

**Un ordinamento con
tempo di esecuzione in
 $\Theta(n \lg n)$**

Fondi (merge)

Fondi(L,R,A)

Input: L,R ed A sono array di m, n e n+m elementi

prec: L e R sono ordinati, cioè

$L[0] \leq \dots \leq L[m-1]$ e $R[0] \leq \dots \leq R[n-1]$

postc: A contiene gli elementi di L e di R e

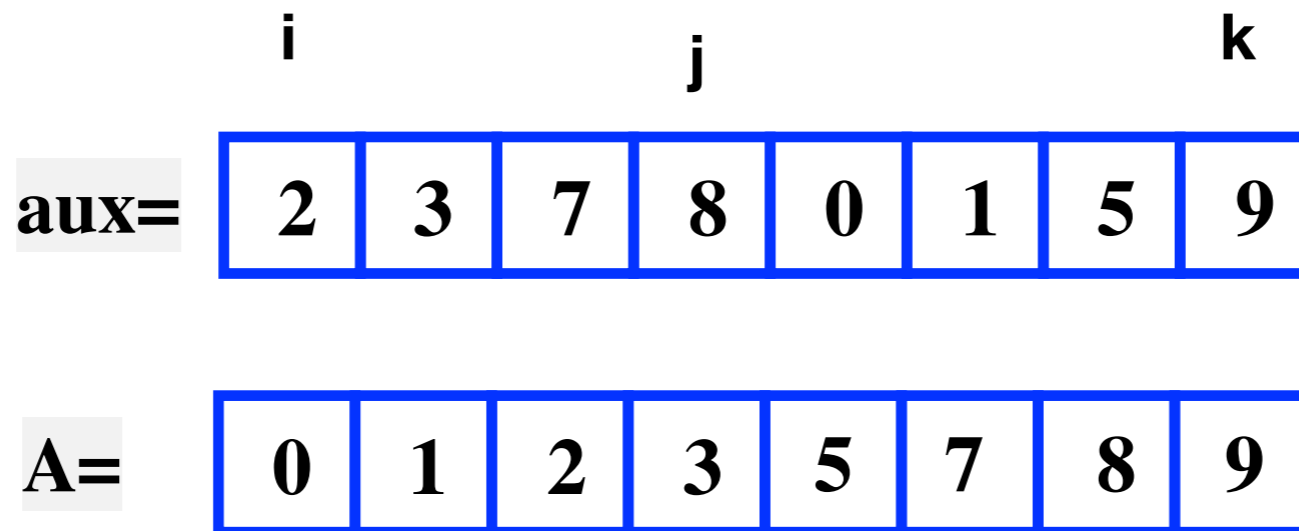
$A[0] \leq \dots \leq A[m+n-1]$

Se $L = A[i:j]$, la porzione dell'array A tra gli indici i e j e $R = A[j+1:k]$ la porzione dell'array A tra gli indici j+1 e k, possiamo eseguire la fusione di queste due parti di A già ordinate.

Fondi (merge)

Possiamo copiare prima le due porzioni in un array ausiliario e scrivere i risultati della fusione in A.

Chiamiamo $\text{merge}(A, \text{aux}, i, j, k)$ questa funzione che fonde, ricopiandoli da aux in A, gli elementi di $A[i:j]$ e $A[j+1, k]$.



Pseudocodice di merge

merge(A,aux, lo, mid, hi)

input: due array A e aux, con lo stesso numero di elementi. Tre interi non negativi

prec: $0 \leq lo \leq mid \leq hi < A.length$, $A[lo] \leq \dots \leq A[mid]$ e $A[mid+1] \leq \dots \leq A[hi]$

output: $A[lo] \leq \dots \leq A[mid] \leq A[mid+1] \leq \dots \leq A[hi]$

{

for (k = lo; k ≤ hi; k++) // copia gli elementi da lo a hi in aux

aux[k] = A[k];

i = lo, j = mid+1;

for (k = lo; k ≤ hi; k++)

{ // fusione

if (i > mid)

A[k] = aux[j++]; // la prima porzione è vuota

else if (j > hi)

A[k] = aux[i++]; // la seconda porzione è vuota

else if (aux[j] ≤ aux[i])

A[k] = aux[j++];

else

A[k] = aux[i++];

}

}

lo i mid j hi

aux=

2	3	7	8	0	1	5	9
---	---	---	---	---	---	---	---

A=

0	1	2	3				
---	---	---	---	--	--	--	--

mergeSort iterativo

L'idea è:

fondi sotto arrays consecutivi di quello iniziale raddoppiando ad ogni passo la dimensione dei sotto arrays da fondere, fino ad arrivare ad aver ordinato tutto l'array iniziale. La prima dimensione è 1 perchè ogni elemento isolato è un insieme ordinato.

dim = 1

merge(A, aux, 0, 0, 1)

merge(A, aux, 2, 2, 3)

merge(A, aux, 4, 4, 5)

merge(A, aux, 6, 6, 7)

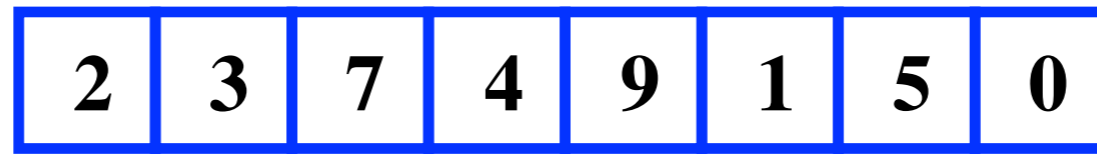
dim = 2

merge(A, aux, 0, 1, 3)

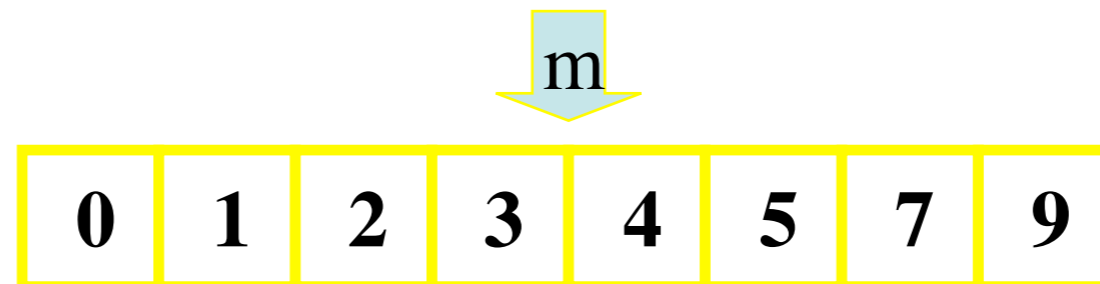
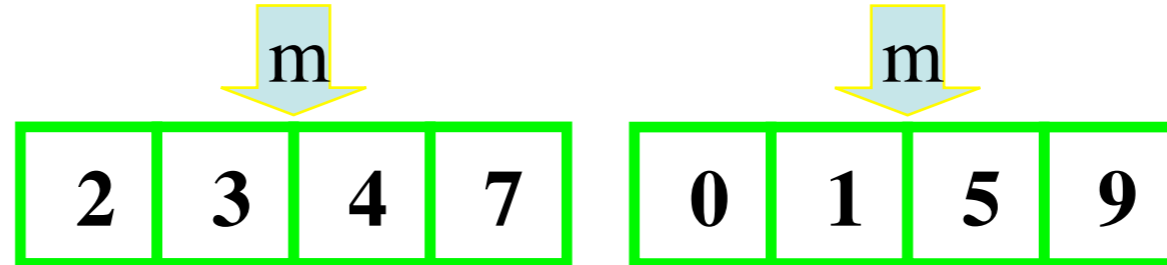
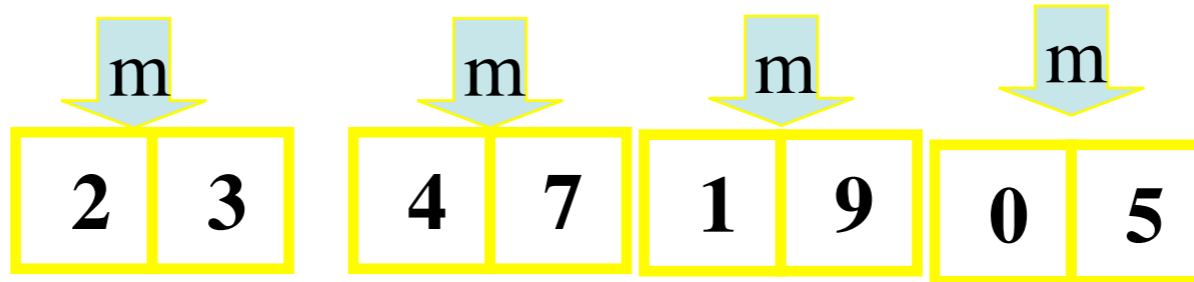
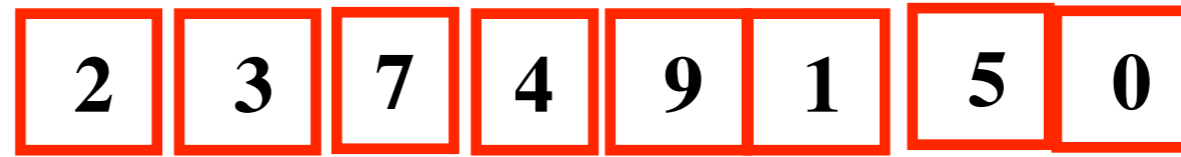
merge(A, aux, 4, 5, 7)

dim = 4

merge(A, aux, 0, 3, 7)



m = merge

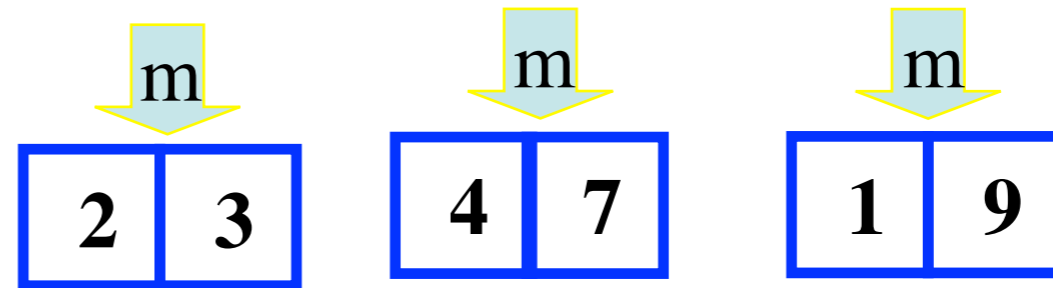
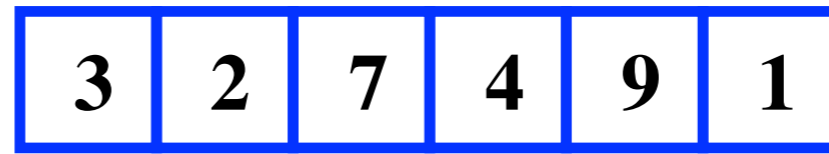


dim = 1

merge(A, aux, 0, 0, 1)

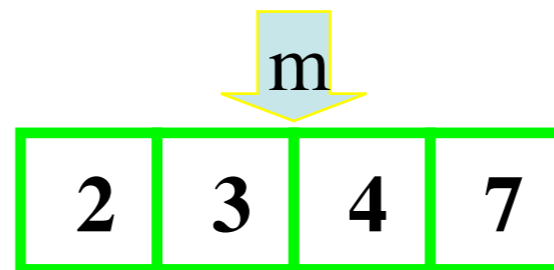
merge(A, aux, 2, 2, 3)

merge(A, aux, 4, 4, 5)



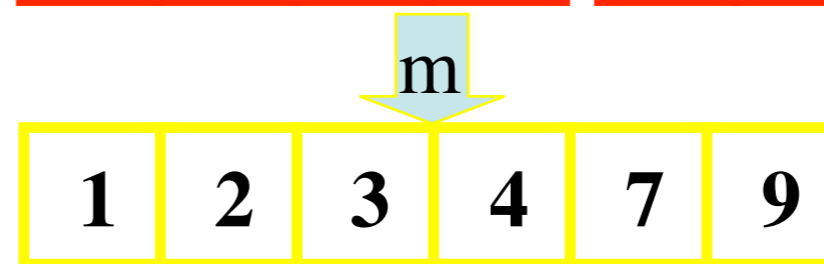
dim = 2

merge(A, aux, 0, 1, 3)



dim = 4

merge(A, aux, 0, 3, 5)



Analisi mergesort iterativo

Qual'è il tempo di esecuzione?

Per comodità mettiamoci nel caso che $n = 2^k$

Passo 1: si eseguono $n/2=2^{k-1}$ fusioni su elementi singoli

Passo 2: si eseguono $n/4=2^{k-2}$ fusioni su coppie di elementi

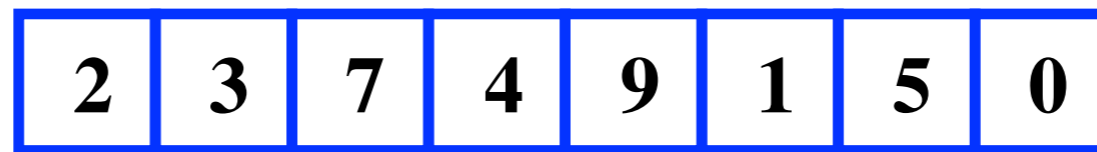
Passo 3: si eseguono $n/8=2^{k-3}$ fusioni su quadruple di elementi

...

qual'è l'ultimo passo?

passo $k-1$: si esegue 1 fusione su due array di 2^{k-1} elementi

$n = 2^3$



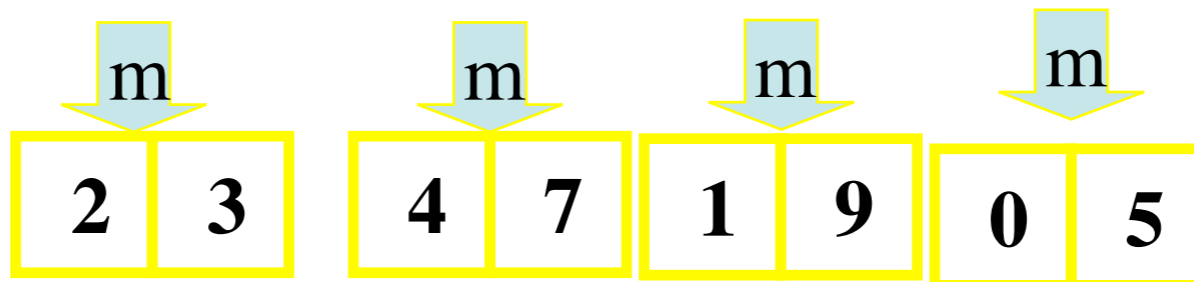
m = merge

dim = 1

2^2 merge,
ognuna eseguita in $\Theta(2)$



tempo totale $\Theta(n)$

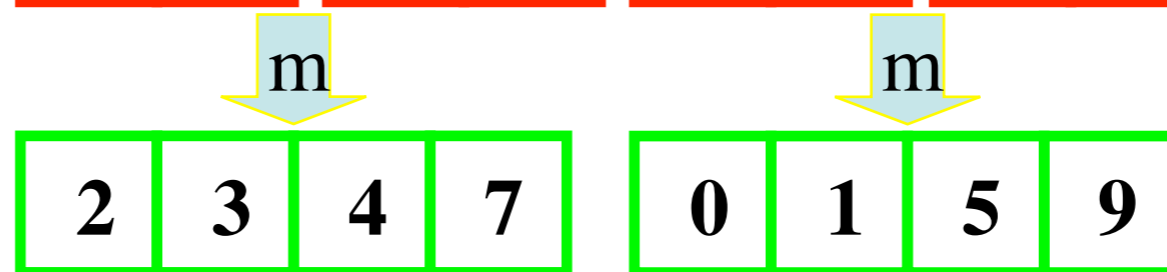


dim = 2

2 merge,
ognuna eseguita in $\Theta(2^2)$



tempo totale $\Theta(n)$

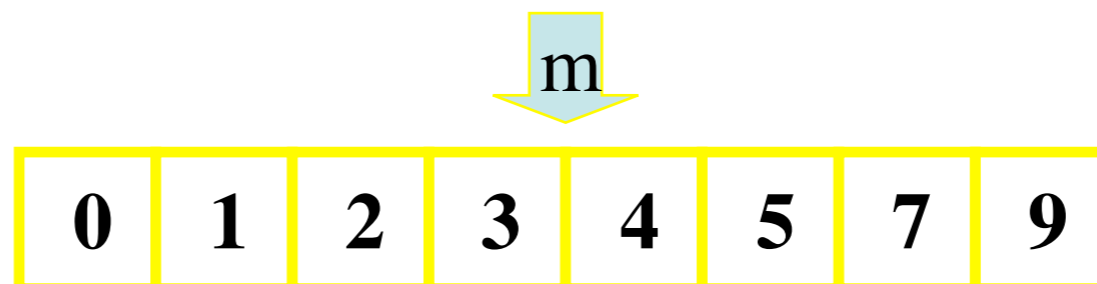


dim = 4

1 merge,
eseguita in $\Theta(2^3)$



tempo totale $\Theta(n)$



Analisi mergesort iterativo

Sia $n = 2^k$

Il passo 1 si esegue in $2^{k-1} * \Theta(2) = \Theta(2^k)$

il passo 2 si esegue in $2^{k-2} * \Theta(4) = \Theta(2^k)$

il passo 3 si esegue in $2^{k-3} * \Theta(8) = \Theta(2^k)$

...

il passo $k-1$ si esegue in $1 * \Theta(2^k) = \Theta(2^k)$

Quindi otteniamo $\Theta(k * 2^k) = \Theta(n \lg n)$

Iterativo

Mergeliter(A)

{...

n = A.length;

sia aux un nuovo array di n elementi

for (dim = 1; dim < n; dim = 2*dim)

for (lo = 0; lo < n - dim; lo += dim+dim)

merge(A, aux, lo, lo+dim-1, Math.min(lo+dim+dim-1, n-1));

}

**L'algoritmo non lavora in loco, ma usa un array ausiliario.
Dunque usa una memoria aggiuntiva di dimensione lineare.**

Tabella di confronto

operazioni al secondo	dimensione del problema = 1 milione	dimensione del problema = 1 milione	dimensione del problema = 1 milione	dimensione del problema = 1 miliardo	dimensione del problema = 1 miliardo	dimensione del problema = 1 miliardo
andamento algoritmo	n	$n \lg n$	n^2	n	$n \lg n$	n^2
10^6	secondi	secondi	settimane	ore	ore	mai
10^9	istantaneo	istantaneo	ore	secondi	secondi	decenni
10^{12}	istantaneo	istantaneo	secondi	istantaneo	istantaneo	settimane

La tabella proviene dal testo di Robert Sedgwick: Algoritmi in Java 3° ed.

Varianti: mergeSort “naturale”

L'idea è di sfruttare la presenza di sotto sequenze ordinate nell'array in input e applicare la merge a queste.

2	3	5	0	4	1	5	9
---	---	---	---	---	---	---	---

2	3	5	0	4	1	5	9
---	---	---	---	---	---	---	---

Si fanno più confronti per identificare le sotto sequenze già ordinate, ma in media meno chiamate della merge

TimSort di Python



Inventato da Tim Peters, è l'algoritmo standard di ordinamento in Python, OpenJDK 7 e Android JDK 1.5

- mergeSort naturale
- usa l'insertionSort binario (cioè la versione con la ricerca binaria) per costruire delle sotto sequenze ordinate, se necessario.
- Ulteriori passi di ottimizzazione, come sfruttare la migliore prestazione di merge su array lunghi una potenza di 2 e dimensionando accuratamente le sottosequenze da ordinare con l'insertion sort binario.

L'algoritmo ottenuto è sempre in $\Theta(n \lg n)$ nel caso peggiore, ma nel caso migliore è lineare e le costanti sono ridimensionate tanto che le prestazioni sperimentali sono migliori in molti casi di altri algoritmi con tempo di esecuzione asintotico in $O(n \lg n)$.

Conclusione

