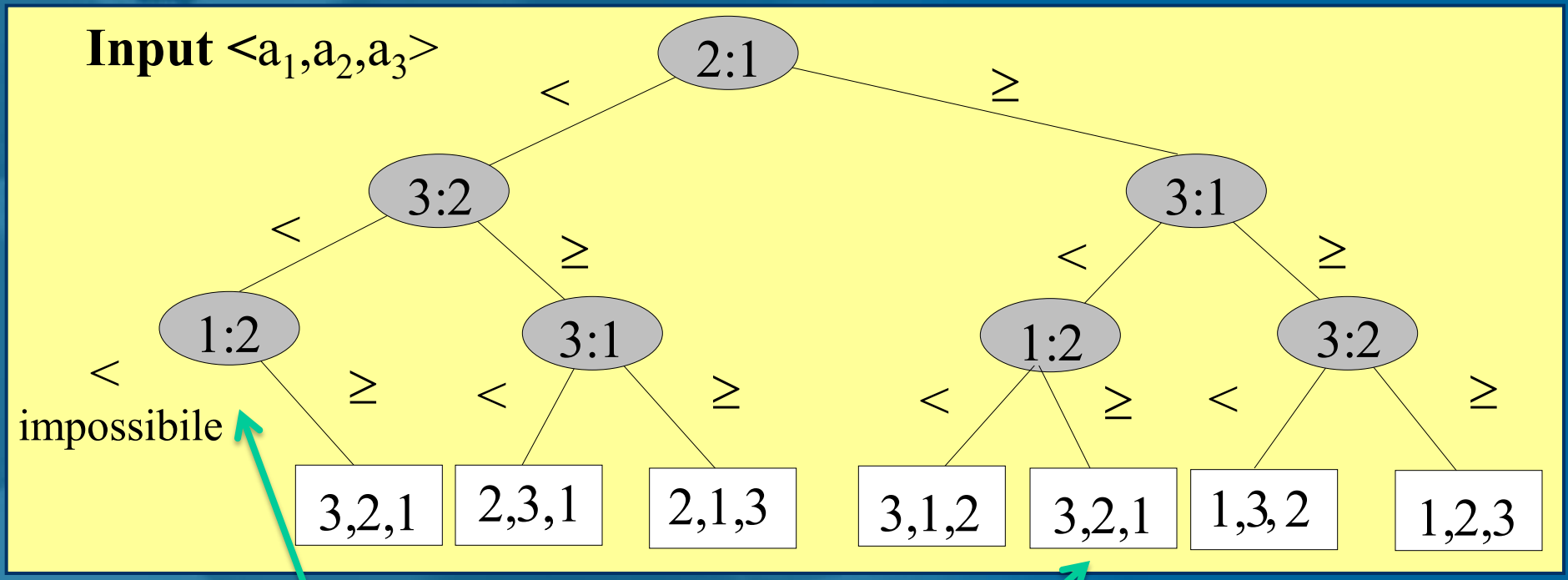
Algoritmi e Strutture Dati

Capitolo 4

Un algoritmo di ordinamento ottimo:

Il MergeSort (*)

Correzione esercizi: Albero di decisione del SS

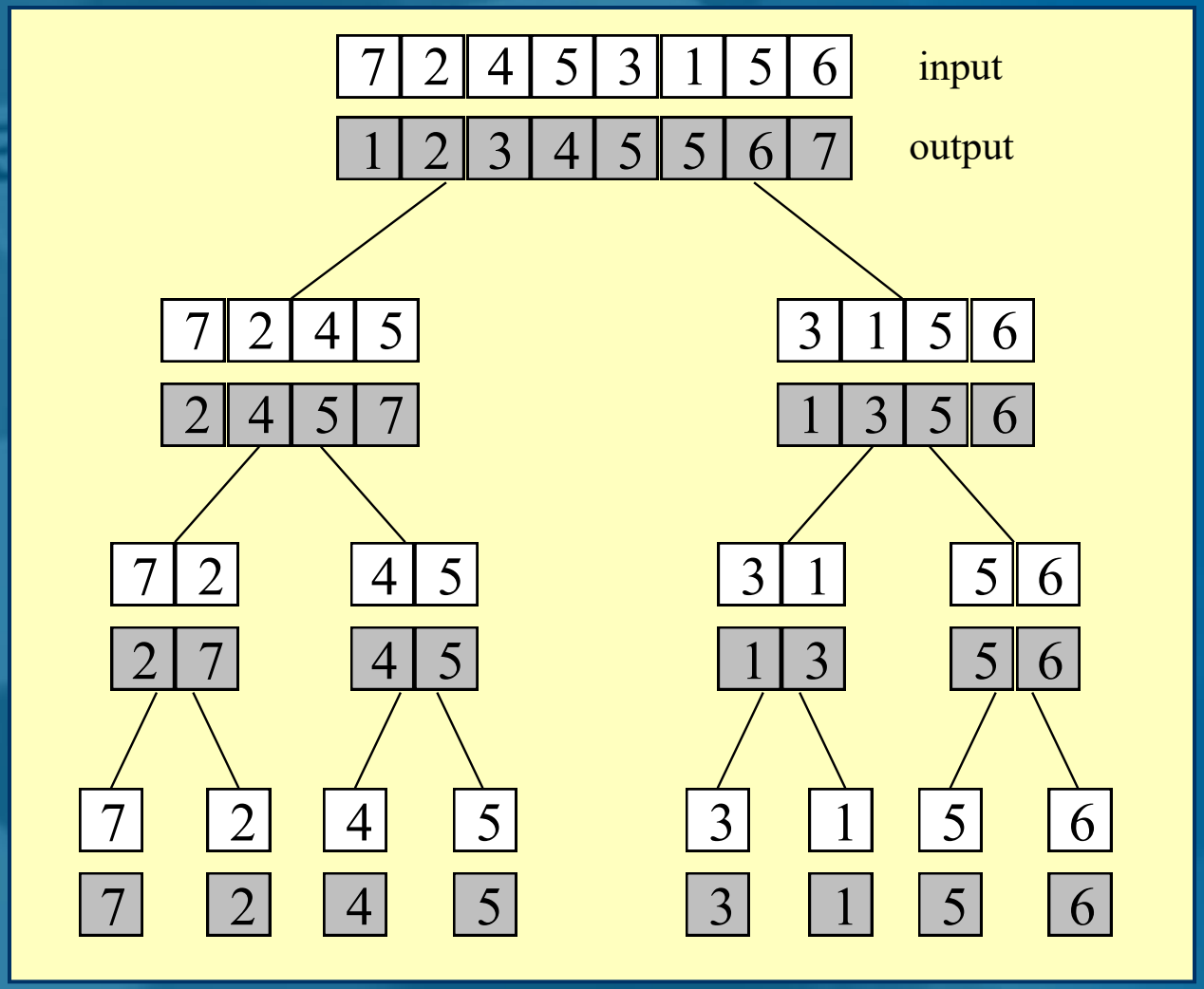


- Osservazione 1: l'albero non è strettamente binario: compare un confronto inutile, perché siamo nel ramo in cui 2<1
- Osservazione 2: tutte le foglie sono alla stessa altezza (e infatti il SS esegue sempre lo stesso numero di confronti!)
- Osservazione 3: questa permutazione <3,2,1> è raggiungibile solo nel caso in cui $a_1=a_2$

Punto della situazione

- Problema dell'ordinamento basato su confronti:
  - Lower bound - $\Omega(n \log n)$ albero di decisione
  - Upper bound – $O(n^2)$ IS 2 e 3
- Abbiamo ridotto il gap tra LB e UB da $\Theta(n)$ a $\Theta(n/\log n)$; proviamo a costruire un algoritmo ottimo (cioè a ridurre il gap a $\Theta(1)$) usando la tecnica del divide et impera:
  1. Divide: dividi l'array a metà
  2. Risolvi il sottoproblema ricorsivamente
  3. Impera: fondi le due sottosequenze ordinate

Esempio di esecuzione



Input ed output delle chiamate ricorsive

Fusione di sequenze ordinate

- Due array ordinati A e B possono essere fusi rapidamente, e questo ci consentirà di eseguire efficientemente il passo di *impera*:
  - Scorri in parallelo A e B, estraendo ripetutamente il minimo tra i due elementi puntati e copiandolo nell'array di output, finché A oppure B non diventa vuoto;
  - Copia gli elementi dell'array non ancora completamente svuotato alla fine dell'array di output.

Notazione: dato un array A e due indici $x \leq y$, denotiamo con A[x;y] la porzione di A costituita da A[x], A[x+1],…,A[y]

Algoritmo di fusione di sequenze ordinate

Merge (array A, indici i_1, f_1, f_2)

1. Sia X un array ausiliario di lunghezza f_2-i_1+1
2. i=1
3. $i_2=f_1+1$
4. **while** ($i_1 \le f_1$ e $i_2 \le f_2$) **do**
5. **if** ($A[i_1] \le A[i_2]$)
6. **then** $X[i]=A[i_1]$
7. incrementa i e i_1
8. **else** $X[i]=A[i_2]$
9. incrementa i e i_2
10. **if** ($i_1<f_1$) **then** copia $A[i_1;f_1]$ alla fine di X
11. **else** copia $A[i_2;f_2]$ alla fine di X
12. copia X in $A[i_1;f_2]$

fonde $A[i_1;f_1]$ e $A[f_1+1;f_2]$ output in $A[i_1;f_2]$

Osservazione: usa l'array ausiliario X

Costo dell'algoritmo di merge

Lemma

La procedure Merge fonde due sequenze ordinate di lunghezza n_1 e n_2 eseguendo al più $n_1+ n_2 -1$ confronti

Dim: Ogni confronto "consuma" un elemento di A.

Nel caso peggiore tutti gli elementi tranne l'ultimo sono aggiunti alla sequenza X tramite un confronto.

Il numero totale di elementi è $n_1+ n_2$. Quindi il numero totale di confronti è $n_1+ n_2 -1$. QED

Numero di confronti: $C(n=n_1+ n_2)=O(n_1+ n_2)=O(n)$
(si noti che vale anche $C(n)=\Omega(\min\{n_1,n_2\})$)

Numero di operazioni (confronti + copie)? $T(n)=\Theta(n_1+ n_2)$

MergeSort (J. von Neumann, 1945)

MergeSort(array A, indici i, f)

1. **if** (i ≥ f) **then return**
2. m = ⌊(i+f)/2⌋
3. `MergeSort`(A,i,m)
4. `MergeSort`(A,m+1,f)
5. `Merge`(A,i,m,f)

Ordina (ricorsivamente) A[i;f]

Ovviamente la chiamata principale è **Mergesort**(A,1,n)

Tempo di esecuzione

- Il numero di confronti del MergeSort è descritto dalla seguente relazione di ricorrenza:

$$C(n) = 2\,C(n/2) + \Theta(n) \qquad C(1)=0$$

(si noti che $f(n)=\Theta(n)$, in quanto il numero di confronti nelle fusioni è ovviamente $C(n)=O(n)$, ma anche $C(n)=\Omega(\min\{n_1,n_2\})=\Omega(\min\{n/2, n/2\})=\Omega(n)$)

- Usando il caso 2 del Teorema Master (infatti $a=b=2$, e quindi $f(n)\equiv\Theta(n)=\Theta(n^{\log_2 2})\equiv\Theta(n^{\log_b a})$), si ottiene

$$C(n) = \Theta(n^{\log_2 2} \log n) = \Theta(n \log n)$$

- Infine, per il tempo di esecuzione totale (confronti + copie), si ha:

$$T(n) = 2\,T(n/2) + \Theta(n) \quad T(1)=1 \Rightarrow T(n) = \Theta(n \log n)$$

Più precisamente…

1. Nel caso peggiore, il MS esegue $(n \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1)$ confronti, che corrisponde ad un numero compreso tra $(n \log n - n + 1)$ e $(n \log n + n + O(\log n))$
2. Nel caso medio, il MS esegue $(n \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1) - 0.2645 \cdot n$ confronti
3. Se modificassimo la procedura di **`Merge`** facendo un controllo preliminare tra ultimo elemento della prima sequenza e primo della seconda prima ancora di allocare memoria aggiuntiva, allora nell'ipotesi in cui l'array di input fosse già ordinato il MS eseguirebbe $\Theta(n)$ confronti; infatti, l'equazione di ricorrenza diventerebbe $C(n) = 2\,C(n/2) + \Theta(1)$, caso 1 del teorema master con soluzione $C(n) = \Theta(n^{\log_2 2}) \equiv \Theta(n)$

Osservazioni finali

- Problema dell'ordinamento: abbiamo chiuso il gap!
  - Lower bound - $\Omega(n \log n)$ albero di decisione
  - Upper bound – $\Theta(n \log n)$ Merge Sort

  Il MergeSort è dunque un algoritmo (asintoticamente) ottimo rispetto al numero di confronti eseguiti nel caso peggiore
- Il MergeSort non ordina *in loco*, e utilizza memoria ausiliaria (l'occupazione di memoria finale è pari a $\Theta(n \log n)$), in quanto viene speso $\Theta(n)$ spazio in ognuno dei $\Theta(\log n)$ livelli dell'albero della ricorsione.

Esercizi di approfondimento

- Illustrare l'evoluzione di MergeSort applicata all'array A=<31,41,59,26,41,58,11,7>

- Risolvere la relazione di ricorrenza del MergeSort (costo complessivo) utilizzando il metodo dell'iterazione