DI ESECUZIONI DEL TEST DEL CICLO-WHILE 5-7 PER IL VALORE j

$$T(n) = C_1 n + (C_2 + C_4 + C_8) \cdot (n-1) + C_5 \sum_{j=2}^n t_j + (C_6 + C_7) \sum_{j=2}^n (t_j - 1)$$

- SI OSSERVI CHE ANCHE PER INPUT DI UNA STESSA DIMENSIONE n ,
I VALORI t_j DIPENDONO DAL PARTICOLARE INPUT

$$1 \leq t_j \leq j$$

$$\text{CASO } t_j = j \quad \left(\text{USEREMO } \sum_{j=1}^m j = \frac{n(n+1)}{2} \right)$$

$$T(n) = c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \sum_{j=2}^n j + (c_6 + c_7) \sum_{j=2}^n (j-1)$$

$$= c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \left(\sum_{j=1}^n j - 1 \right) + (c_6 + c_7) \sum_{j=1}^{n-1} j$$

$$= c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) + (c_6 + c_7) \frac{(n-1)n}{2}$$

$$= c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \left(\frac{n^2}{2} + \frac{n}{2} - 1 \right) + (c_6 + c_7) \left(\frac{n^2}{2} - \frac{n}{2} \right)$$

$$= \frac{1}{2}(c_5 + c_6 + c_7)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right)n - (c_2 + c_4 + c_5 + c_8)$$

$$A n^2 + B n + C$$

$$A \neq 0$$

CASO MIGLIORE (BEST CASE ANALYSIS) POCO INTERESSANTE

- SI HA QUANDO L'INPUT E' GIA' ORDINATO (IN SENSO NON-DECRESCENTE)
- IN QUESTO CASO $t_j = 1$, PER $j = 2, 3, \dots, n$, E QUINDI

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

CIOE'

$$T(n) = An + B, \quad \text{CON } A \text{ E } B \text{ COSTANTI}$$

- DUNQUE IN QUESTO CASO $T(n)$ E' LINEARE.

CASO PEGGIORE (WORST CASE ANALYSIS)

- SI HA QUANDO L'INPUT E' GIA' ORDINATO IN SENSO DECRESCENTE
- IN QUESTO CASO $t_j = j$, PER $j = 2, 3, \dots, n$, E QUINDI, FACENDO USO DELL'IDENTITA'

$$\sum_{j=1}^n j = \frac{n(n+1)}{2}$$

SI HA:

$$T(n) = \frac{1}{2}(c_5 + c_6 + c_7)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

CIOE'

$$T(n) = An^2 + Bn + C, \quad \text{CON } A, B \text{ E } C \text{ COSTANTI}$$

- IN QUESTO CASO $T(n)$ E' QUADRATICO

- IN GENERALE CI LIMITEREMO A DETERMINARE IL TEMPO DI ESECUZIONE NEL CASO PEGGIORE, IN QUANTO:
 - RAPPRESENTA UN LIMITE SUPERIORE AL TEMPO DI ESECUZIONE PER QUALSIASI INPUT
 - IN MOLTI CASI IL CASO PEGGIORE SI VERIFICA SPESSO
 - IL CASO MEDIO SPESSO E' ALTRETTANTO CATTIVO QUANTO QUELLO PEGGIORE (ES. $t_j \approx j/2$ IN MEDIA)
- L'ANALISI NEL CASO MEDIO (AVERAGE CASE ANALYSIS) RICHIEDE TECNICHE DI ANALISI PROBABILISTICA

IN GENERALE, SAREMO INTERESSATI ALL'ORDINE DI CRESCITA DELLA FUNZIONE TEMPO DI ESECUZIONE.

QUINDI, NON SOLO TRASCUREREMO, NEL CASO DI INSERTION SORT, I VALORI DELLE COSTANTI C_1, C_2, \dots, C_8 , MA SEMPLIFICHEREMO ULTERIORMENTE LE ESPRESSIONI $An + B$ E $An^2 + Bn + C$ TRASCURANDO I TERMINI DI ORDINE INFERIORE E LA COSTANTE MOLTIPLICATIVA DEI TERMINI DI ORDINE SUPERIORE, E DUNQUE DIREMO CHE LA COMPLESSITA' DI INSERTION SORT E'

$\Theta(n)$, NEL CASO MIGLIORE

$\Theta(n^2)$, NEL CASO PEGGIORE E NEL CASO MEDIO

- INSERTION SORT E' BASATO SULL'APPROCCIO INCREMENTALE:
L'INPUT A VIENE INFATTI ORDINATO IN MANIERA INCREMENTALE:
 $A[1..2]$, $A[1..3]$, ..., $A[1..n-1]$, $A[1..n]$
- UN APPROCCIO MOLTO IMPORTANTE PER LA PROGETTAZIONE
DI ALGORITMI RICORSIVI E' IL DIVIDE ET IMPERA

FIN QUI

L'APPROCCIO DIVIDE ET IMPERA

IL PARADIGMA DIVIDE ET IMPERA PREVEDE TRE PASSI A OGNI LIVELLO DI RICORSIONE:

DIVIDE : IL PROBLEMA VIENE SUDDIVISO IN UN CERTO NUMERO DI SOTTOPROBLEMI (DELLA STESSA NATURA)

IMPERA : CIASCUN SOTTOPROBLEMA E' RISOLTO IN MANIERA RICORSIVA (O IN MANIERA DIRETTA, SE SUFFICIENTEMENTE PICCOLO)

COMBINA : LE SOLUZIONI DEI SOTTOPROBLEMI VENGONO COMBinate PER GENERARE UNA SOLUZIONE DEL PROBLEMA ORIGINALE

CASO DI STUDIO: L'ALGORITMO MERGE SORT

DIVIDE : LA SEQUENZA DEGLI n ELEMENTI DA ORDINARE E' DIVISA IN 2 SEQUENZE DI $\frac{n}{2}$ (CIRCA) ELEMENTI

IMPERA : CIASCUNA SOTTOSEQUENZA E' ORDINATA RICORSIVAMENTE

COMBINA : LE DUE SOTTOSEQUENZE ORDINATE SONO FUSE IN UN'UNICA SEQUENZA ORDINATA

MERGE-SORT (A, p, r)
{ORDINA LA SOTTOSEQUENZA $A[p..r]$ }

if $p < r$

then

$q := \lfloor (p+r)/2 \rfloor$

MERGE-SORT (A, p, q)

MERGE-SORT ($A, q+1, r$)

MERGE (A, p, q, r)

MERGE (A, p, q, r)

{ $p \leq q < r$, $A[p..q]$ E $A[q+1..r]$ SONO ORDINATI }

$n_1 := q - p + 1$; $n_2 := r - q$

CREA $L[1..n_1+1]$ E $R[1..n_2+1]$

for $i := 1$ to n_1
do $L[i] := A[p+i-1]$ } COPIA $A[p..q]$ IN $L[1..n_1]$

for $j := 1$ to n_2
do $R[j] := A[q+j]$ } COPIA $A[q+1..r]$ IN $R[1..n_2]$

$L[n_1+1] := +\infty$; $R[n_2+1] := +\infty$ } SENTINELLE

$i := 1$; $j := 1$

for $k := p$ to r

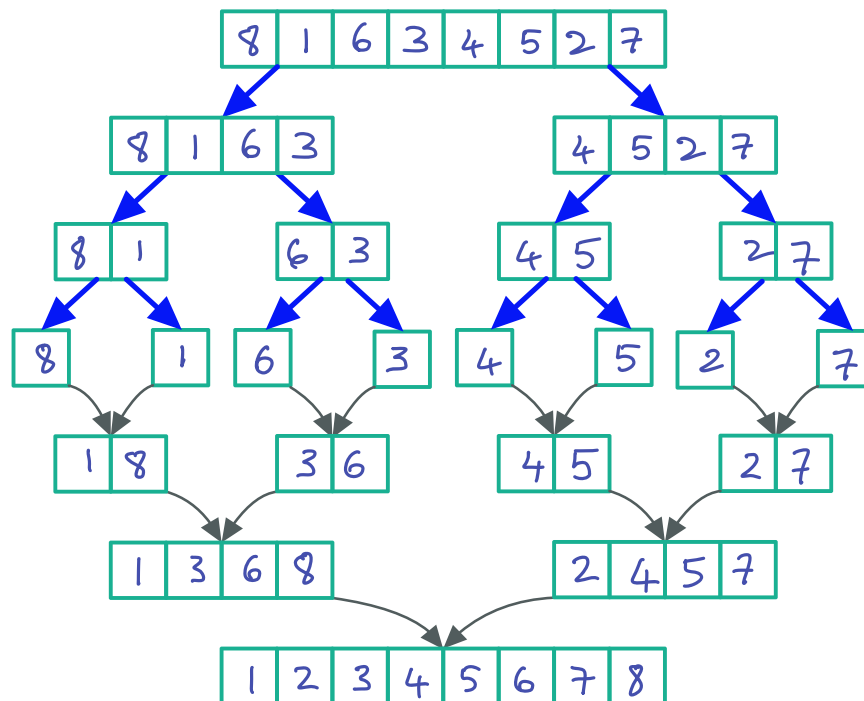
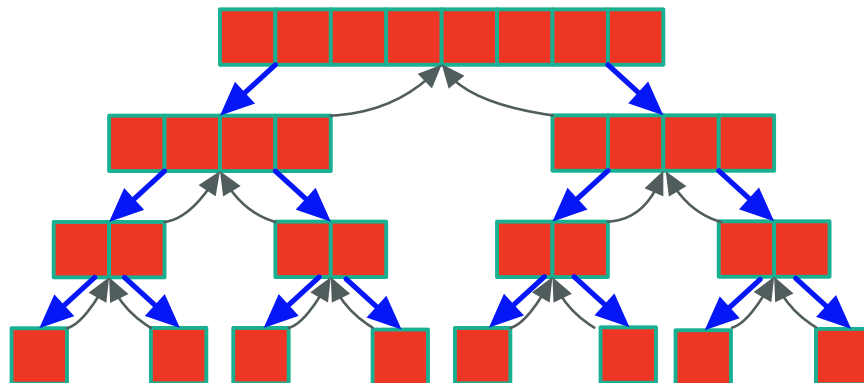
do if $L[i] \leq R[j]$
then $A[k] := L[i]$

$i := i + 1$
else $A[k] := R[j]$
 $j := j + 1$

COMPLESSITA' $\Theta(n)$, con $n = r - p + 1$

DIVIDE

COMBINA



ANALISI DEGLI ALGORITMI DIVIDE ET IMPERA

- $T(m)$ - TEMPO DI ESECUZIONE SU INPUT DI DIMENSIONE n
- a - NUMERO DI SOTTOPROBLEMI
- $\frac{m}{b}$ - DIMENSIONE DI CIASCUN SOTTOPROBLEMA
- s - SOGLIA AL DI SOTTO DELLA QUALE NON C'E' RICORSIONE
- $D(m)$ - TEMPO PER DIVIDERE IL PROBLEMA IN SOTTOPROBLEMI
- $C(m)$ - TEMPO PER COMBINARE LE SOLUZIONI DEI SOTTOPROBLEMI NELLA SOLUZIONE DEL PROBLEMA ORIGINALE
- $\left(\frac{n}{b_1}, \frac{m}{b_2}, \dots, \frac{m}{b_a}\right)$

SI OTTIENE LA SEGUENTE RICORRENZA:

$$T(m) = \begin{cases} \Theta(1) & \text{SE } n \leq s \\ a T\left(\frac{m}{b}\right) + D(m) + C(m) & \text{SE } n > s \end{cases}$$

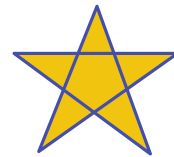
OPPURE $T(m) = \left(\sum_{i=1}^a T\left(\frac{m}{b_i}\right) \right) + D(m) + C(m) \quad \text{SE } m > s$

NEL CASO DI MERGE SORT SI HA:

$$s = 1$$

$$a = 2$$

$$b = 2$$



$$D(n) = \Theta(1)$$

$$C(n) = \Theta(n)$$

(COMPLESSITA' DI MERGE)

È QUINDI $T(n)$ SODDISFA LA SEGUENTE RICORRENZA:

$$T(n) = \begin{cases} \Theta(1) & \text{SE } n=1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{SE } n>1 \end{cases}$$

CHE HA SOLUZIONE

$$T(n) = \Theta(n \lg n)$$

STUDIEREMO UN METODO GENERALE PER RISOLVERE RICORRENZE
DI QUESTO TIPO

RISCRIVIAMO LA RICORRENZA NELLA FORMA

$$T(n) = \begin{cases} c & \text{SE } n=1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{SE } n>1 \end{cases}$$

REGOLE DI RISCrittURA

$$T(k) = ck + 2T\left(\frac{k}{2}\right), \quad k \geq 2$$

$$T(1) = c$$

$$k = n$$

$$T(n) = cn + 2T\left(\frac{n}{2}\right)$$

$$k = \frac{n}{2}$$

$$= cn + 2\left(c\frac{n}{2} + 2T\left(\frac{n}{2^2}\right)\right)$$

$$= 2cn + 2^2 T\left(\frac{n}{2^2}\right)$$

$$k = \frac{n}{2^2}$$

$$= 2cn + 2^2 \left(c\frac{n}{2^2} + 2T\left(\frac{n}{2^3}\right)\right)$$

$$= 3cn + 2^3 \cdot T\left(\frac{n}{2^3}\right)$$

$$\vdots$$

$$= icn + 2^i \cdot T\left(\frac{n}{2^i}\right)$$

$$\vdots$$

$$= kcn + 2^k T\left(\frac{n}{2^k}\right)$$

$$= cn \lg n + n T(1)$$

$$= cn \lg n + cn$$

Supp. $n = 2^k$
 $k = \lg n$

RISCRIVIAMO LA RICORRENZA NELLA FORMA

$$T(n) = \begin{cases} c & \text{SE } n=1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{SE } n>1 \end{cases}$$

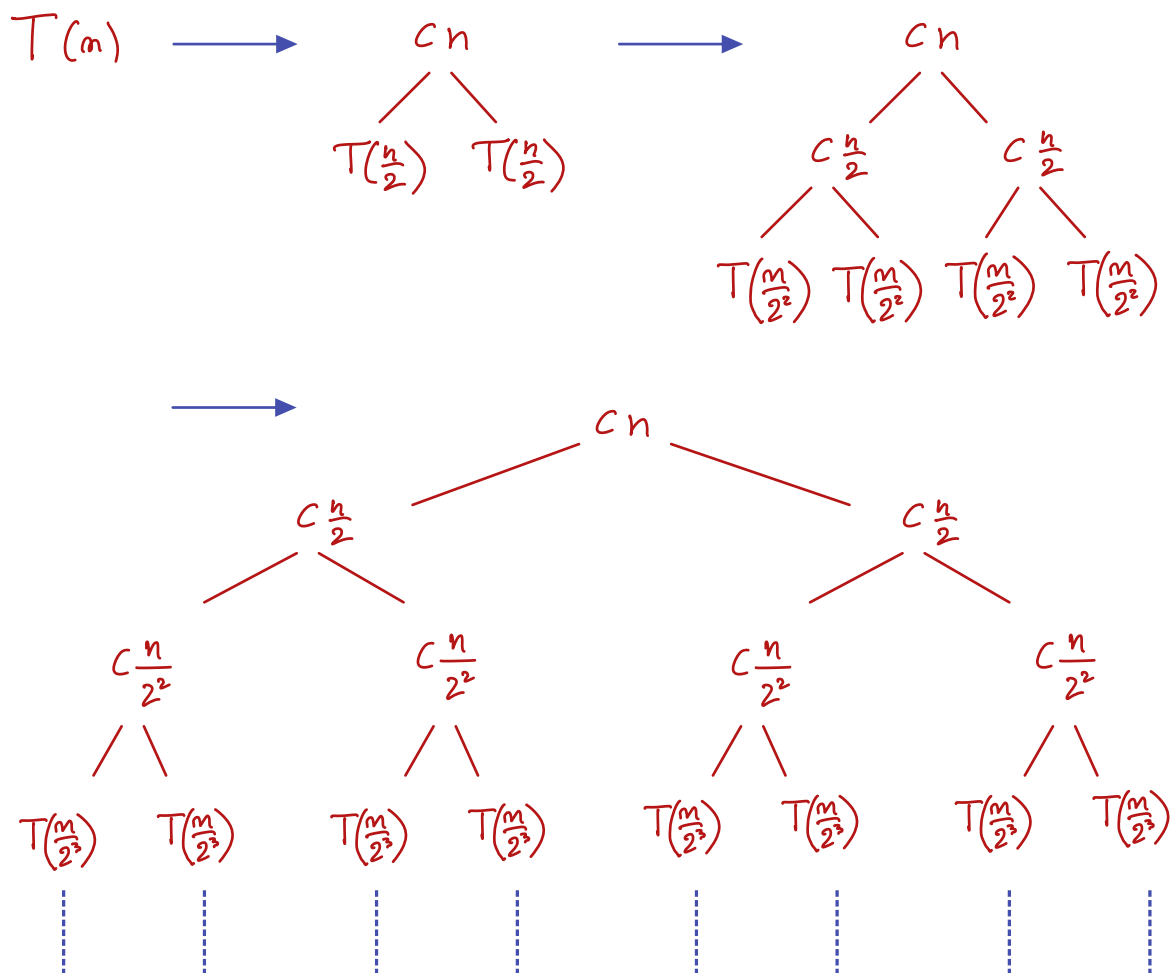
REGOLE DI RISCrittURA PER LA COSTRUZIONE
DI UN ALBERO DI RICORSIONE

$$\frac{T(1)}{c}$$

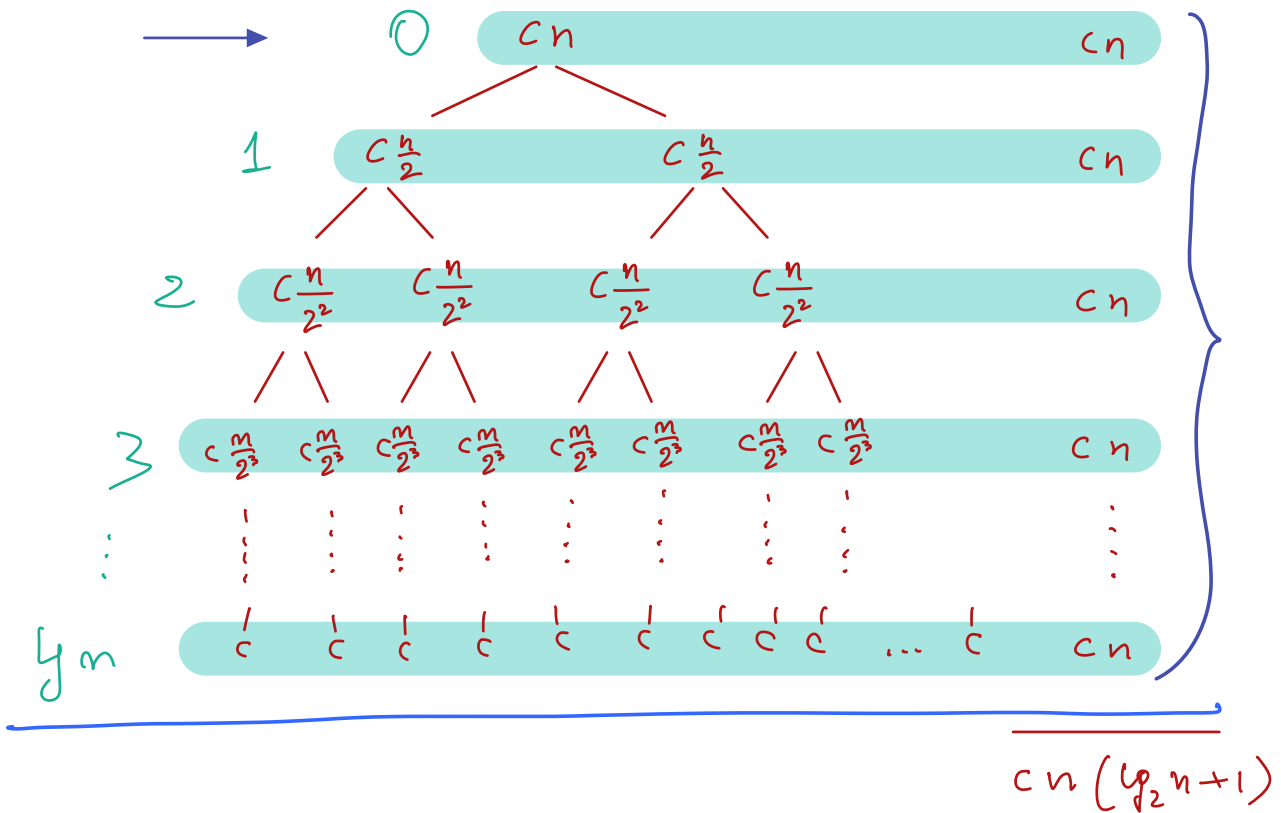
$$\frac{T(k)}{ck}$$

$$\swarrow \quad \searrow$$

$$T\left(\frac{k}{2}\right) \quad T\left(\frac{k}{2}\right)$$



$$\begin{array}{rcl}
 \cancel{T(n)} & T(n) & T(n) \\
 c & & \\
 & 10 & 30 \\
 & 7 + 3 & 14 + 16 \\
 & & + 5 = 45
 \end{array}$$



SUPPOSE THAT $n = 2^k$ ($k = \lg_2 n$)

$$\therefore \# \text{ LEVELS} = k + 1 = \lg_2 n + 1$$

$$\begin{aligned}
 \therefore T(n) &= cn(\lg_2 n + 1) \\
 &= cn \lg_2 n + cn
 \end{aligned}$$

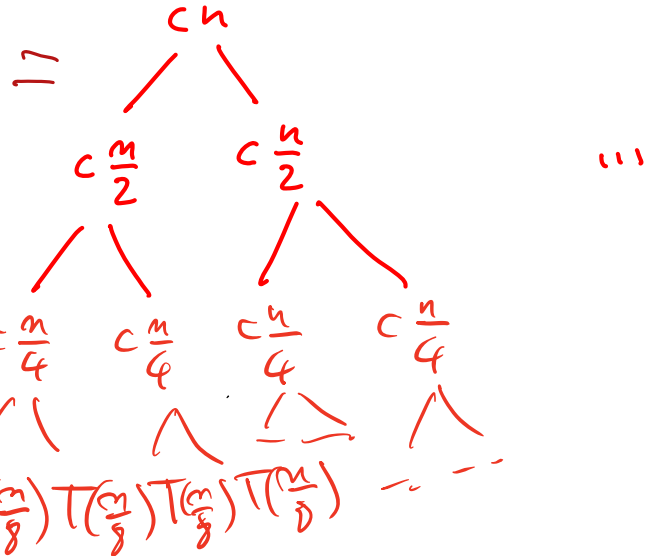
RISCRIVIAMO LA RICORRENZA NELLA FORMA

$$T(n) = \begin{cases} c & \text{SE } n=1 \\ 2T(\frac{n}{2}) + cn & \text{SE } n>1 \end{cases}$$

$$n = 2^k$$

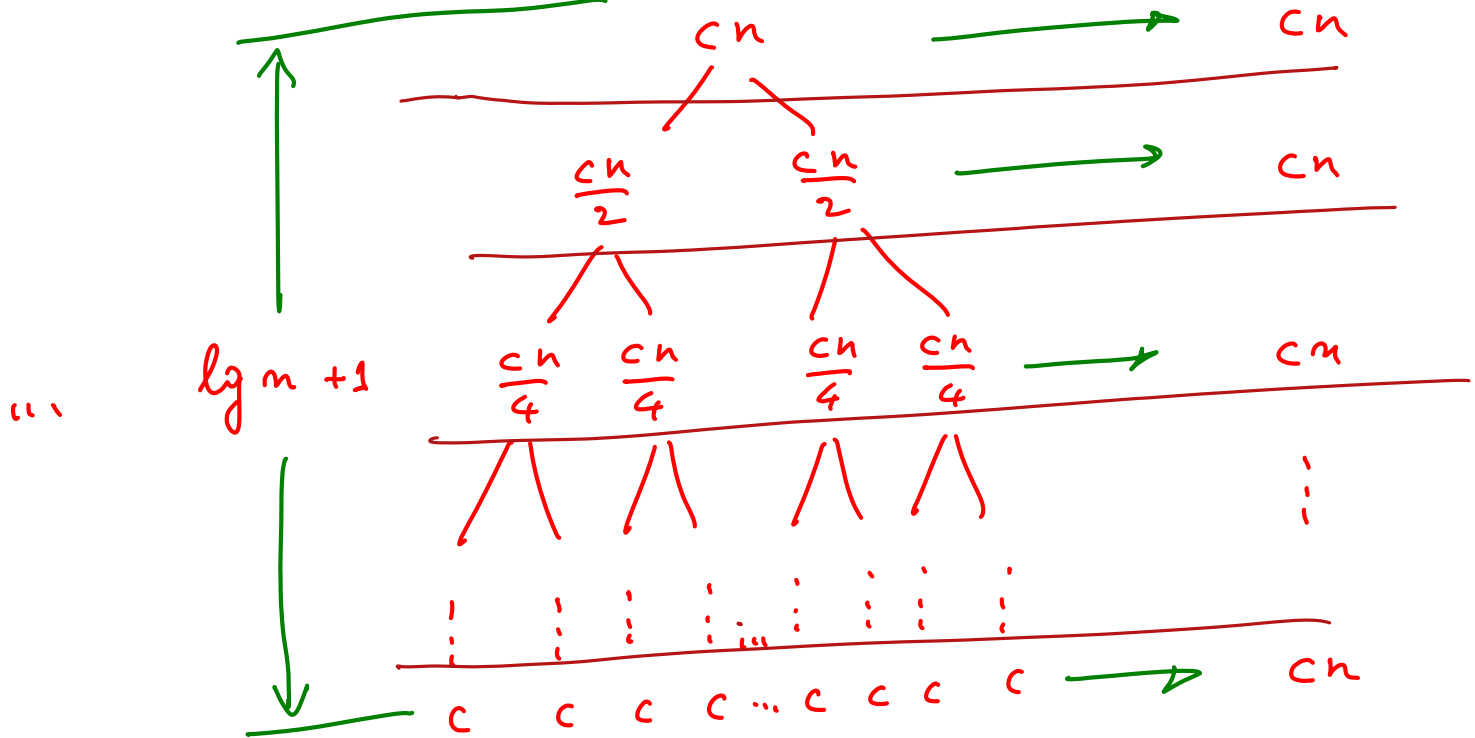
E LA RISOLVIAMO COSTRUIENDO L'ALBERO DI RICORSIONE

$$T(n) = \begin{array}{c} cn \\ \swarrow \quad \searrow \\ T(\frac{n}{2}) \quad T(\frac{n}{2}) \end{array}$$



$$T(\frac{n}{4}) = 2T(\frac{n}{8}) + c\frac{n}{4}$$

$$n = 2^k \Rightarrow k = \lg n$$



$k+1$ - livelli

TOTALE: $cn \lg n + cn$

PERTANTO: $T(n) = \Theta(n \lg n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn = 2^2 T\left(\frac{n}{2^2}\right) + 2cn$$

$$= 2^2\left(2T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right) + 2cn = 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

⋮

$$= 2^h T\left(\frac{n}{2^h}\right) + hcn$$

⋮

$$= 2^{\lg n} T(1) + cn \lg n$$

$$\geq cn + cn \lg n$$