In questa lezione

- **Il Mergesort: primo esempio di applicazione della tecnica divide et impera**
- **analisi tempo di esecuzione del Mergesort**

- **[CLRS] par. 2.3.**

Progettazione di algoritmi

- **Gli ordinamenti quadratici come l'insertionSort usano un approccio incrementale**
- **Il Mergesort che studiamo in questa lezione è un'applicazione della tecnica di progettazione di algoritmi nota come "divide et impera" (divide and conquer)**

Divide et impera

Dato un problema

- **Dividi: dividi il problema in un certo numero di sotto problemi, che sono istanze dello stesso problema, ma di dimensione inferiore e**
- **Conquista: risolvi ricorsivamente i sotto problemi, quando il sotto problema è abbastanza piccolo risolvilo direttamente**
- **Combina: usa le soluzioni dei sotto problemi per determinare la soluzione del problema dato**

Divide et impera e Mergesort

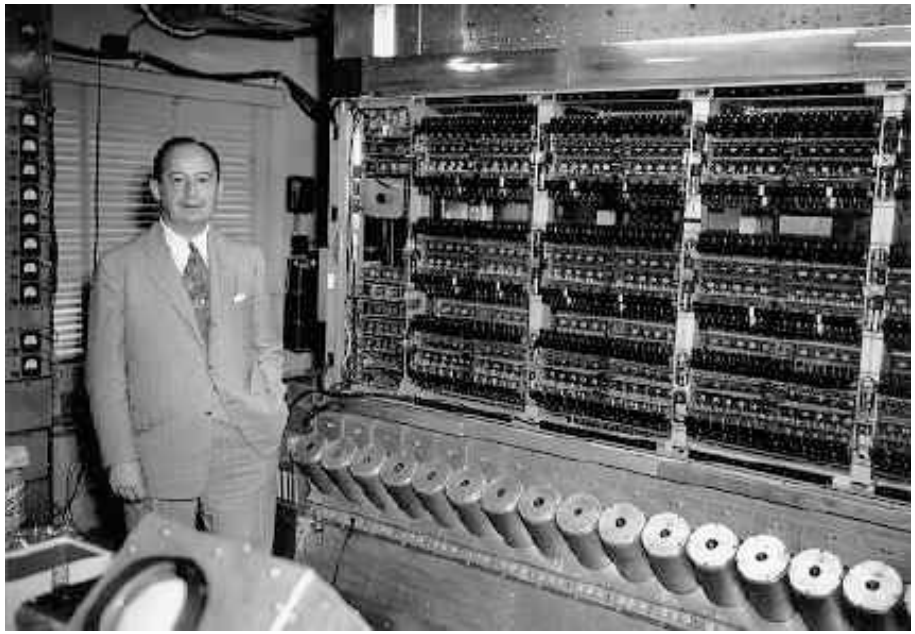
- **Dividi: dividi la sequenza di n elementi da ordinare in due sottosequenze di circa n/2 elementi**

- **Conquista: Ordina le sottosequenze ricorsivamente usando il Mergesort,**
  quando la sequenza contiene un solo elemento
  è ordinato per definizione.

- **Combina: fondi (merge) le sotto sequenze ordinate, producendo così la sequenza voluta**

Paternità

Knuth attribuisce a Von Neumann la stesura di un programma di ordinamento basato sul Mergesort, più o meno in corrispondenza con l'uscita del suo rapporto interno "First Draft of a Report on the EDVAC (Electronic Discrete Variable Automatic Computer)" del 1945.



John von Neumann(1903-1957)

L' EDVAC è il primo calcolatore che usa una memoria magnetica. Questo è un grosso passo avanti tecnologico perché i programmi possono essere caricati sul nastro.

Algoritmo Mergesort: l'idea

Mergesort (A)

Input: un array di interi di n elementi
output: A ordinato crescente:
A[0] ≤ ...≤ A[n-1]

1. **Se n = 1 l'array è già ordinato**
2. **altrimenti dividi A circa in due metà e ricorsivamente ordina con il Mergesort le due metà**
3. **Fondi (Merge) in un un unica array ordinato i due array ordinati**

Mergesort (A)

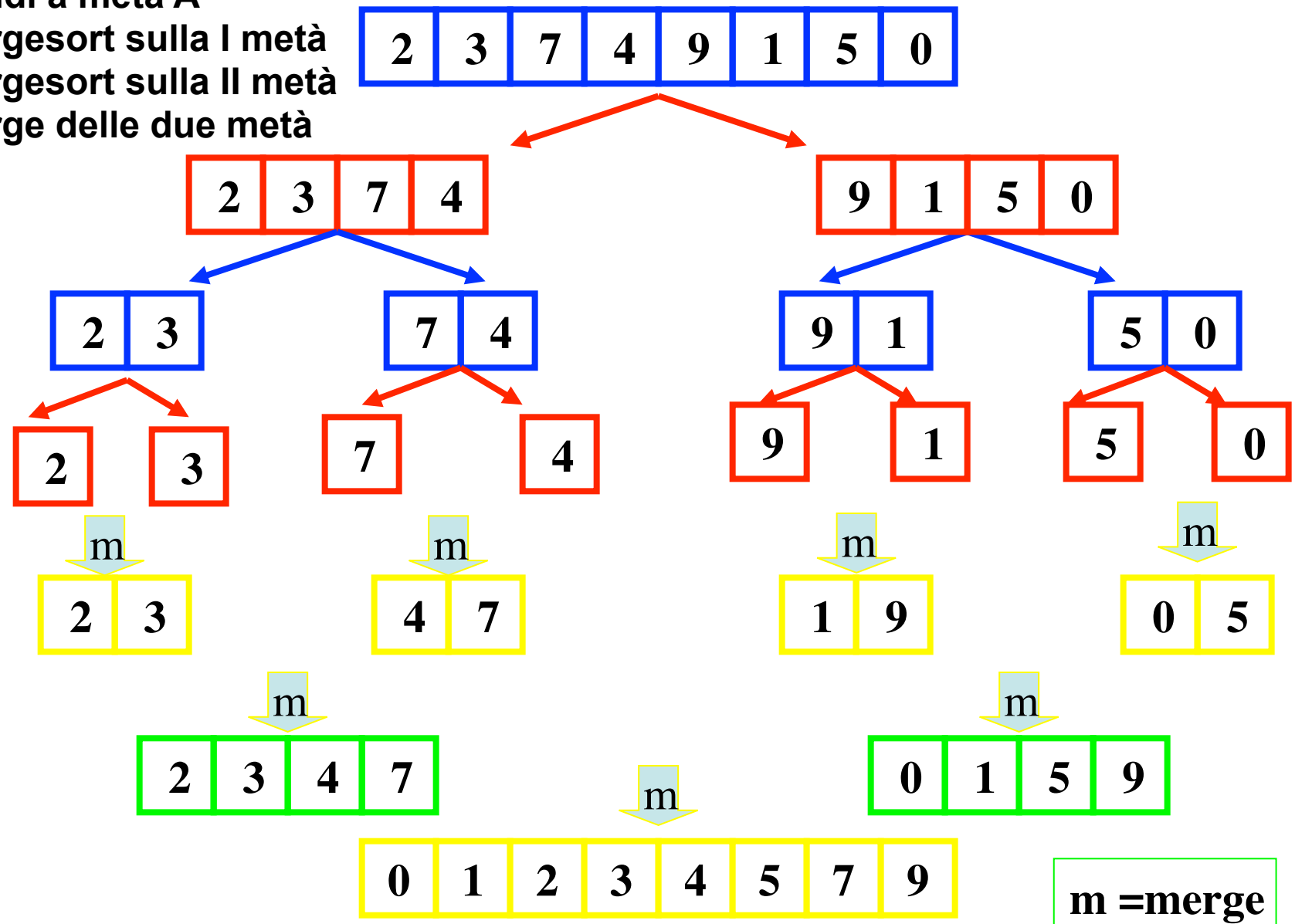
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

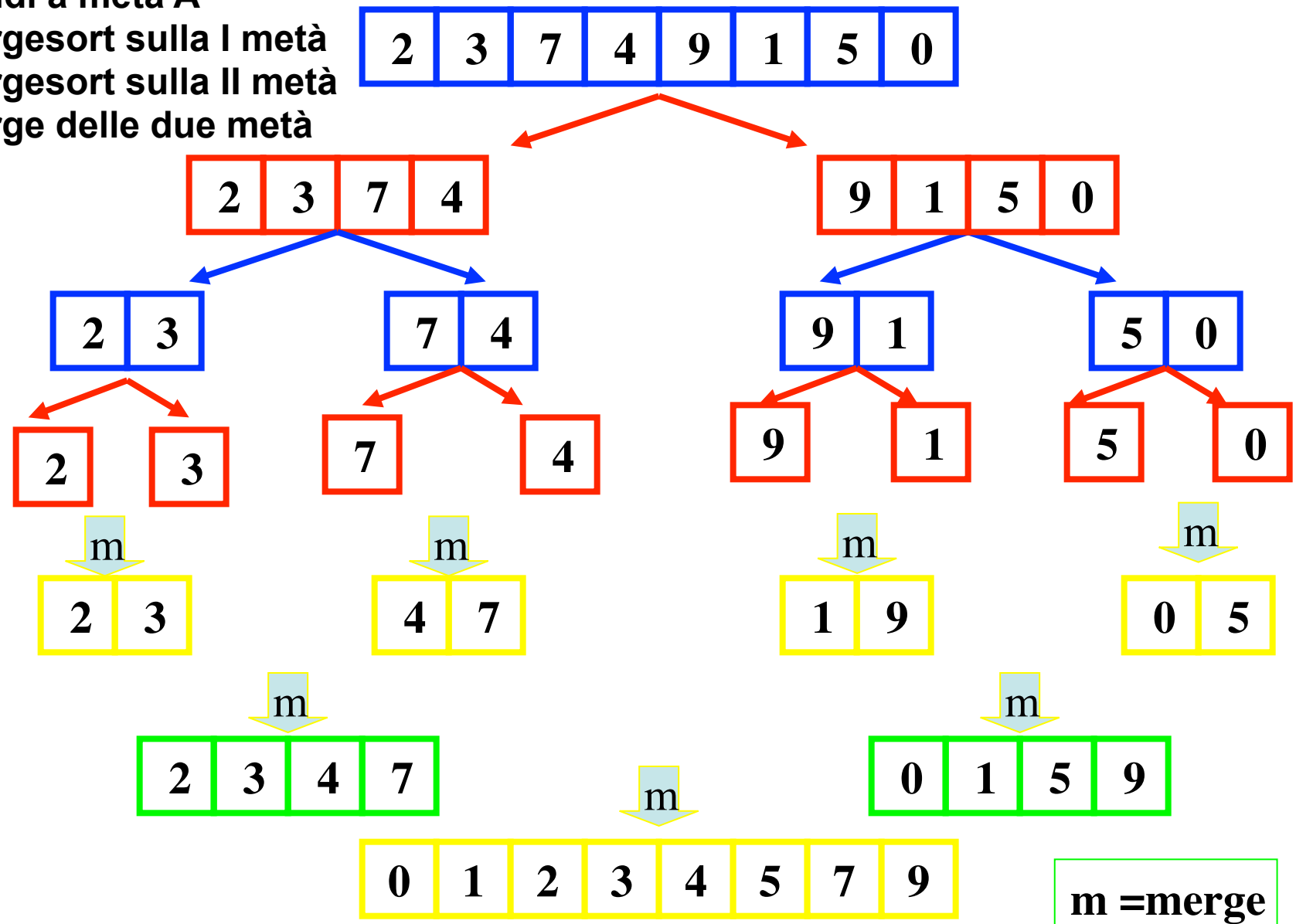
Mergesort sulla II metà

Merge delle due metà



Mergesort (A)

if A ha più di un elemento
dividi a metà A
Mergesort sulla I metà
Mergesort sulla II metà
Merge delle due metà



Mergesort (A)

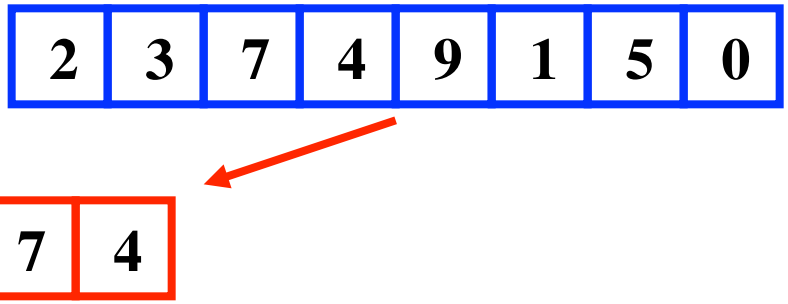
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



Mergesort (A)

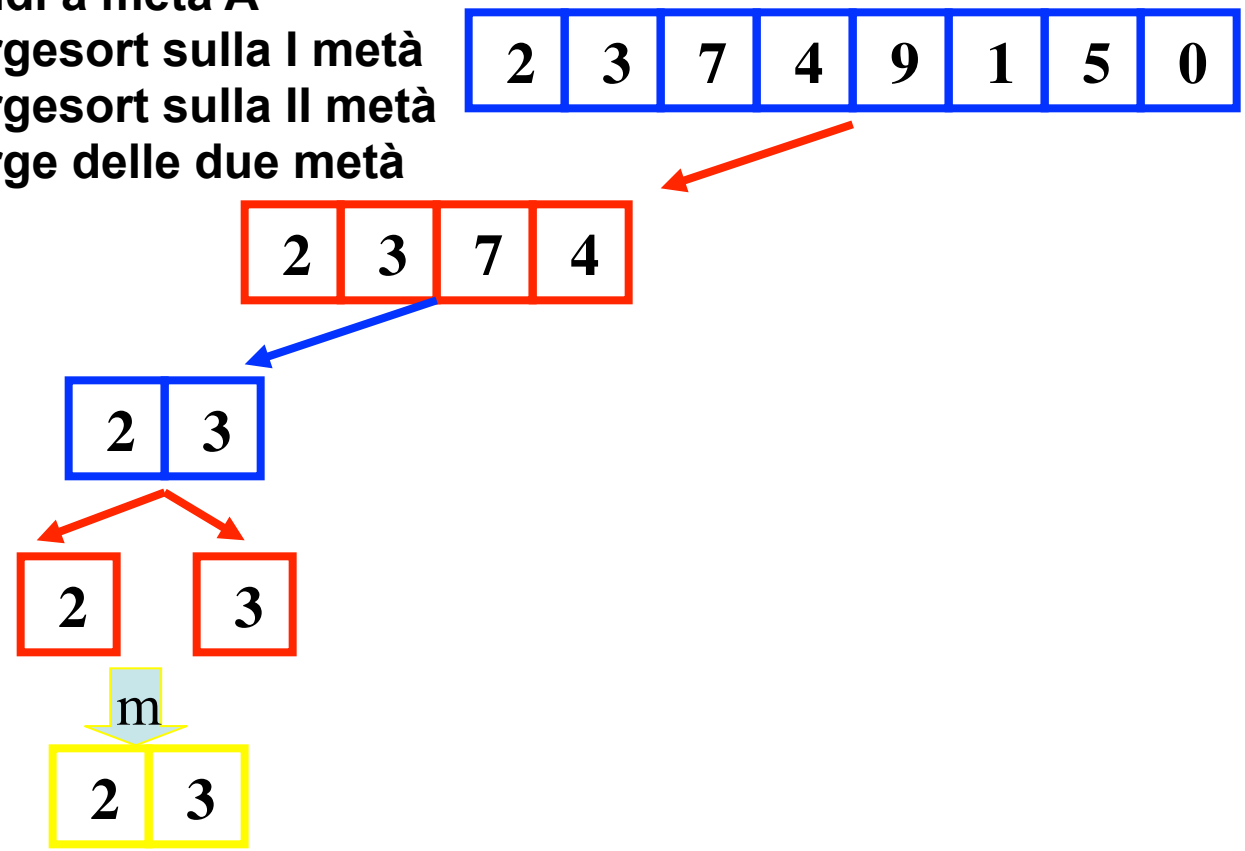
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



Mergesort (A)

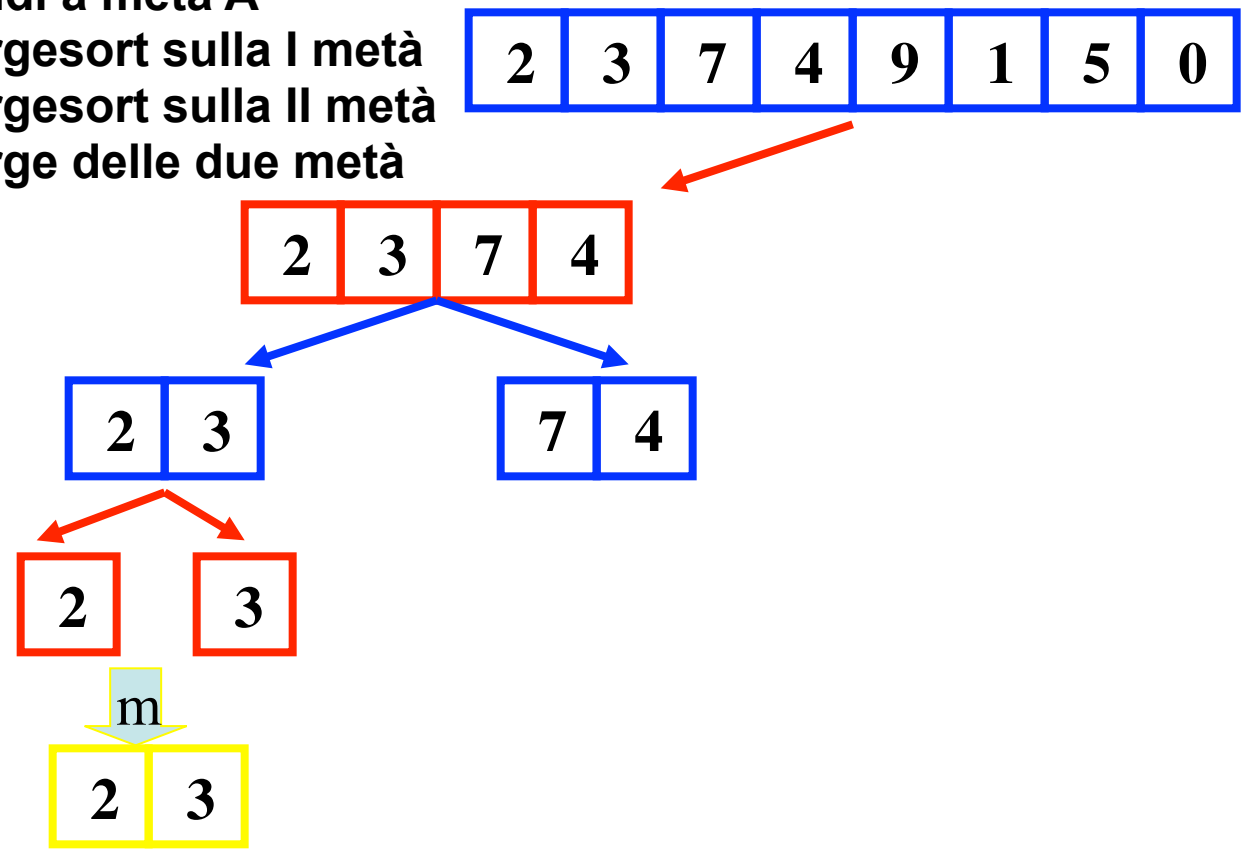
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



m =merge

Mergesort (A)

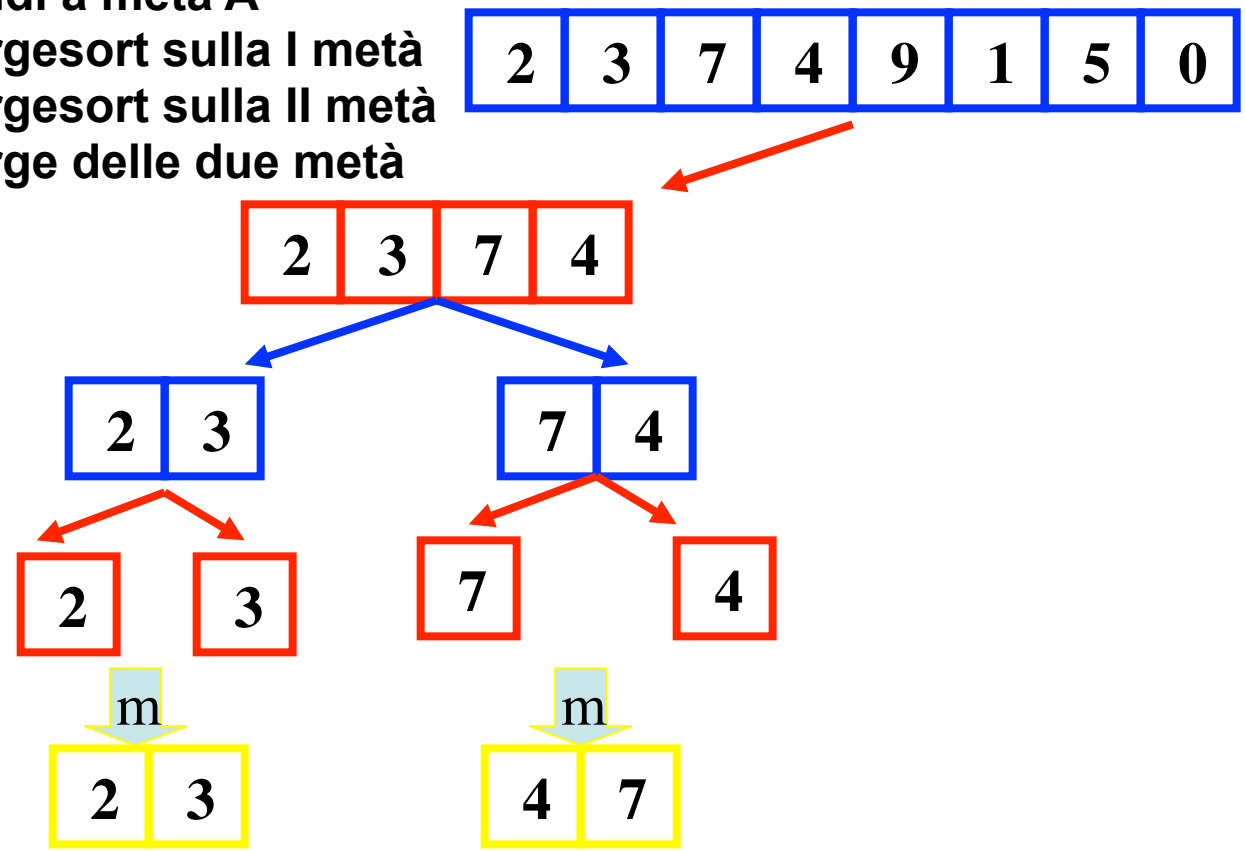
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



m =merge

Mergesort (A)

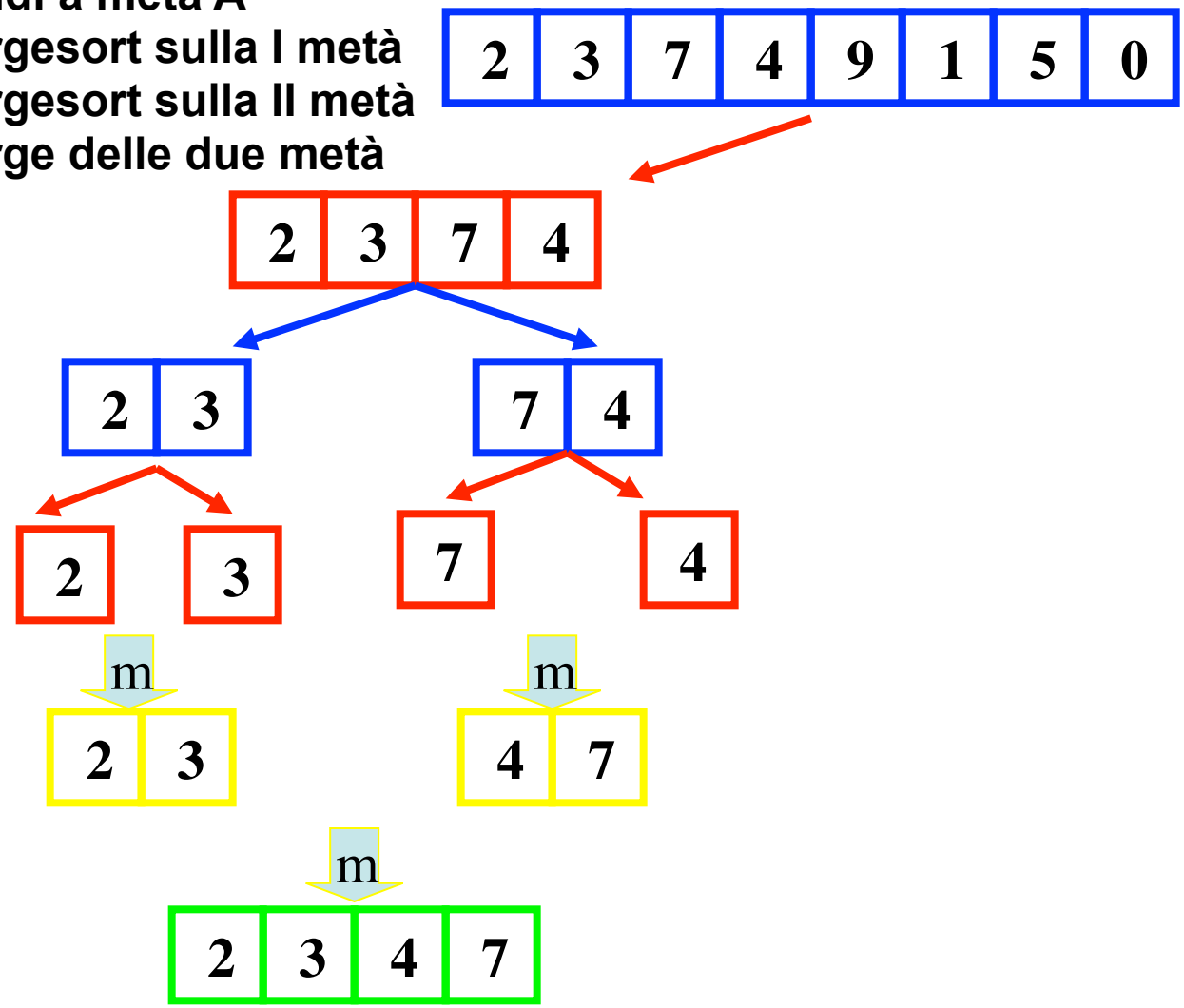
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



m =merge

Mergesort (A)

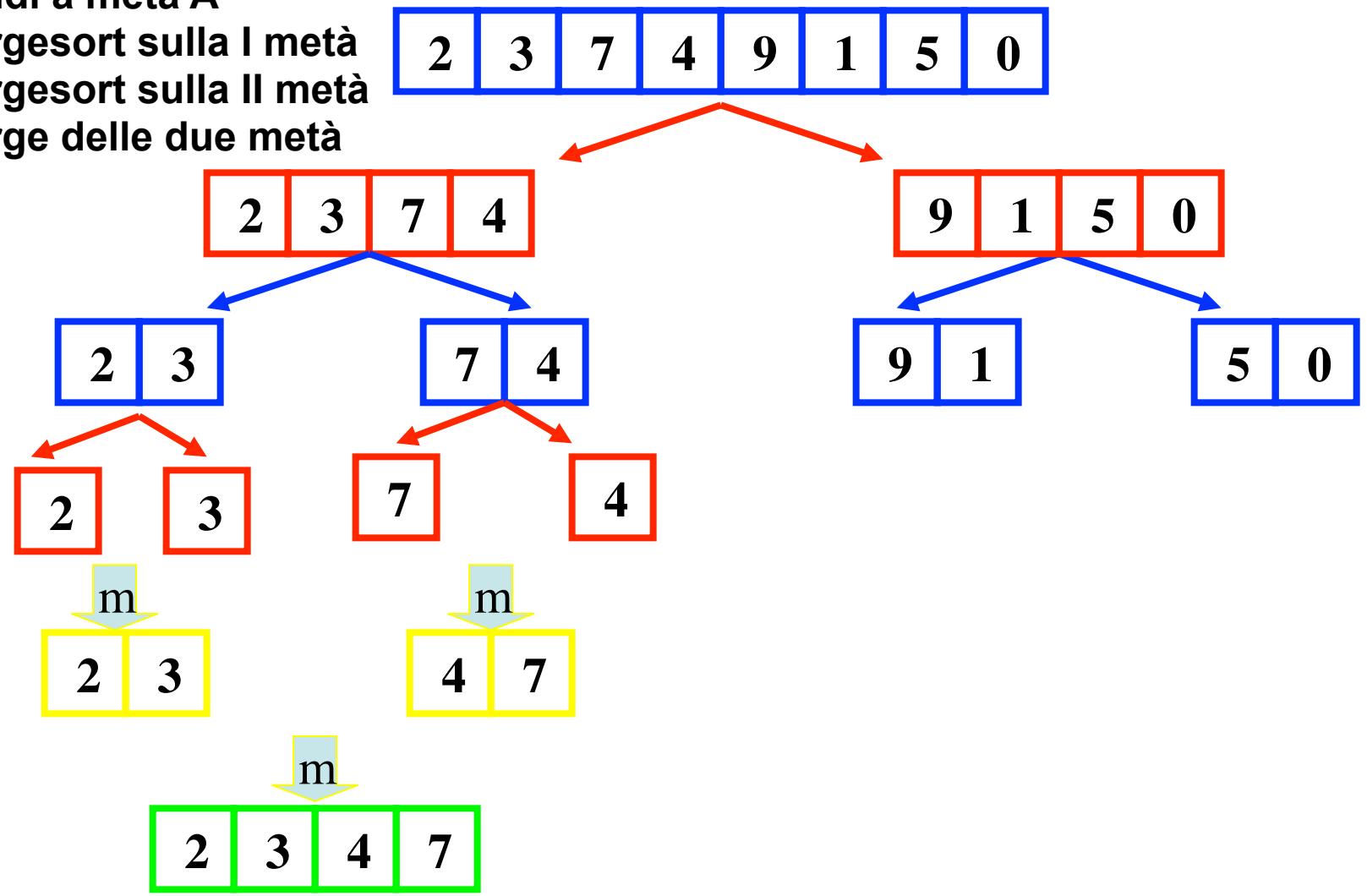
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



m =merge

Mergesort (A)

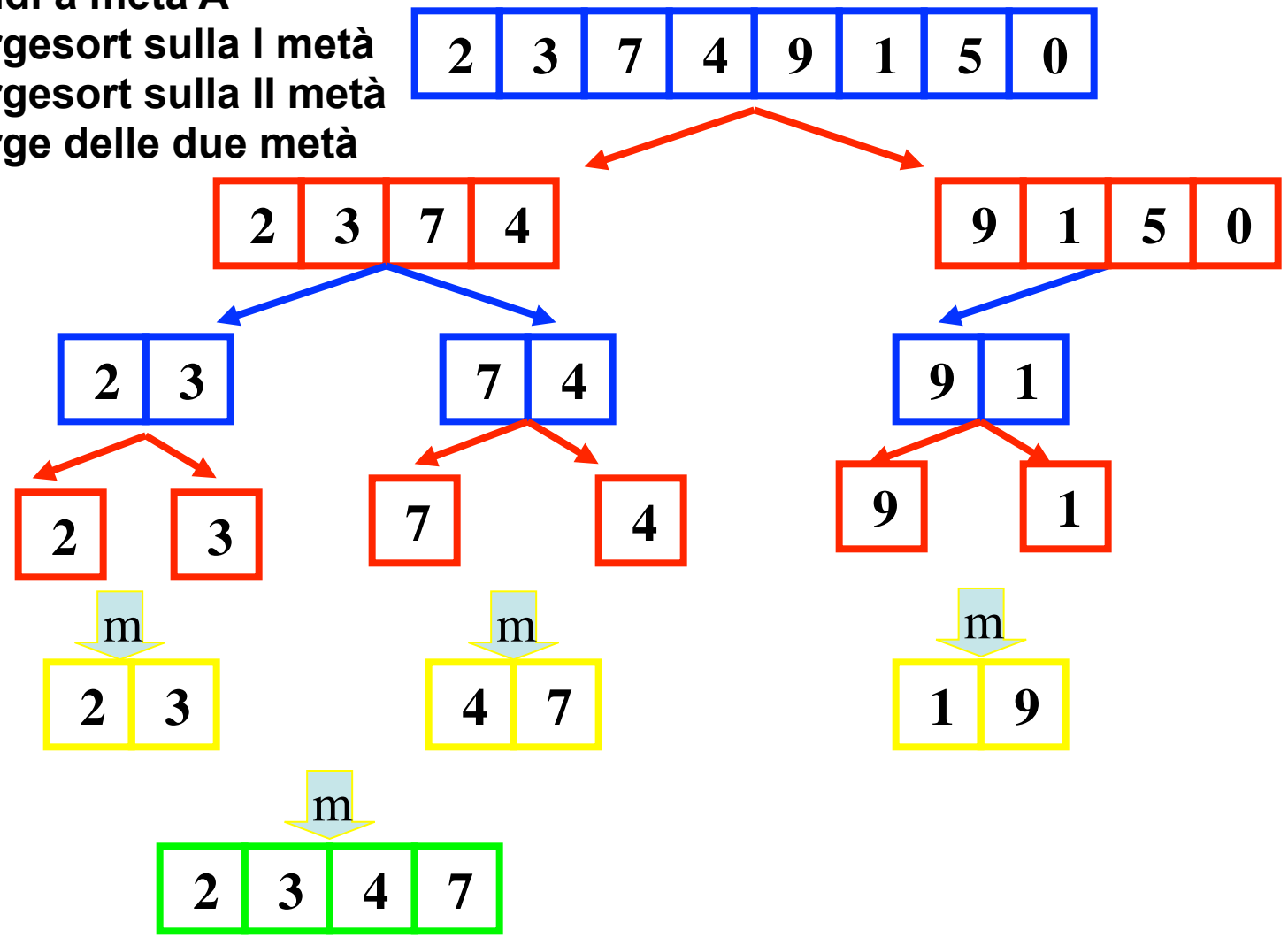
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



m =merge

Mergesort (A)

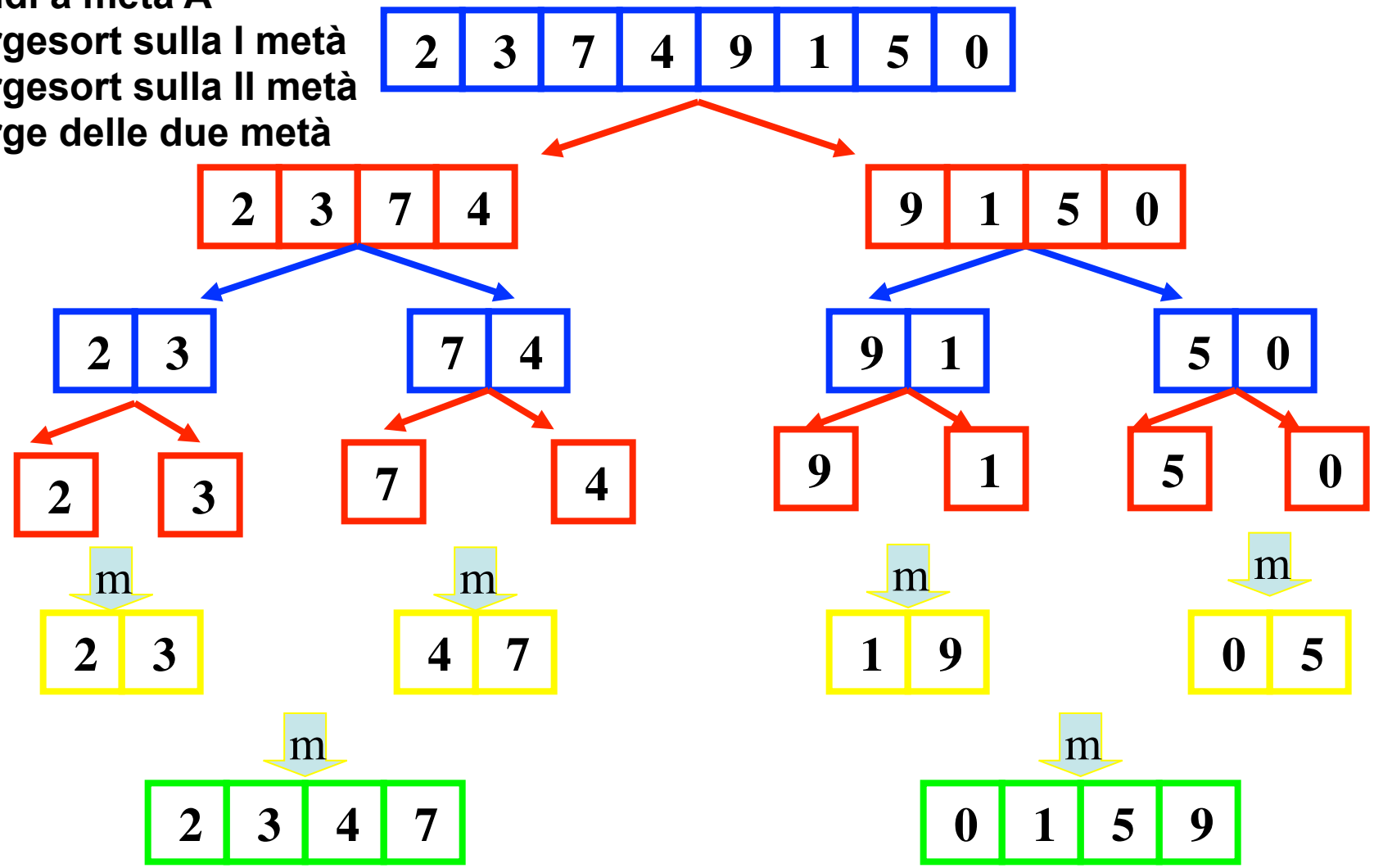
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



m =merge

Mergesort (A)

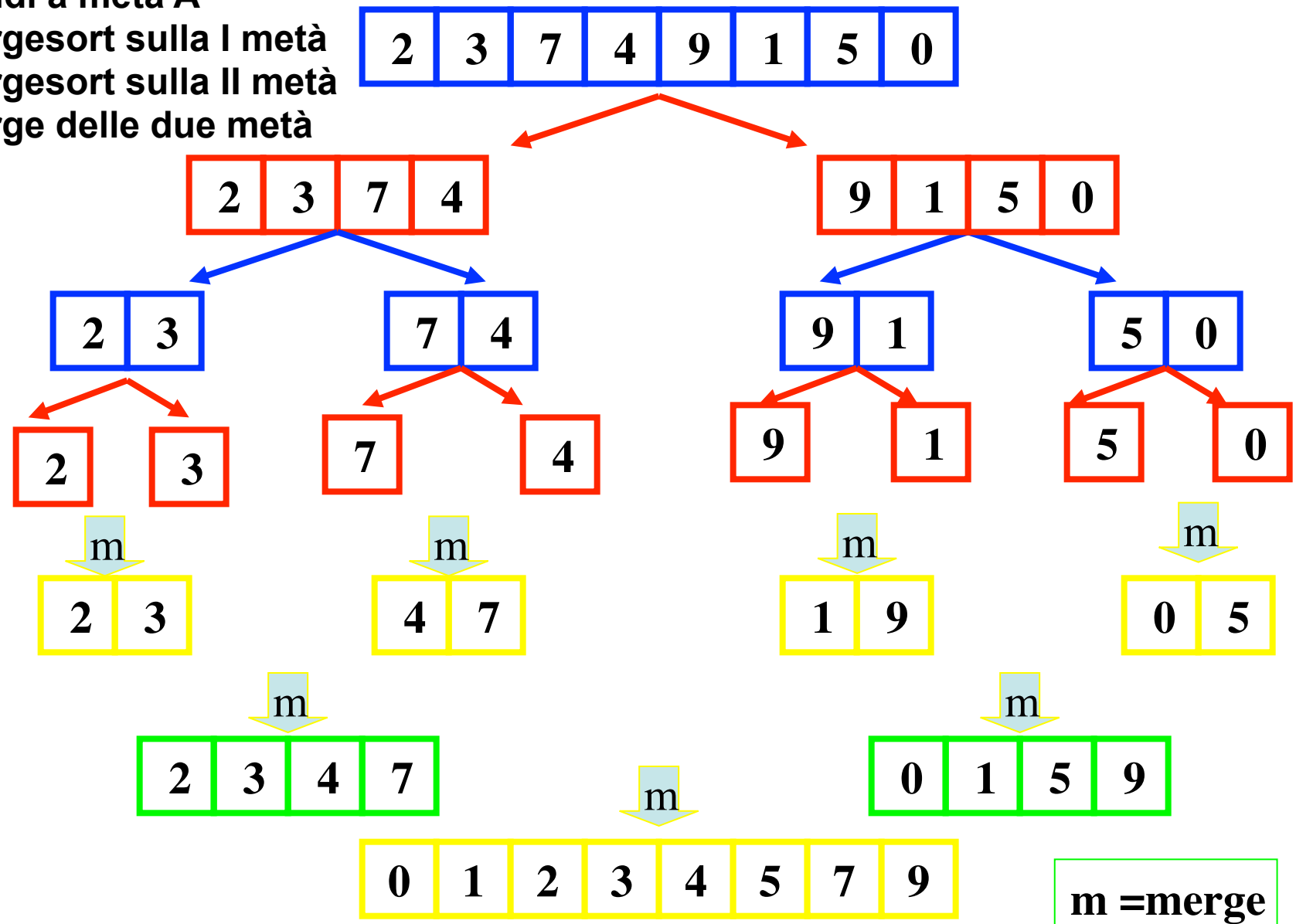
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



Mergesort (A)

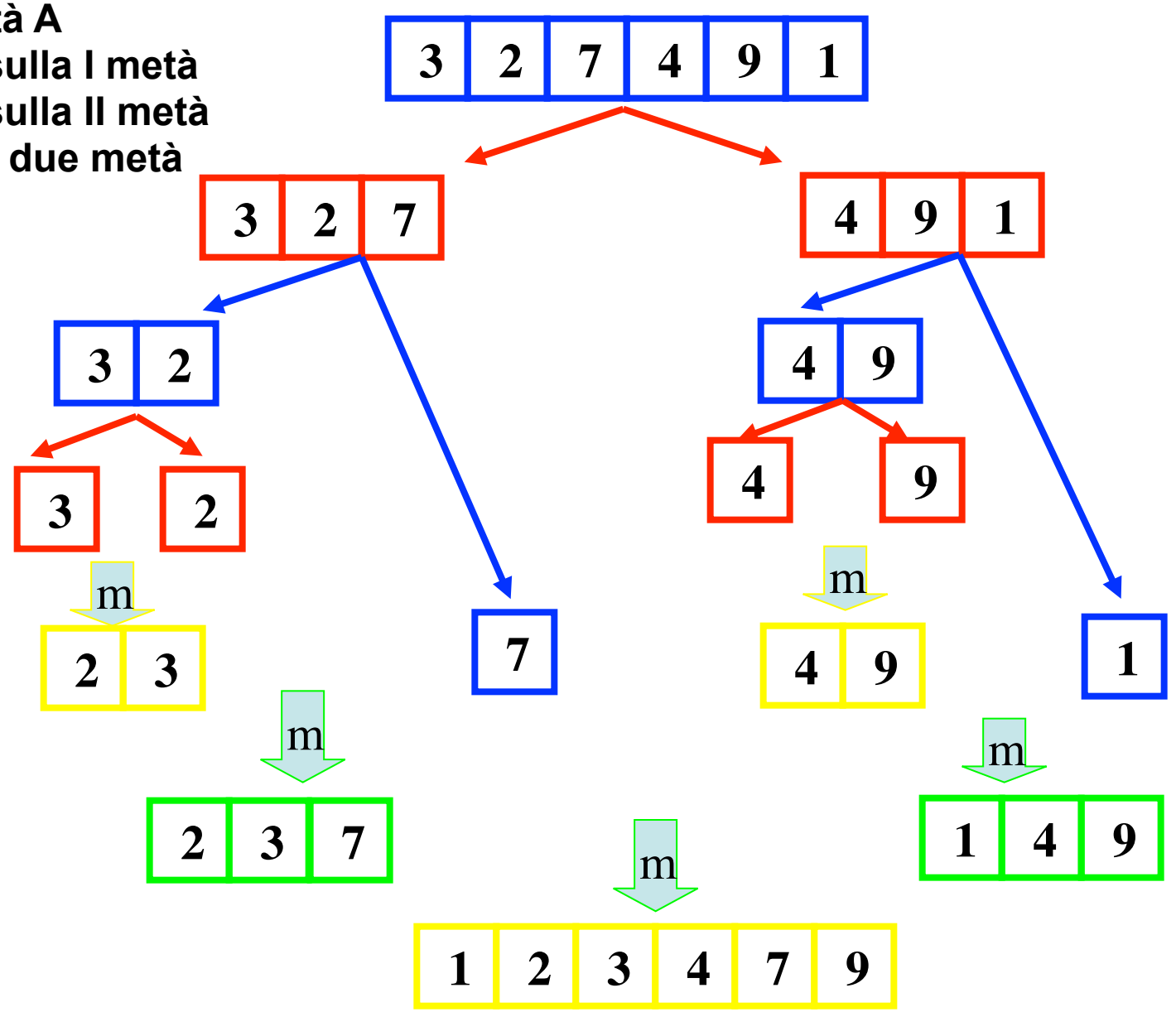
if A ha più di un elemento

dividi a metà A

Mergesort sulla I metà

Mergesort sulla II metà

Merge delle due metà



Mergesort: correttezza

Mergesort (A)
if A ha più di un elemento
dividi a metà A
Mergesort sulla I metà
Mergesort sulla II metà
Merge delle due metà

Dimostriamo che Mergesort è corretto per induzione su n:
Se n=2 è corretto.
Supponiamo che sia corretto per tutti gli arrays con meno di n elementi. Sia A un array con n > 2 elementi ed eseguiamo l'algoritmo.

Le due chiamate agiscono su un numero di elementi che è circa la metà di n, quindi per queste chiamate vale l'ipotesi induttiva: i due sottoarrys sono ordinati al termine dell'esecuzione. Poi viene eseguita la Merge, che correttamente fonde i due sotto arrays ordinati in uno ordinato di n elementi.

Mergesort: complessità

Mergesort (A)		
if A ha più di un elemento	**$\Theta(1)$**	
dividi a metà A	**$\Theta(1)$**	
Mergesort sulla I metà	**?**	**$T(\lceil n/2 \rceil)$**
Mergesort sulla II metà	**?**	**$T(\lfloor n/2 \rfloor)$**
Merge delle due metà	**$\Theta(n)$**	

Relazione di ricorrenza

$T(n) = \Theta(1)$ se $n \leq 1$

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$ altrimenti

$\lceil n/2 \rceil$ = Parte intera superiore di n/2 **$\lfloor n/2 \rfloor$ = Parte intera inferiore**

Risolvere una ricorrenza

Una ricorrenza è un'equazione o una disequazione che lega il valore di una funzione ai valori che essa assume su argomenti più piccoli.

Risolvere una ricorrenza per una funzione T(n) consiste nel determinare un limite asintotico (O grande, Ω o Θ) per T(n)

Metodi di soluzione:
Iterazione
Sostituzione → **In questo corso**
Teorema dell'esperto

Metodo della sostituzione

Fase 1. Si fa una previsione (guess) sulla soluzione

Fase 2. Si verifica se la previsione è esatta, usando l'induzione

Passo 1: semplificare

La ricorrenza è

$T(n) = \Theta(1)$ se $n \leq 1$
$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$ altrimenti

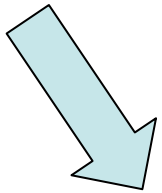
Passiamo ai reali:

$T(n) = \Theta(1)$ se $n \leq 1$
$T(n) = 2T(n/2) + \Theta(n)$ altrimenti

Passo 2: esplicitare le costanti

Per poter risolvere una ricorrenza bisogna esplicitare le costanti nascoste

$T(n) = \Theta(1)$ se $n \leq 1$
$T(n) = 2T(n/2) + \Theta(n)$ altrimenti



$T(n) = d$ se $n \leq 1$
$T(n) = 2T(n/2) + cn$ altrimenti

Dove d e c sono costanti positive, la prima dà conto del tempo speso nel passo base, la seconda dei tempi costanti relativi alle istruzioni nella merge.

SVILUPPO, con calcoli dettagliati

$T(n) = d$ se $n \leq 1$
$T(n) = 2T(n/2)+cn.$

$T(n) = 2T(n/2)+cn =$
$2(2T(n/2^2) + cn/2) + cn =$
$2^2T(n/2^2) + cn + cn =$
$2^2(2T(n/2^3) + cn/2^2) + cn + cn =$
$2^3T(n/2^3) + cn + cn + cn =$
…

Perchè $T(n/2) = 2T(n/2^2) + cn/2$
in base alla definizione di T

Perchè $T(n/2^2) = 2T(n/2^3) + cn/2^2$
in base alla definizione di T

$2^iT(n/2^i) + i*cn =$

Al passo i-simo di sviluppo

…
$2^hT(1) + h*cn =$
$nT(1) + \lg n*cn =$
$nd + cn\lg n = O(n \lg n)$

Supponendo $n = 2^h$

SVILUPPO, con calcoli dettagliati

$T(n) = d$ se $n \leq 1$
$T(n) = 2T(n/2)+cn.$

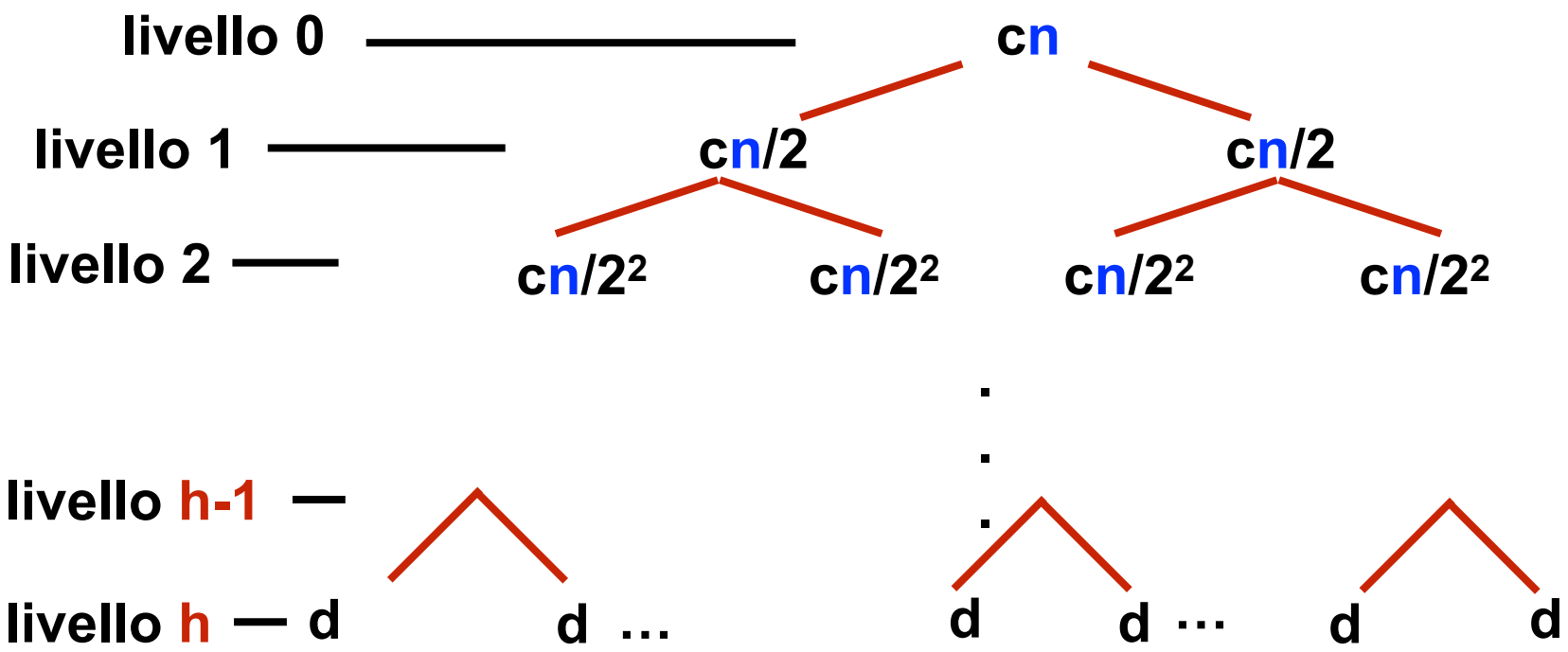
$T(n) = 2^h T(m) + h*cn =$
$nT(m) + \lg n * cn =$
$nd + cn \lg n = O(n \lg n)$

Se $2^{h-1} < n \leq 2^h$ allora
$1/2 = 2^{h-1}/2^h < n/2^h \leq 2^h/2^h = 1$
Con $m \leq 1$

Albero di ricorsione

$T(n) = d$ se $n \leq 1$
$T(n) = 2T(n/2) + cn.$

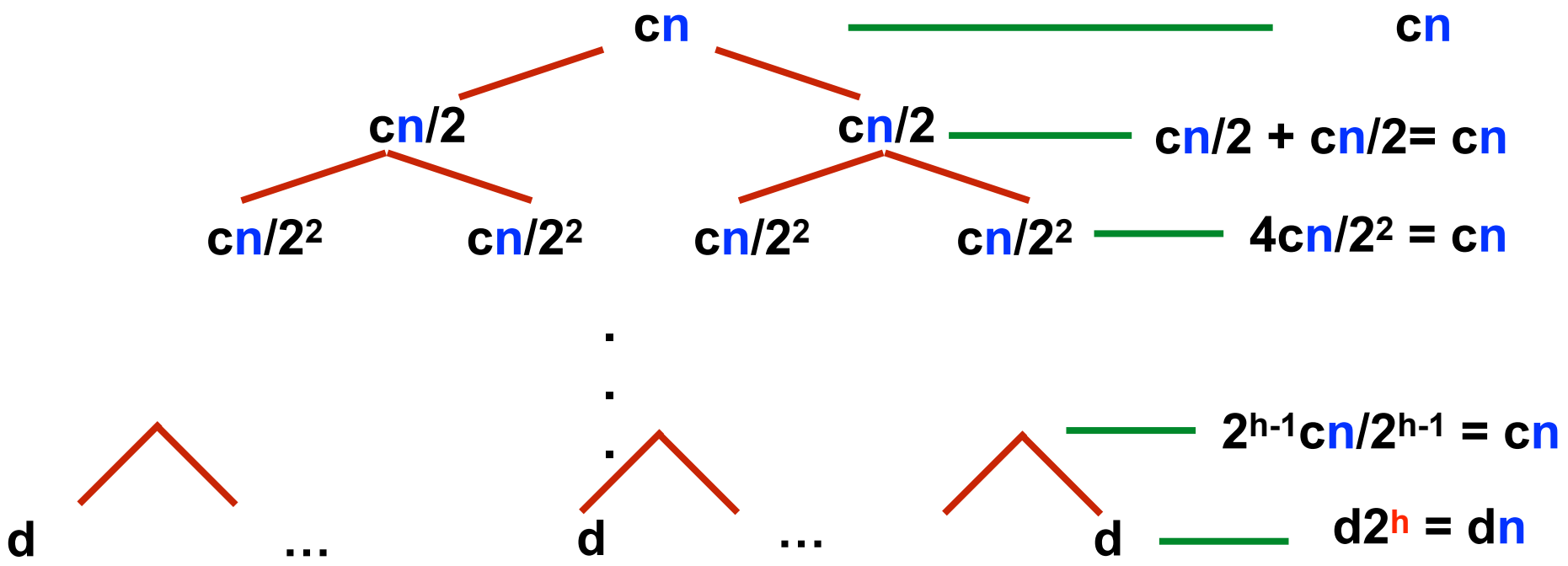
Sia $2^{h-1} < n \leq 2^h$



Albero di ricorsione

$T(n) = 2T(n/2)+cn.$

Sia $2^{h-1} < n \leq 2^h$



E' un albero binario con $h+1$ livelli.
Su ogni livello, tranne l'ultimo, la somma dei costi è cn
quindi si ha un costo totale circa $cn \lg n + dn$.
Quindi la nostra ipotesi è che $T(n) = \Theta(n \lg n)$

Prova che T(n) = O(n lg n)

Dimostriamo per induzione che $T(n) = O(n \lg n)$ e cioè che esistono due costanti positive k ed n_0 tali che $T(n) \leq k\, n \lg n$ per ogni $n \geq n_0$.

Sappiamo che $T(n) = 2T(n/2) + cn$.

Supponiamo che la nostra tesi sia vera per ogni $m < n$, allora

$$T(n) = 2T(n/2) + cn$$
$$\leq 2k\,n/2 \lg n/2 + cn =$$
$$k\, n\, (\lg n - \lg 2) + cn$$
$$= kn \lg n - kn + cn.$$

$T(n/2) \leq k\,n/2 \lg n/2$ — **Per ipotesi induttiva**

Prova che T(n) = O(n lg n)

Abbiamo ottenuto, applicando l'ipotesi induttiva che

$T(n) \leq kn \lg n - kn + cn.$

Vogliamo che $T(n) \leq kn \lg n$ per un certo k e per ogni $n \geq n_0$,

Vediamo se troviamo un valore di k per cui
$kn \lg n - kn + cn \leq kn \lg n$
Perchè se è vero per $kn \lg n - kn + cn$ è vero anche per $T(n)$
visto che $T(n) \leq kn \lg n - kn + cn$

Prova che T(n) = O(n lg n)

Vediamo se troviamo un valore di k per cui
kn lg n - kn + cn ≤ kn lg n
⇔ cn ≤ kn
⇔ k ≥ c, per ogni n ≥ 1.

Possiamo concludere che esistono due costanti k = c ed n_0 tali che T(n) ≤ k n lgn per ogni n ≥ n_0.

Però la scelta di n_0 = 1 va modificata perché T(1) = d.

Ricordiamo che c e d sono due costanti legate alla relazione di ricorrenza e che hanno un significato, la prima dando conto del costo costante delle operazioni al di fuori delle chiamate ricorsivi e d dà conto del costo del passo base.

Passo base

**Poiché T(1) = d, che non è minore o uguale di k1 lg 1 = 0,
calcoliamo T(2) e T(3), che dipendono da T(m), con m≤1, per trovare un valore di k che rende vera la disuguaglianza e un valore iniziale per n.**

Calcoliamo T(2):
T(2) = 2T(1) + 2c = 2d+2c
T(2) = 2d+2c ≤ 2k lg2 è vera se k è per esempio il massimo tra c e d, oppure uguale di c+d.

Calcoliamo T(3):
T(3) = 2T(3/2) + 3c = 2d+3c
T(3) = 2d+3c ≤ 3k lg 3 che è vera se prendiamo k = c+d

Quindi possiamo concludere che con le scelte k = c+d e n_0 = 2
T(n) ≤ k n lgn per ogni n ≥ n_0.

Prova che $T(n) = \Omega(n \lg n)$

Per poter concludere che il tempo di esecuzione del Mergesort è $\Theta(n \lg n)$, bisogna dimostrare anche che esistono due costanti positive b ed n_0 tali che $T(n) \geq k\, n \lg n$ per ogni $n \geq n_0$.
Ragioniamo analogamente per induzione.
Supponiamo che la nostra tesi sia vera per ogni $m < n$, allora
$T(n) = 2T(n/2) + cn \geq$
$2k\, n/2 \lg n/2 + cn =$
$k\, n (\lg n - \lg 2) + cn =$
$kn \lg n - kn + cn.$
Si vuole che $T(n) \geq kn \lg n$ per ogni $n \geq n_0$,
Se è vero per $kn \lg n - bn + cn$ è vero anche per $T(n)$:
$kn \lg n - kn + cn \geq kn \lg n$
$\Leftrightarrow - kn + cn \geq 0$
$\Leftrightarrow k \leq c$, **per ogni $n \geq 1$. Quindi possiamo prendere $k = c$ e $n_0 = 1$. Controlliamo $T(1)$**

Prova che $T(n) = \Omega(n \lg n)$

$T(m) = d$, con $m \leq 1$, che è maggiore o uguale di $k1 \lg 1 = 0$, per ogni scelta di k.

Concludiamo che esistono due costanti $k = c$ ed $n_0 = 1$ tali che $T(n) \geq k\, n \lg n$ per ogni $n \geq n_0$

Θ(*n* lg *n*) per il Mergesort

Ricordiamo che abbiamo dimostrato che Ω(*n* lg *n*) è un limite inferiore, nel caso peggiore, per i problema dell'ordinamento, nel caso di algoritmi basati sul confronto.

Gli algoritmi di ordinamento con tempo di esecuzione Θ(*n lg n*) nel caso peggiore, come il mergeSort e l'heapSort, sono asintoticamente ottimali (il loro albero di decisione ha altezza minima).

Nel caso del mergesort possiamo dire che anche in tutti gli altri casi il tempo di esecuzione è Θ(*n lg n*).