DAL TESTO:

ALGORITMI E STRUTTURE DATI

C. DEMETRESCU
I. FINOCCHI
G. ITALIANO

McGROW-HILL 2008

ORDINAMENTO IN TEMPO LINEARE

Finora abbiamo descritto vari algoritmi per ordinare in tempo $O(n \log n)$, ed abbiamo anche dimostrato una delimitazione inferiore che implica che questo tempo di esecuzione è ottimo nel modello dei confronti. Ma cosa possiamo dire al di fuori di questo modello, ad esempio se sono consentite altre operazioni a parte i confronti? I risultati dipenderanno dal tipo di dato specifico che dobbiamo ordinare: ad esempio numeri interi, numeri in virgola mobile, o stringhe di caratteri.

Ordinare n interi nell'intervallo $[1, n]$. Consideriamo innanzitutto un esempio molto semplice. Supponiamo di dover ordinare n interi *distinti* i cui valori sono tutti compresi tra 1 e n. Qual è il tempo richiesto per determinare la sequenza ordinata? La risposta è immediata: il tempo è $O(1)$, poiché la sequenza deve essere necessariamente $1, 2, 3, ..., n-1, n$. Consideriamo ora un esempio meno banale. Supponiamo che tutti i numeri siano ancora nell'intervallo $[1, n]$, ma che alcuni possano essere duplicati. L'algoritmo `integerSort` specificato in Figura 4.20 usa un array ausiliario Y per contare quante copie ci sono per ogni numero, come mostrato in Figura 4.21. Osserviamo che `integerSort` non effettua alcun confronto tra elementi, e quindi per analizzarlo dobbiamo tornare a considerare il tempo di esecuzione nell'accezione più generale.

Lemma 4.7 *L'algoritmo* `integerSort` *ordina n numeri interi in $[1, n]$ in tempo $O(n)$.*

Dimostrazione. Dopo l'esecuzione del ciclo `for` alla riga 3, $Y[i]$ contiene il numero di volte che il valore i compare in X, per ogni $i \in [1, n]$. I valori sono poi considerati in ordine crescente, e ciascuno viene riscritto su X tante volte quanto specificato nella corrispondente posizione dell'array Y. Da ciò segue la correttezza. Rispetto al tempo di esecuzione, l'unico punto non ovvio nell'analisi deriva

ATTENZIONE: QUI GLI INDICI VANNO DA 1 A n.

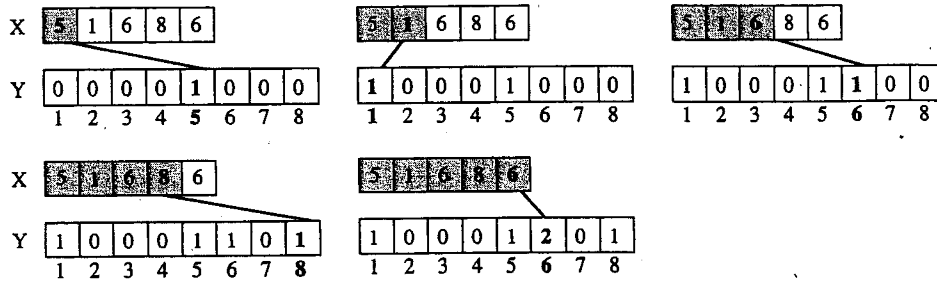
```
    algoritmo integerSort(intero n, array X)
1.     sia Y un array di dimensione n
2.     for i = 1 to n do Y[i] ← 0
3.     for i = 1 to n do incrementa Y[X[i]]
4.     j ← 1
5.     for i = 1 to n do
6.        while (Y[i] > 0) do
7.           X[j] ← i
8.           incrementa j
9.           decrementa Y[i]
```

Figura 4.20 L'algoritmo `integerSort` per ordinare n numeri interi in $[1, n]$ in tempo lineare: dopo l'esecuzione della riga 3, $Y[k]$ è il numero di volte che k compare in X.

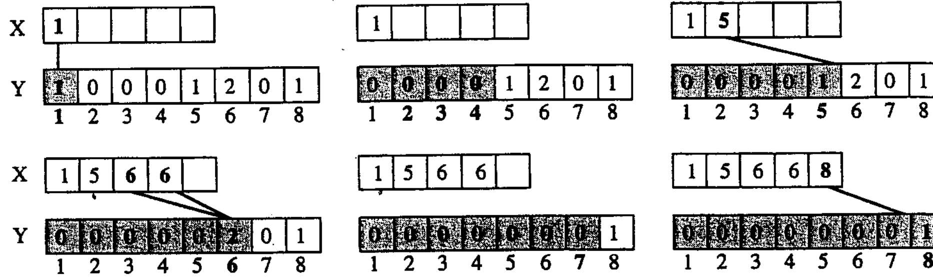
dal fatto che il terzo ciclo `for` nello pseudocodice di Figura 4.20 contiene un ciclo `while` innestato. Normalmente, quando siamo in presenza di cicli innestati, la limitazione sul tempo si ottiene effettuando il prodotto del numero di operazioni di ogni ciclo; dobbiamo invece dimostrare che questo ciclo richiede comunque tempo $O(n)$, e non $O(n^2)$. Ciò segue dalla seguente osservazione. Il ciclo interno può essere eseguito anche n volte, ma non su tutte le n iterazioni del ciclo esterno! Per convincersi di questo, è sufficiente associare al tempo necessario per incrementare gli elementi dell'array Y il tempo necessario per decrementarli: dato che ci sono soltanto n incrementi, ci saranno solo n decrementi. □

Ordinare n interi nell'intervallo $[1, k]$. L'algoritmo `integerSort` può essere facilmente adattato per trattare il caso in cui il massimo valore da ordinare, k, sia diverso da n. È infatti sufficiente usare un array ausiliario Y di dimensione k, anziché n. In tal modo potremo indicizzare k valori diversi, ma il tempo di esecuzione diventerà $O(n + k)$: ogni elemento di Y infatti deve essere inizializzato e poi letto almeno una volta durante la ricostruzione di X. Nello pseudocodice di Figura 4.20, basta sostituire k a n nelle linee 1, 2 e 5. La Figura 4.21 mostra l'algoritmo `integerSort` in azione.

Osserviamo che il tempo $O(n + k)$ è lineare nella dimensione dell'istanza se k è $O(n)$. Ma cosa accade per valori di k più grandi? Esiste un algoritmo per ordinare n numeri interi in tempo $O(n)$ se il massimo valore k è, ad esempio, $O(n^c)$, per una costante $c > 1$? Risponderemo a questa domanda, affermativamente, nel Paragrafo 4.7.



(a) Calcolo di Y



(b) Ricostruzione di X

Figura 4.21 Esecuzione dell'algoritmo `integerSort` per ordinare n interi in $[1, k]$: nell'esempio $n = 5$ e $k = 8$.