Algoritmi di ordinamento - Esercizi

Pietro Di Lena

Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

Algoritmi e Strutture di Dati
Anno Accademico 2022/2023

Esercizio 1

- Consideriamo un array A con n interi nell'intervallo $[1, \cdots, k]$
- Analizzare il costo nel caso pessimo per ordinare A con MERGESORT e COUNTINGSORT quando
  - $k = O(1)$
  - $k = \Theta(\log n)$
  - $k = \Theta(n)$
  - $k = \Theta(n^2)$
- Quale algoritmo sceglieremmo, MERGESORT o COUNTINGSORT, assumendo di conoscere il valore asintotico di k?

Esercizio 1 - Soluzione

- Il costo nel caso pessimo di MERGESORT, $\Theta(n \log n)$, è indipendente da k, mentre il costo nel caso pessimo di COUNTINGSORT, $\Theta(n + k)$, dipende sia da n che da k.
- Il costo pessimo di COUNTINGSORT è quindi
  - $\Theta(n)$, se $k = O(1)$
  - $\Theta(n)$, se $k = \Theta(\log n)$
  - $\Theta(n)$, se $k = \Theta(n)$
  - $\Theta(n^2)$, *se* $= \Theta(n^2)$
- COUNTINGSORT è la scelta migliore in tutti i casi tranne quando $k = \Theta(n^2)$, nel qual caso MERGESORT risulta essere più efficiente.

Esercizio 2

- Dobbiamo cercare i k oggetti più costosi in una lista non ordinata contenente n oggetti
- Ideare due algoritmi per risolvere tale problema
- Assumiamo che l'algoritmo prenda in input un array di lunghezza n e posizioni i k valori più grandi in A nelle prime k posizioni dell'array (non necessariamente ordinandoli)
- Indicare qual è l'algoritmo più efficiente (nel caso pessimo) quando
  1. $k = O(1)$
  2. $k = \Theta(logn)$
  3. $k = \Theta(n)$

Esercizio 2 - Soluzione

- Una possibile soluzione è semplicemente quella di ordinare in modo inverso l'array con MERGESORT. In questo caso abbiamo un costo pessimo di $\Theta(n \log n)$, che non dipende da k. Una seconda possibile soluzione è una semplice ricerca sequenziale

```
1: function TOPK(ARRAY A[1, ··· , n], INT k)
2:     for i = 1, ··· , k do
3:         ▷ j = index of the max value in A[i, ··· , n]
4:         j = MAX(A[i, ··· , n])
5:         SWAP(A, i, j)
```

Con TOPK eseguiamo su A k ricerche, il cui costo totale è $O(nk)$:

$$\Theta(n)+\cdots+\Theta(n-k+1) = \Theta\left(\sum_{i=0}^{k-1} n-i\right) = \Theta\left(nk - \sum_{i=0}^{k-1} i\right) = O(nk)$$

- 1. Se $k = O(1)$, TOPK ha un costo lineare, meglio che MERGESORT
  2. Se $k = \Theta(\log n)$ entrambi gli algoritmi sono una valida soluzione
  3. Se $k = \Theta(n)$ la scelta migliore è utilizzare MERGESORT poichè TOPK ha un costo pessimo quadratico $O(n^2)$

Esercizio 3

- Consideriamo la seguente variante di MERGESORT
  1. Dividiamo l'array A in tre sotto-array A_1, A_2, A_3 di uguale lunghezza
  2. Richiamiamo ricorsivamente l'algoritmo su A_1, A_2, A_3
  3. Eseguiamo merge tra A_1 e A_2 e poi tra l'array ottenuto e A_3
- E' questa variante più o meno efficiente del MERGESORT originale?

Esercizio 3 - Soluzione

- Possiamo scrivere la relazione di ricorrenza della variante del MERGE-SORT e confrontare il suo costo con quello della versione originale
- Il costo di una chiamata ricorsiva dipende dalle due chiamate alla procedura MERGE
  - La prima chiamata unisce A_1 con A_2 con costo $\frac{1}{3}n + \frac{1}{3}n = \frac{2}{3}n$
  - La seconda chiamata unisce il vettore riorganizzato $A_1 + A_2$ con A_3 ed ha un costo $\frac{2}{3}n + \frac{1}{3}n = n$
  - Il costo totale è quindi $\frac{2}{3}n + n = \frac{5}{3}n = \Theta(n)$
- L'equazione di ricorrenza completa è

$$T(n) = \begin{cases} 1 & n = 0 \\ 3T(n/3) + n & n > 0 \end{cases}$$

  che, secondo il Master Theorem, ha soluzione $\Theta(n \log n)$ ($\alpha = \beta = 1$)
- Conclusione: la variante è asintoticamente equivalente alla versione originale (con qualche costante più alta a causa delle due chiamate a MERGE)

Esercizio 4

- Consideriamo un array A con $m + n$ elementi, dove solo i primi m elementi sono ordinati
- Scrivere un algoritmo di ordinamento per tale problema che abbia un costo pessimo migliore di $O((m + n) \log (m + n))$
- Analizzare il caso pessimo dell'algoritmo proposto nei seguenti casi:
  1. $n = O(1)$
  2. $n = O(\log m)$
  3. $n = O(m)$

Esercizio 4 - Soluzione

- Una idea semplice è quella di ordinare $A[m+1, \cdots, n]$ e poi usare la procedura MERGE per unire $A[1, \cdots, m]$ con $A[m+1, \cdots, n]$

```
1: function NEWSORT(Array A[1, ··· , m+n], int m)
2:     MERGESORT(A, m + 1, m + n)
3:     MERGE(A, 1, m, n)
```

che nel caso pessimo ha costo $\Theta(n \log n + m + n) = \Theta(n \log n + m)$

1. Se $n = O(1)$ allora NEWSORT ha costo pessimo $\Theta(m)$
2. Se $n = O(\log m)$ allora NEWSORT ha costo pessimo $\Theta(m)$
   - Notiamo che $\log m \cdot \log(\log m) = O(\log^2 m) = O(m)$
3. Se $n = O(m)$ allora NEWSORT has costo pessimo $\Theta(m \log m)$

Esercizio 5

- Dato un array non ordinato A con n numeri e un indice $1 \leq k \leq n$, ideare un algoritmo chet *selezioni* il k-esimo elemento in A
- Esempio: se $A = [6, 1, 2, 3, 1, 4]$ e $k = 3$ la funzione ritorna 2
- In particolare, ideare un algoritmo con costo medio $O(n)$
- Analizzare il costo dell'algoritmo anche nel caso ottimo e pessimo

Esercizio 5 - Soluzione

- Possiamo usare la funzione PARTITION di quicksort QUICKSORT

```
1: function QUICKSELECT(ARRAY A[1, ··· , n], INT k, INT p, INT r) → INT
2:     q = PARTITION(A, p, r)
3:     if q == k then
4:         return A[q]
5:     else if q < k then
6:         return QUICKSELECT(A, k, q + 1, r)
7:     else
8:         return QUICKSELECT(A, k, p, q − 1)
```

- La funzione PARTITION è la stessa che in QUICKSORT
- Il costo di PARTITION è quindi $\Theta(r - p + 1)$
  - Costo lineare sulla dimensione in input $A[p, \cdots, r]$

Esercizio 4 - Soluzione (caso ottimo e pessimo)

- Nel caso ottimo il costo coincide con quello di una sola chiamata a PARTITION (riga 3, q uguale a k dopo la prima chiamata) e quindi $\Theta(n)$. Alternativamente, il caso ottimo si verifica quando tutte le partizioni sono ben bilanciate

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$$

  Per il Master Theorem ($\alpha = 0, \beta = 1$), abbiamo un costo pari a $\Theta(n)$

- Nel caso pessimo le partizioni sono completamente sbilanciate e la ricorsione è sempre effettuata sulla partizione più grande

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n > 1 \end{cases}$$

  Usando il metodo iterativo otteniamo un costo pari a $\Theta(n^2)$

Esercizio 5 - Soluzione (caso medio)

- Per analizzare il caso medio assumiamo che tutte le partizioni siano equiprobabili e consideriamo il costo medio su tutte le possibili partizioni

$$T(n) = \begin{cases} 1 & n = 1 \\ \frac{1}{n-1} \sum_{i=1}^{n-1} T(i) + n & n > 1 \end{cases}$$

- Validiamo l'ipotesi $T(n) \leq cn$ con il metodo di sostituzione:
  - Caso base: $T(1) = 1 \leq c \cdot 1$ vera per ogni $c \geq 1$
  - Induzione: assumendo che l'ipotesi $T(i) \leq ci$ sia vera $\forall 1 \leq i < n$

$$T(n) = \frac{1}{n-1} \sum_{i=1}^{n-1} T(i) + n \leq \frac{c}{n-1} \sum_{i=1}^{n-1} i + n$$
$$= \frac{c}{n-1} \frac{n(n-1)}{2} + n = \frac{n(c+2)}{2}$$

Vera se esiste qualche $c \geq 1$ tale che $\frac{n(c+2)}{2} \leq cn \Rightarrow c \geq 2$