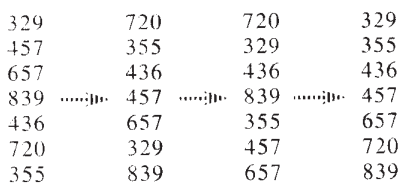

Figura 8.3 L'operazione di radix sort su una lista di sette numeri di 3 cifre. La prima colonna a sinistra è l'input. La altre colonne mostrano la lista dopo gli ordinamenti successivi rispetto alle cifre in posizioni di significatività crescente. L'ombreggiatura indica la posizione della cifra sulla quale viene effettuato l'ordinamento per generare ciascuna lista dalla precedente.

8.3 Radix sort

Radix sort è l'algoritmo utilizzato dalle macchine per ordinare le schede perforate, che adesso si trovano soltanto nei musei di calcolatori. Le schede sono composte da 80 colonne e ogni colonna può essere perforata in una delle 12 posizioni disponibili. La macchina ordinatrice può essere meccanicamente "programmata" per esaminare una particolare colonna di ogni scheda di un mazzo e poi distribuire le singole schede in uno dei 12 contenitori in funzione della posizione perforata. Un operatore può così raccogliere le schede nei vari contenitori, in modo che le schede perforate nella prima posizione si trovino sopra quelle perforate nella seconda posizione e così via.

Per le cifre decimali, sono utilizzate soltanto 10 posizioni in ogni colonna (le altre due posizioni sono utilizzate per codificare i caratteri non numerici). Un numero di d cifre occupa un campo di d colonne. Poiché la macchina ordinatrice può esaminare una sola colonna alla volta, il problema di ordinare n schede rispetto a un numero di d cifre richiede un algoritmo di ordinamento.

Intuitivamente, vorremmo ordinare i numeri in base alla loro cifra *più significativa*, ordinare ciascuno dei contenitori risultanti in modo ricorsivo e, poi, combinare ordinatamente i mazzi delle schede. Purtroppo, poiché le schede in 9 dei 10 contenitori devono essere tenute da parte mentre ordiniamo un singolo contenitore, questa procedura genera molte pile intermedie di schede di cui bisogna tenere traccia (vedere l'Esercizio 8.3-5).

Radix sort risolve il problema dell'ordinamento delle schede in una maniera contraria all'intuizione, ordinando prima le schede in base alla cifra *meno significativa*. Le schede vengono poi combinate in un unico mazzo: le schede del contenitore 0 precedono quelle del contenitore 1 che, a loro volta, precedono quelle del contenitore 2 e così via. Poi tutto il mazzo viene ordinato di nuovo in funzione della seconda cifra meno significativa e ricombinato in maniera analoga. Il processo continua finché le schede saranno ordinate rispetto a tutte le d cifre. È importante notare che a questo punto le schede sono completamente ordinate rispetto al numero di d cifre. Quindi, occorrono soltanto d passaggi attraverso il mazzo per completare l'ordinamento. La Figura 8.3 mostra come opera radix sort con un "mazzo" di sette numeri di 3 cifre.

È essenziale che gli ordinamenti delle cifre in questo algoritmo siano stabili. L'ordinamento svolto da una macchina ordinatrice di schede è stabile, ma l'operatore deve stare attento a non cambiare l'ordine delle schede mentre escono da un contenitore, anche se tutte le schede in un contenitore hanno la stessa cifra nella colonna scelta.

In un tipico calcolatore, che è una macchina sequenziale ad accesso casuale, radix sort viene talvolta utilizzato per ordinare record di informazioni con più campi chiave. Un tipico esempio è l'ordinamento delle date in base a tre chiavi:

anno, mese e giorno. Potremmo eseguire un algoritmo di ordinamento con una funzione di confronto che opera su due date così: confronta gli anni e, nel caso di pareggio, confronta i mesi; nel caso di un altro pareggio, confronta i giorni. In alternativa, potremmo ordinare le informazioni tre volte con un ordinamento stabile: prima in base al giorno, poi in base al mese e, infine, in base all'anno.

Il codice per radix sort è semplice. La seguente procedura suppone che ogni elemento nell'array A di n elementi abbia d cifre, dove la cifra 1 è quella di ordine più basso e la cifra d è quella di ordine più alto.

```
RADIX-SORT(A, d)
1  for i ← 1 to d
2      do usa un ordinamento stabile per ordinare l'array A sulla cifra i
```

Lemma 8.3
Dati n numeri di d cifre, dove ogni cifra può avere fino a k valori possibili, la procedura RADIX-SORT ordina correttamente i numeri nel tempo $\Theta(d(n+k))$, se l'ordinamento stabile utilizzato dalla procedura impiega un tempo $\Theta(n+k)$.

Dimostrazione La correttezza di radix sort si dimostra per induzione sulla colonna da ordinare (vedere l'Esercizio 8.3-3). L'analisi del tempo di esecuzione dipende dall'ordinamento stabile che viene utilizzato come algoritmo di ordinamento intermedio. Se ogni cifra si trova nell'intervallo da 0 a $k-1$ (in modo che possa assumere i k valori possibili) e k non è troppo grande, counting sort è la scelta ovvia da fare. Ogni passaggio su n numeri di d cifre richiede un tempo $\Theta(n+k)$. Poiché ci sono d passaggi, il tempo totale di radix sort è $\Theta(d(n+k))$. ■

Quando d è costante e $k = O(n)$, radix sort viene eseguito in tempo lineare. Più in generale, abbiamo una certa flessibilità sul modo in cui ripartire le singole chiavi in cifre.

Lemma 8.4
Dati n numeri di b bit e un intero positivo $r \leq b$, RADIX-SORT ordina correttamente questi numeri nel tempo $\Theta((b/r)(n+2^r))$.

Dimostrazione Per un valore $r \leq b$, consideriamo ogni chiave come se avesse $d = \lceil b/r \rceil$ cifre di r bit ciascuna. Ogni cifra è un numero intero compreso nell'intervallo da 0 a $2^r - 1$, quindi possiamo utilizzare counting sort con $k = 2^r - 1$ (per esempio, possiamo considerare una parola di 32 bit come se avesse 4 cifre di 8 bit; quindi $b = 32$, $r = 8$, $k = 2^r - 1 = 255$ e $d = b/r = 4$). Ogni passaggio di counting sort richiede il tempo $\Theta(n+k) = \Theta(n+2^r)$; poiché ci sono d passaggi, il tempo di esecuzione totale è $\Theta(d(n+2^r)) = \Theta((b/r)(n+2^r))$. ■

Dati i valori di n e b, scegliamo il valore di r, con $r \leq b$, che rende minima l'espressione $(b/r)(n+2^r)$. Se $b < \lfloor \lg n \rfloor$, allora per qualsiasi valore di $r \leq b$, si ha $(n+2^r) = \Theta(n)$. Quindi, scegliendo $r = b$, si ottiene un tempo di esecuzione $(b/b)(n+2^b) = \Theta(n)$, che è asintoticamente ottimale. Se $b \geq \lfloor \lg n \rfloor$, allora scegliendo $r = \lfloor \lg n \rfloor$ si ottiene il tempo migliore a meno di un fattore costante, cosa che possiamo verificare nel modo seguente. Scegliendo $r = \lfloor \lg n \rfloor$, si ha un tempo di esecuzione pari a $\Theta(bn/\lg n)$. Aumentando r oltre $\lfloor \lg n \rfloor$, il termine 2^r nel numeratore cresce più rapidamente del termine r nel denominatore; quindi, aumentando r oltre $\lfloor \lg n \rfloor$, si ottiene un tempo di esecuzione pari a $\Omega(bn/\lg n)$.

Se invece diminuiamo r sotto $\lfloor \lg n \rfloor$, il termine b/r cresce e il termine $n + 2^r$ resta a $\Theta(n)$.

Radix sort è preferibile a un algoritmo di ordinamento basato sui confronti, come quicksort? Se $b = O(\lg n)$, come spesso accade, e scegliamo $r \approx \lg n$, allora il tempo di esecuzione di radix sort è $\Theta(n)$, che sembra migliore di $\Theta(n \lg n)$, il tempo di esecuzione nel caso medio di quicksort. Notiamo però che i fattori costanti nascosti nella notazione Θ sono differenti. Sebbene radix sort richieda meno passaggi di quicksort sulle n chiavi, tuttavia ogni passaggio di radix sort potrebbe richiedere un tempo significativamente più lungo. La scelta dell'algoritmo di ordinamento ottimale dipende dalle caratteristiche delle implementazioni, della macchina (per esempio, quicksort spesso usa le cache hardware in modo più efficiente di radix sort) e dei dati di input. Inoltre, la versione di radix sort che usa counting sort come ordinamento stabile intermedio non effettua l'ordinamento sul posto, come invece fanno molti degli ordinamenti per confronti con tempo $\Theta(n \lg n)$. Quindi, se lo spazio nella memoria principale è limitato, potrebbe essere preferibile un algoritmo di ordinamento sul posto come quicksort.