Selezione per distribuzione

Problema: selezione dell'elemento con rango r in un array a di n elementi distinti.

- Si vuole evitare di ordinare a
- NB: Il problema diventa quello di trovare il minimo quando $r = 1$ e il massimo quando $r = n$.

Osservazione: la funzione `Distribuzione` permette di trovare il rango del pivot, posizionando tutti gli elementi di rango inferiore alla sua sinistra e tutti quelli di rango superiore alla sua destra.

Possiamo modificare il codice del quicksort procedendo ricorsivamente nel *solo* segmento dell'array contenente l'elemento da selezionare.

La ricorsione ha termine quando il segmento è composto da un solo elemento.

Selezione per distribuzione

```
1  QuickSelect( a, sinistra, r, destra ):
2    IF (sinistra == destra) {
3      RETURN a[sinistra];
4    } ELSE {
5      scegli pivot nell'intervallo [sinistra...destra];
6      indiceFinalePivot = Distribuzione(a, sinistra, pivot, destra);
7      IF (r-1 == indiceFinalePivot) {
8        RETURN a[indiceFinalePivot];
9      } ELSE IF (r-1 < indiceFinalePivot) {
10       RETURN QuickSelect( a, sinistra, r, indiceFinalePivot-1 );
11     } ELSE {
12       RETURN QuickSelect( a, indiceFinalePivot+1, r, destra );
13     }
14   }
```

Analisi di QuickSelect mediante relazione di ricorrenza

- Caso base:
  Se il segmento sul quale opera l'algoritmo contiene un solo elemento allora l'algoritmo esegue un numero costante di operazioni per cui il costo è $\leq c$ per una certa costante c positiva.
  Se l'indice restituito da *Distribuzione*$(a, \mathit{sinistra}, \mathit{pivot}, \mathit{destra})$ è uguale a $r - 1$, l'algoritmo termina. Il costo in questo caso è dato dal costo lineare di Distribuzione più il costo costante delle altre istruzioni per cui il costo totale è $\leq c_1 n$, dove $c_1 > 0$ è una certa costante.
- Passo ricorsivo: Il costo in questo caso è dato dal costo lineare di Distribuzione più il costo costante delle altre istruzioni e il costo della chiamata ricorsiva sul segmento degli elementi minori del pivot **oppure** in quello degli elementi maggiori del pivot. Il costo in questo caso è quindi al più pari a cn (per una certa costante $c > 0$) più il costo della chiamata ricorsiva.

Analisi di QuickSelect mediante relazione di ricorrenza

Relazione di ricorrenza per il tempo $T(n)$ di esecuzione dell'algoritmo.
Indichiamo con r_p il rango del pivot

- Caso base:
  $T(n) \leq c_0$ per $n =$ e
  $T(n) \leq c_1 n$ se $r_p = r$.
- Passo ricorsivo: Ci sono $r_p - 1$ elementi a sinistra del pivot e $n - r_p$ elementi a destra, per cui $T(n) \leq \max\{T(r_p - 1), T(n - r_p)\} + cn$.

$$T(n) \leq \begin{cases} c_0 & \text{se } n = 1 \\ c_1 n & \text{se } n > 1 \text{ e } r_p = r - 1 \\ \max\{T(r_p - 1), T(n - r_p)\} + cn & \text{altrimenti} \end{cases}$$

Analisi di QuickSelect mediante relazione di ricorrenza

caso pessimo

- Il pivot è tutto a sinistra ($r_p = 1$) e $r > r_p$ oppure tutto a destra ($r_p = n$) e $r < r_p$. In entrambi i casi, la relazione diventa $T(n) \leq T(n-1) + cn$.
- Applichiamo iterativamente la relazione di ricorrenza:

$$T(n) \leq T(n-1)+cn \leq T(n-2)+c(n-1)+cn \leq \ldots \leq T(n-i)+\sum_{j=n-i+1}^{n} c\,j.$$

- Sostitutendo $i = n-1$ nell'ultima disequazione, otteniamo

$$T(n) \leq T(1) + \sum_{j=2}^{n} c\,j \leq c_0 + \sum_{j=2}^{n} c\,j = c_0 + c\,n(n+1)/2 - c = O(n^2).$$

Analisi di QuickSelect mediante relazione di ricorrenza

caso ottimo

- L'elemento di rango r è proprio il pivot ($r_p = r$), per cui si esce dalla procedura senza effettuare la ricorsione e si ha che $T(n) = O(n)$.
- Il caso ottimo si verifica anche quando ad ogni chiamata ricorsiva viene dimezzata la lunghezza del segmento in cui effettuare la selezione.

$$T(n) \leq T(n/2) + cn \leq T(n/4) + c(n/2) + cn \leq \ldots \leq T(\frac{n}{2^i}) + \sum_{j=0}^{i-1} c\frac{n}{2^j}.$$

Dopo $\log n$ applicazioni della relazione di ricorrenza otteniamo

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{2^{\log n}}\right) + \sum_{j=0}^{\log n - 1} c\frac{n}{2^j} = T(1) + cn \sum_{j=0}^{\log n - 1} \frac{1}{2^j} \\ &\leq c_0 + cn\left(\frac{1 - 1/2^{\log n}}{1/2}\right) = c_0 + 2cn(1 - 1/n) = O(n) \end{aligned}$$

Efficienza del QuickSelect randomizzato: intuizione

- Per il QuickSelect, vale un discorso analogo a quello fatto per il QuickSort
- Ci sono molte possibili scelte del pivot che fanno in modo che l'algoritmo si comporti bene.
- Scegliendo il pivot in modo random (con distribuzione di probabilità uniforme) è probabile che si scelga un pivot "ben posizionato" e cioè un pivot tale che esiste una costante $a > 1$ per cui una frazione $1/a$ degli elementi sono minori o uguali del pivot e una frazione $1 - 1/a$ degli elementi sono maggiori o uguali del pivot.
- Si può dimostrare formalmente che il QuickSelect randomizzato ha tempo di esecuzione medio $O(n)$.

Moltiplicazione di interi

- Algoritmo che usiamo comunemente ha tempo di esecuzione $O(n^2)$, dove n e` il numero di cifre di ciascun numero

```
      2345 x
      5382 =

      _____
      4690
    18760
    7035
  11725

_____________
 12620790
```

Moltiplicazione veloce di interi

Ogni numero intero w di n cifre può essere scritto come $10^{n/2} \times w_s + w_d$

- w_s indica il numero formato dalle $n/2$ cifre più significative di w
- w_d denota il numero formato dalle $n/2$ cifre meno significative.

Ad esempio 124100 può essere scritto come $10^3 \times 124 + 100$

Per moltiplicare due numeri x e y, vale l'uguaglianza

$$\begin{aligned} x\,y &= (10^{n/2}\,x_s + x_d)(10^{n/2}\,y_s + y_d) \\ &= 10^n\,x_s\,y_s + 10^{n/2}(x_s\,y_d + x_d\,y_s) + x_d\,y_d \end{aligned}$$

Decomposizione: se x e y hanno almeno due cifre, dividili come numeri x_s, x_d, y_s e y_d aventi ciascuno la metà delle cifre.

Ricorsione: calcola ricorsivamente le moltiplicazioni $x_s\,y_s$, $x_s\,y_d$, $x_d\,y_s$ e $x_d\,y_d$.

Ricombinazione: combina i numeri risultanti usando l'uguaglianza riportata sopra.

Moltiplicazione veloce di interi

- l'algoritmo esegue quattro moltiplicazioni di due numeri di $n/2$ cifre (ad un costo di $T(n/2)$), e tre somme di numeri di n cifre (a un costo $O(n)$)
- la moltiplicazione per il valore 10^k può essere realizzata spostando le cifre di k posizioni verso sinistra e riempiendo di 0 la parte destra
- il costo della decomposizione e della ricombinazione è cn

Vale la relazione di ricorrenza

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 4\,T(n/2) + cn & \text{altrimenti} \end{cases}$$

Moltiplicazione veloce di interi

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 4T(n/2) + cn & \text{altrimenti} \end{cases}$$

Assumiamo per semplicità $n = 2^k$ per un certo k e applichiamo iterativamente la relazione di ricorrenza:

$$\begin{aligned} T(n) &\leq cn + 4T(n/2) \leq cn + 4(cn/2 + 4T(n/2^2)) = cn + 2cn + 4^2 T(n/2^2) \\ &\leq cn + 2cn + 4^2(cn/2^2 + 4T(n/2^3)) = cn + 2cn + 2^2 cn + 4^3 T(n/2^3) \\ &\leq \ldots \\ &\leq cn + 2cn + 2^2 cn + \ldots + 2^{i-1} cn + 4^i T(n/2^i) \\ &= cn \sum_{j=0}^{i-1} 2^j + 4^i T(n/2^i) = cn2^i - cn + 4^i T(n/2^i) \end{aligned}$$

Ponendo $i = k = \log_2 n$ si ha $T(n) \leq cn^2 - cn + n^2 T(1) = O(n^2)$.

Moltiplicazione veloce di interi

- È possibile progettare un algoritmo più veloce?
- Abbiamo visto che $x\,y = 10^n\,x_s\,y_s + 10^{n/2}(x_s\,y_d + x_d\,y_s) + x_d\,y_d$.
- Osserviamo che sommando e sottraendo $x_s y_s + x_d y_d$ a $x_s y_d + x_d y_s$ si ha

$$\begin{aligned} x_s y_d + x_d y_s &= x_s y_d + x_d y_s + x_s y_s + x_d y_d - x_s y_s - x_d y_d \\ &= x_s y_s + x_d y_d + (x_s y_d + x_d y_s - x_s y_s - x_d y_d) \end{aligned}$$

- Poiché $x_s y_d + x_d y_s - x_s y_s - x_d y_d = -(x_s - x_d) \times (y_s - y_d)$ allora possiamo scrivere

$$x_s y_d + x_d y_s = x_s y_s + x_d y_d - (x_s - x_d) \times (y_s - y_d)$$

- quindi il valore $x_s y_d + x_d y_s$ può essere calcolato facendo uso di $x_s y_s$, $x_d y_d$ e $(x_s - x_d) \times (y_s - y_d)$
- Quindi per computare il prodotto xy sono necessarie tre moltiplicazioni e non più quattro come prima

Moltiplicazione veloce di interi

Si ha quindi la relazione di ricorrenza

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 3T(n/2) + cn & \text{altrimenti} \end{cases}$$

Assumiamo per semplicità $n = 2^k$, per un certo k, e applichiamo iterativamente la relazione di ricorrenza:

$$\begin{aligned}
T(n) &\leq cn + 3T(n/2) \leq cn + 3(cn/2 + 3T(n/2^2)) = cn + (3/2)cn + 3^2 T(n/2^2) \\
&\leq cn + (3/2)cn + 3^2(cn/2^2 + 3T(n/2^3)) = cn + (3/2)cn + (3/2)^2 cn + 3^3 T(n/2^3) \\
&\leq \cdots \\
&\leq cn + (3/2)cn + (3/2)^2 cn + \ldots + (3/2)^{i-1} cn + 3^i T(n/2^i) \\
&= cn \sum_{j=0}^{i-1} (3/2)^j + 3^i T(n/2^i) = cn \left(\frac{(3/2)^i - 1}{3/2 - 1} \right) + 3^i T(n/2^i) \\
&= 2cn((3/2)^i - 1) + 3^i T(n/2^i) = 2cn(3/2)^i - 2cn + 3^i T(n/2^i)
\end{aligned}$$

Continua nella prossima slide

Ponendo $i = k = \log_2 n$ si ha

$$\begin{aligned}
T(n) &\leq 2cn(3/2)^{\log_2 n} - 2cn + 3^{\log_2 n} T(1) \\
&= 2cn \left(2^{\log_2(3/2)}\right)^{\log_2 n} - 2cn + \left(2^{\log_2 3}\right)^{\log_2 n} T(1) \\
&= 2cn \left(2^{\log_2 n}\right)^{\log_2(3/2)} - 2cn + \left(2^{\log_2 n}\right)^{\log_2 3} T(1) \\
&= 2cn\, n^{\log_2(3/2)} - 2cn + n^{\log_2 3} T(1) \\
&= 2cn\, n^{\log_2 3 - 1} - 2cn + n^{\log_2 3} T(1) \\
&= 2cn^{\log_2 3} - 2cn + n^{\log_2 3} T(1) \\
&\leq 2cn^{\log_2 3} - 2cn + n^{\log_2 3} c_0 \\
&= O(n^{\log_2 3}) = O(n^{1,585})
\end{aligned}$$