Ordinamento per distribuzione

L'algoritmo di ordinamento per distribuzione (*quicksort*) opera nel modo seguente.

Decomposizione: se la sequenza ha almeno due elementi, scegli un elemento **pivot** e dividi la sequenza in due sotto-sequenze in modo tale che la prima contenga elementi minori o uguali al pivot e la seconda gli elementi maggiori o uguali del pivot.

Ricorsione: ordina ricorsivamente le due sotto-sequenze.

Ricombinazione: non occorre fare alcun lavoro.

```
1 QuickSort( a, sinistra, destra ):
2
3   IF (sinistra < destra) {
4     scegli pivot nell'intervallo [sinistra...destra];
5     indiceFinalePivot = Distribuzione(a, sinistra, pivot, destra);
6     QuickSort( a, sinistra, indiceFinalePivot-1 );
7     QuickSort( a, indiceFinalePivot+1, destra );
8   }
```

Ordinamento per distribuzione

```
1  Distribuzione( a, sx, px, dx ):
2    IF (px != dx) Scambia( px, dx );
3    i = sx;
4    j = dx-1;
5    WHILE (i <= j) {
6      WHILE ((i <= j) && (A[i] <= A[dx]))
7        i = i+1;
8      WHILE ((i <= j) && (A[j] => A[dx]))
9        j = j-1;
10     IF (i < j) Scambia( i, j ); i=i+1,j=j-1;
11   }
12   IF (i != dx) Scambia( i, dx );
13   RETURN i;
```

```
1 Scambia( i, j ):                                  ⟨pre: sx ≤ i, j ≤ dx⟩
2   temp = a[j]; a[j] = a[i]; a[i] = temp;
```

Distribuzione

- Data la posizione px del pivot in un segmento $a[\mathtt{sx}, \mathtt{dx}]$:
  - scambia gli elementi $a[\mathtt{px}]$ e $a[\mathtt{dx}]$, se $\mathtt{px} \neq \mathtt{dx}$
  - usa due indici i e j per scandire il segmento: i parte da sx e va verso destra e j parte da $\mathtt{dx} - 1$ e va verso sinistra fino a quando $\mathtt{i} \leq \mathtt{j}$
  - ogni volta che si ha $a[\mathtt{i}] > \mathtt{pivot}$ e $a[\mathtt{j}] < \mathtt{pivot}$, scambia $a[\mathtt{i}]$ con $a[\mathtt{j}]$ e poi riprende la scansione
  - alla fine della scansione posiziona il pivot nella sua posizione corretta

Analisi di Distribuzione

1. per stimare il tempo richiesto dal while esterno dobbiamo stimare il numero di iterazioni eseguite complessivamente dei due while interni.
2. numero totale di iterazioni del primo while interno = numero confronti tra un elemento $a[\mathtt{i}]$ con il pivot
3. numero totale di iterazioni del secondo while interno = numero confronti tra un elemento $a[\mathtt{j}]$ con il pivot
4. dopo ogni confronto di $a[\mathtt{i}]$ con il pivot o viene incrementato i (nel while stesso o nell'if) o il while esterno termina dopo al più un'iterazione
5. dopo ogni confronto di $a[\mathtt{j}]$ con il pivot o viene decrementato j (nel while stesso o nell'if) o il while esterno termina dopo al più un'iterazione
6. per i due punti precedenti si ha che ad ogni iterazione di ciascuno dei while interni, fatta al più eccezione per l'ultima, viene confrontato un nuovo elemento con il pivot. Inoltre, siccome $i \leq j$, vi è un unico elemento che può essere confrontato in entrambi i while interni (se nell'ultima iterazione $i = j$ e $a[i] > a[dx]$).
7. il numero totale di confronti con il pivot è quindi al più $n + 1$ e quindi il numero totale di iterazioni dei due while interni è complessivamente al più $n + 1$
8. tempo $O(n)$

Analisi di QuickSort mediante relazione di ricorrenza

Relazione di ricorrenza per il tempo $T(n)$ di esecuzione dell'algoritmo.

- Caso base: $T(n) \leq c_0$ per $n \leq 1$.
- Passo ricorsivo: sia r il rango dell'elemento pivot. Ci sono $r-1$ elementi a sinistra del pivot e $n-r$ elementi a destra, per cui $T(n) \leq T(r-1) + T(n-r) + cn$.

Analisi di QuickSort mediante relazione di ricorrenza

caso pessimo

- Il pivot è tutto a sinistra ($r = 1$) oppure tutto a destra ($r = n$). In entrambi i casi, la relazione diventa $T(n) \leq T(n-1) + T(0) + cn \leq T(n-1) + c'n$ per un'opportuna costante c'
- Applichiamo iterativamente la relazione di ricorrenza:

$$T(n) \leq T(n-1)+c'n \leq T(n-2)+c'(n-1)+c'n \leq \ldots \leq T(n-i)+\sum_{j=0}^{i-1} c'(n-j).$$

- Sostitutendo $i = n-1$ nell'espressione più a destra, otteniamo

$$T(n) \leq T(1)+\sum_{j=0}^{n-2} c'(n-j) \leq c_0+\sum_{j=2}^{n} c'j \leq c_0+c'(n+1)n/2-c' = O(n^2),$$

Analisi di QuickSort mediante relazione di ricorrenza

caso ottimo

- La distribuzione è bilanciata ($r = n/2$)), la ricorsione avviene su ciascuna metà
- In questa situazione, il costo è simile a quella dell'ordinamento per fusione.
- Possiamo dimostrare che il costo è di $O(n \log n)$ tempo

Efficienza del QuickSort randomizzato: intuizione

- Affinché QuickSort abbia tempo di esecuzione $O(n \log n)$ non è necessario che ogni volta il pivot sia l'elemento centrale ma è sufficiente che una frazione costante degli elementi risulti minore o uguale del pivot.
- Sia m la dimensione del segmento di array da ordinare in una certa chiamata ricorsiva. Supponiamo che il segmento venga suddiviso in due segmenti (escluso il pivot) di dimensione rispettivamente pari circa a $(m-1)(\frac{1}{d})$ e $(m-1)(1-\frac{1}{d})$, con $d > 1$ costante. Diciamo "circa" perché in realtà per un segmento occorre prendere la parte intera superiore e per l'altra quella inferiore.
- Ovviamente quanto più sono diverse le lunghezze dei due segmenti (d molto piccolo o molto grande) tanto peggiore è il comportamento dell'algoritmo.
- Supponiamo che la chiamata ricorsiva in cui la suddivisione risulta più sbilanciata, suddivida il segmento da ordinare (privato del pivot) in due parti di dimensione pari rispettivamente a circa $\frac{1}{\beta}$ e $1-\frac{1}{\beta}$ della dimensione del segmento, dove β è una costante positiva.
- Il tempo richiesto è sicuramente non più grande di quello che sarebbe richiesto se una tale suddivisione si verificasse per ogni chiamata ricorsiva su input maggiori di β. Per input di dimensione minore o uguale di β ci mettiamo nel caso peggiore, cioè quello in cui un segmento è vuoto e l'altro contiene tutti gli elementi diversi dal pivot. Il tempo sarà comunque limitato da una costante che indichiamo con c_1.

Efficienza del QuickSort randomizzato: intuizione

- Omettiamo le parti intere inferiori e superiori. Si può dimostrare che ciò non influisce sul comportamento asintotico della ricorrenza.
- Consideriamo quindi la relazione di ricorrenza

$$T(n) \leq \begin{cases} T((n-1)/\beta) + T((n-1)(1-1/\beta)) + cn & \text{se } n > \beta \\ c_1 & \text{per } n \leq \beta \end{cases}$$

  Vogliamo dimostrare che questa relazione di ricorrenza ha soluzione $O(n \log n)$ per qualsiasi costante $\beta > 1$.
- Possiamo assumere che $\beta \neq 2$ in quanto abbiamo già visto che in quel caso $T(n) = O(n \log n)$. Possiamo inoltre assumere senza perdere di generalità che $\beta > 2$, cioè che il primo segmento sia più piccolo del secondo.
- Dimostriamo con il metodo della sostituzione che $T(n) = O(n \log n)$. Per far ciò dimostreremo per induzione che esiste una costante $c' > 0$ per cui $T(n) \leq c' n \log n$ per ogni $n \geq \beta$.
- Base induzione: per $n = \beta$, si ha $T(\beta) \leq c_1$. Perché sia $T(\beta) \leq c'(\beta \log \beta)$ basta quindi scegliere c' tale che $c' \geq c_1/(\beta \log \beta)$.

Efficienza del QuickSort randomizzato: intuizione

- Ci sono quindi molte possibili scelte del pivot che fanno in modo che l'algoritmo si comporti bene.
- Questo ci suggerisce che scegliere il pivot in modo random (con distribuzione di probabilità uniforme) porta con buona probabilità a scegliere un pivot "ben posizionato" e cioè un pivot che suddivide il segmento da ordinare nel modo descritto in precedenza e ad avere un tempo di esecuzione $O(n \log n)$.
- Si può dimostrare formalmente che il QuickSort randomizzato ha tempo di esecuzione medio $O(n \log n)$.

Efficienza del QuickSort randomizzato: intuizione

- Passo induttivo. Supponiamo vera la disuguaglianza per $2, \ldots, n-1$. Si ha

$$\begin{aligned} T(n) &\leq T((n-1)/\beta) + T((n-1)(1-1/\beta)) + cn \\ &\leq c'((n-1)/\beta)\log((n-1)/\beta) + c'((n-1)(1-1/\beta)\log((n-1)(1-1/\beta)) + cn \\ &\leq c'(n/\beta)\log(n/\beta) + c'n(1-1/\beta)\log(n(1-1/\beta)) + cn \\ &= c'(n/\beta)(\log(n/\beta) - \log(n(1-1/\beta))) + c'n\log(n(1-1/\beta)) + cn \\ &= -c'(n/\beta)\log(\beta-1) + c'n\log(n(1-1/\beta)) + cn \\ &\leq -c'(n/\beta)\log(\beta-1) + c'n\log(n-1) + cn. \end{aligned}$$

- Perché risulti $T(n) \leq c' n \log n$ basta imporre $-(c'/\beta)\log(\beta-1) + c \leq 0$ che è soddisfatta per $c' \geq c\beta/(\log(\beta-1))$
- Quindi dobbiamo scegliere

$$c' = \max\{c_1/(\beta \log \beta), c\beta/(\log(\beta-1))\}.$$

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

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 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

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:

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

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

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

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 $O(n)$.