Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

Il problema dell'ordinamento: algoritmo Quicksort

Giancarlo Bongiovanni
Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

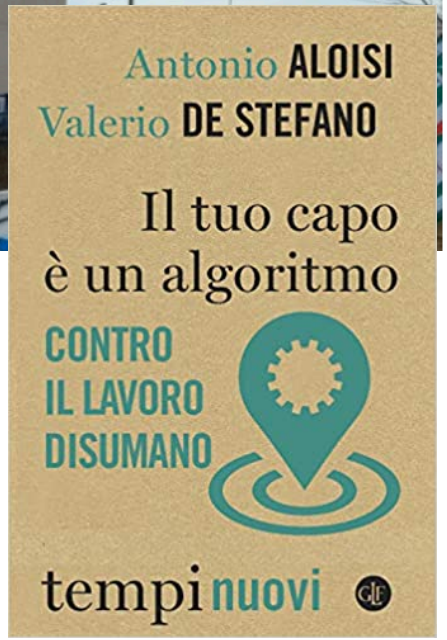
Queste dispense sono state realizzate sulla base delle slide preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

La notizia della settimana

IL RACCONTO DEL DRIVER

Amazon, i lavoratori: “Scioperiamo contro il suo algoritmo”

di **Luigi Garofalo** | 22 Marzo 2021, ore 10:25



Quicksort - 1

L'algoritmo ***quicksort*** (***ordinamento veloce***) ha costo $O(n^2)$ nel caso peggiore ***tuttavia nella pratica*** è spesso la **soluzione migliore** per grandi valori di n perché:

- il suo tempo di esecuzione **atteso** è $\Theta(n \log n)$;
- i fattori costanti nascosti sono molto piccoli;
- permette l'ordinamento "in loco".

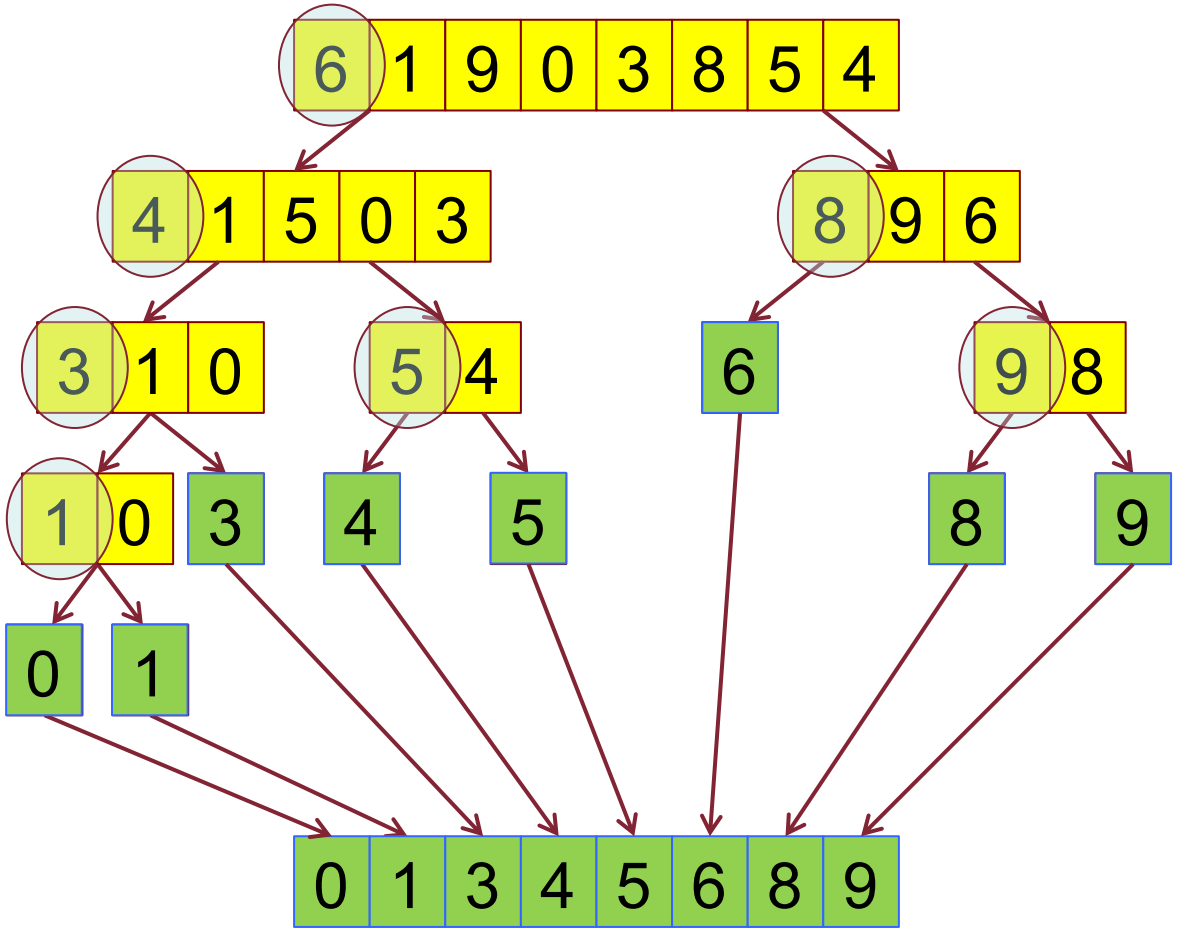
Riunisce i vantaggi del Selection sort (ordinamento in loco) e del Merge sort (ridotto tempo di esecuzione). Ha però lo svantaggio dell'elevato costo computazionale nel caso peggiore.

Quicksort - 2

Anche l'algoritmo ***quicksort*** è un algoritmo ricorsivo che adotta una tecnica algoritmica detta ***divide et impera***:

- ***divide***: nella sequenza di *n* elementi si seleziona un ***pivot*** (o **perno**). La sequenza viene quindi divisa in due sottosequenze: quella degli elementi minori o uguali del pivot, e quella degli elementi maggiori o uguali del pivot;
- ***impera***: le due sottosequenze vengono ordinate ricorsivamente;
- ***passo base*:** la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento;
- ***combina***: non occorre.

Quicksort - 3



- ***divide***: nella sequenza di *n* elementi seleziona un ***pivot***. La sequenza viene quindi divisa in due sottosequenze: quella degli elementi minori o uguali del pivot, e quella degli elementi maggiori o uguali del pivot;
- ***impera***: le due sotto-sequenze vengono ordinate ricorsivamente;
- ***passo base*:** la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento;
- ***combina***: non occorre

Quicksort - 4

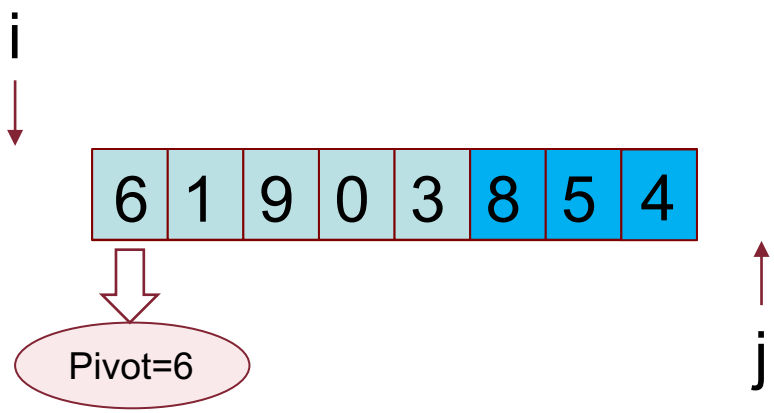
Lo pseudocodice del Quicksort è il seguente:

```
Funzione Quick_sort (A: vettore; ind_primo,
                     ind_ultimo: intero)
   if (ind_primo < ind_ultimo) then
      ind_medio ← Partiziona(A,ind_primo,ind_ultimo)
      Quick _sort (A,ind_primo,ind_medio)
      Quick _sort (A,ind_medio+1,ind_ultimo)
   return
```

In questa implementazione `ind_medio` è l'indice dell'estremo superiore della porzione di sinistra (quella contenente elementi minori o uguali del pivot). Il suo valore è ***sempre*** compreso fra 1 e (n-1)

Quicksort - 5

Prima di vedere il codice della funzione Partiziona vediamone il funzionamento su un esempio.



Quicksort - 6

```
Funzione Partiziona (A: vettore; ind_primo, ind_ultimo:
intero)
   pivot ← A[indice_primo]          //scelta arbitraria
   i ← indice_primo - 1;
   j ← indice_ultimo + 1
   while true
      repeat
         j ← j - 1
      until A[j] ≤ pivot
      repeat
         i ← i + 1
      until A[i] ≥ pivot
      if (i < j) scambia A[i] e A[j]
      else return j
```

Intermezzo: while, repeat, do…while

Nel nostro pseudocodice:

```
while (condizione)
   istruzioni
```

- La condizione si valuta **prima** dell'iterazione;
- L'iterazione si ripete finché la condizione **rimane vera**.

```
repeat
   istruzioni
until (condizione)
```

- La condizione si valuta **dopo** l'iterazione;
- L'iterazione si ripete finché la condizione **rimane falsa**.

In C l'equivalente del repeat è:

```
do {
   istruzioni;
} while (condizione);
```

- La condizione si valuta **dopo** l'iterazione;
- L'iterazione si ripete finché la condizione **rimane vera.**

Intermezzo - 2: while, repeat, do…while

Queste due iterazioni nello pseudocodice della funzione Partiziona():

```
repeat
   j ← j - 1
until A[j] ≤ pivot
repeat
   i ← i + 1
until A[i] ≥ pivot
```

Se volessimo scriverle in C diventerebbero:

```
do {
   j = j - 1;
} while (A[j] > pivot);
do {
   i = i + 1;
} while (A[i] < pivot);
```

Esercizio Impegnativo

Modificare la funzione *partiziona* dimodochè il perno occupi la posizione che avrà alla fine dell'ordinamento.

Nel nostro esempio, partendo dal vettore:

Produca il seguente partizionamento:

minori del perno **perno** **maggiori del perno**

Osservate che con questa modifica, **il perno non entrerà** più nelle successive **chiamate ricorsive**.

I pigri possono vedere la partiziona presentata dal Cormen (che però usa una logica un po' diversa rispetto a quella originale di Tony Hoare presentata qui).

Nel seguito chiameremo *partiziona'* questa procedura, che ci sarà utile a fare qualche ragionamento su Quicksort.

Quicksort - 7

Analizziamo il costo computazionale della funzione `Partiziona()`.

- Le prime tre istruzioni costano *Θ(1).*
- Il `while` ha un costo *O(n)* più un costo pari alla somma dei costi di ciò che avviene al suo interno, che è *Θ(n)* poiché:
  - ciascuna iterazione di ognuno dei due `repeat` costa *Θ(1)* e avvicina di una posizione un indice all'altro;
  - quindi complessivamente si effettuano *Θ(n)* iterazioni dei due `repeat`;
  - terminati i due `repeat` si trova un `if` ed un possibile scambio di elementi, *Θ(1)*

Dunque il costo di `Partiziona()` è *Θ(n)*.

NOTA: il valore j restituito da `Partiziona()` vale:

- 1 se il pivot è più piccolo degli altri elementi;
- (n-1) se il pivot è più grande degli altri elementi.

Però non succede mai che una delle due partizioni sia vuota!

Quicksort - 8

Valutiamo ora il costo computazionale di Quicksort:

```
Funzione Quick_sort (A: vettore;
                     ind_primo, ind_ultimo)
   if (ind_primo < ind_ultimo)                                  Θ(1)
      then
         ind_medio ← Partiziona(A,ind_primo,ind_ultimo)         Θ(n)
         Quick _sort (A,ind_primo,ind_medio)                    T(k)
         Quick _sort (A, ind_medio+1,ind_ultimo)                T(n-k)
   return                                                       Θ(1)
```

$T(n)=T(k)+T(n-k)+\Theta(n)$
$T(1)=\Theta(1)$

Che non sappiamo risolvere con alcuno dei metodi studiati…

Quicksort - 9

Possiamo facilmente derivare la soluzione per due situazioni, il caso migliore e quello peggiore.

- Caso migliore:
  è quello in cui, ad ogni passo, la dimensione dei due sotto-problemi è identica. L'equazione di ricorrenza diventa:

  $T(n) = 2T(n/2) + \Theta(n)$ che ha soluzione $T(n) = \Theta(n \log n)$
- Caso peggiore:
  è quello in cui, ad ogni passo, la dimensione di uno dei due sotto-problemi da risolvere è 1. L'equazione di ricorrenza diventa:

  $T(n) = T(n - 1) + \Theta(n)$ che ha soluzione $T(n) = \Theta(n^2)$

Prima di ritrovarci in una selva oscura

Consideriamo *partiziona'* per semplicità.

Domanda: Quante volte viene chiamata da *quickSort*?

Esattamente *n* volte: ogni elemento diventerà perno almeno una volta (alla peggio su vettori lunghi 1) e poi sparirà dalle chiamate ricorsive.

Domanda: Quali (e quanti) confronti farà *quickSort*?

Il perno verrà confrontato con **tutti gli altri elementi della partizione** (al primo giro $\mathcal{O}(n)$ confronti). Tuttavia due elementi che finiscono in partizioni diverse, non saranno **mai** confrontati tra loro.

Quindi, se la partizione è bilanciata, **elimino di colpo** $(n/2)^2$ **confronti** dagli $n^2/2$ possibili. **Nessun confronto viene ripetuto**.

Domanda: Qual è l'istanza "cattiva" che sbilancia al massimo le partizioni se io scegliessi come perno l'elemento medio del vettore $a[inf+sup/2]$?

Esercizio (istruttivo).

Quick Sort - 10

Valutiamo ora il costo computazionale nel caso medio, nell'ipotesi che il valore del pivot suddivida con uguale probabilità *1/(n-1)* la sequenza da ordinare in due sotto-sequenze di dimensioni *k* ed *n–k*, per tutti i valori di *k* tra *1* ed *n-1*.

$$T(n) = \frac{1}{n-1}\left[\sum_{k=1}^{n-1}\left(T(k)+T(n-k)\right)\right]+\Theta(n)$$

Ora, per ogni valore di *i* e *k = 1, 2, …, n – 1* il termine *T(k)* compare due volte nella sommatoria, la prima quando *k = i* e la seconda quando *k = n – i*. Valutiamo dunque il valore di:

$$T(n) = \frac{2}{n-1}\sum_{q=1}^{n-1}T(q)+\Theta(n)$$

Quick Sort - 11

Utilizziamo il metodo di sostituzione, e quindi eliminiamo innanzi tutto la notazione asintotica:

$$T(n) = \frac{2}{n-1}\sum_{q=1}^{n-1} T(q) + hn$$

$$T(1) = k$$

Per ragioni che saranno chiare tra breve, calcoliamo anche

$T(2) = \frac{2}{2-1}\sum_{q=1}^{1} T(1) + 2h = \frac{2}{1}k + 2h = 2k + 2h.$

Ipotizziamo ora la soluzione:

$$T(n) \leq an \log n$$

Quicksort - 12

Sostituiamo la soluzione innanzi tutto nel caso base.

Visto che *log 1 = 0,* non possiamo utilizzare *T(1)* e sostituiamo quindi in *T(2),* ottenendo:

$$T(2) = 2k + 2h \leq 2a \log 2 = 2a$$

che è vera per a opportunamente grande ($a \geq k + h$).

Quicksort - 13

Per il passo induttivo possiamo scrivere:

$$T(n) = \frac{2}{n-1}\sum_{q=1}^{n-1} T(q) + hn \leq \frac{2}{n-1}\sum_{q=1}^{n-1}(aq \log q) + hn =$$

$$= \frac{2a}{n-1}\sum_{q=1}^{n-1}(q \log q) + hn$$

Valutiamo ora la sommatoria $\sum_{q=1}^{n-1}(q \log q)$, che spezziamo in due:

$$\sum_{q=1}^{n-1}(q \log q) = \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1}(q \log q) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1}(q \log q)$$

$\leq \log \frac{n}{2} = \log n - 1$

$\leq \log n$

Quicksort - 14

Dunque possiamo scrivere:

$$\sum_{q=1}^{n-1}(q \log q) \leq \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q\,(\log n - 1) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \log n =$$

$$= (\log n - 1) \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q + \log n \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q =$$

$$= \log n \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q + \log n \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q =$$

$$= \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \leq \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\frac{n}{2}-1} q$$

Nota: l'ultima disuguaglianza è vera perché $\left\lceil \frac{n}{2} \right\rceil - 1 \geq \frac{n}{2} - 1$

Quicksort - 15

Ora,

$$\log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\frac{n}{2}-1} q = \log n \frac{(n-1)n}{2} - \frac{1}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2} =$$

$$= \frac{1}{2} n(n-1) \log n - \frac{1}{4}\left(\frac{n}{2} - 1\right) n \leq$$

$$\leq \frac{1}{2} n(n-1) \log n - \frac{1}{4}\left(\frac{n}{2} - 1\right)(n-1) =$$

$$= (n-1)\left(\frac{1}{2} n \log n - \frac{n}{8} + \frac{1}{4}\right)$$

Ricapitolando:

$$\sum_{q=1}^{n-1}(q \log q) \leq \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\frac{n}{2}-1} q \leq (n-1)\left(\frac{1}{2} n \log n - \frac{n}{8} + \frac{1}{4}\right)$$

Quicksort - 16

Vogliamo dimostrare che:

$$T(n) \leq \frac{2a}{n-1}\sum_{q=1}^{n-1}(q\ log\ q) + hn$$

Sapendo che:

$$\sum_{q=1}^{n-1}(q\ log\ q) \leq (n-1)(\frac{1}{2}\ n\ log\ n - \frac{n}{8} + \frac{1}{4})$$

possiamo scrivere:

$$T(n) \leq \frac{2a}{n-1}\ (n-1)(\frac{1}{2}\ n\ log\ n - \frac{n}{8} + \frac{1}{4}) + hn = an\ log\ n - \frac{an}{4} + \frac{a}{2} + hn$$

Scegliendo *a* sufficientemente grande si ha che $hn - \frac{an}{4} + \frac{a}{2} \leq 0$ e per tale *a* si avrà:

$$T(n) \leq an\ log\ n - \frac{an}{4} + \frac{a}{2} + hn \leq an\ log\ n$$

Il che ci permette di dimostrare che ***T(n) = O(n log n)***.
Poiché già sappiamo che $T(n) = \Theta(n\ log\ n)$ nel caso migliore, abbiamo che nel caso medio il Quicksort ha un costo computazionale:

$$\boldsymbol{T(n) = \Theta(n\ log\ n)}$$

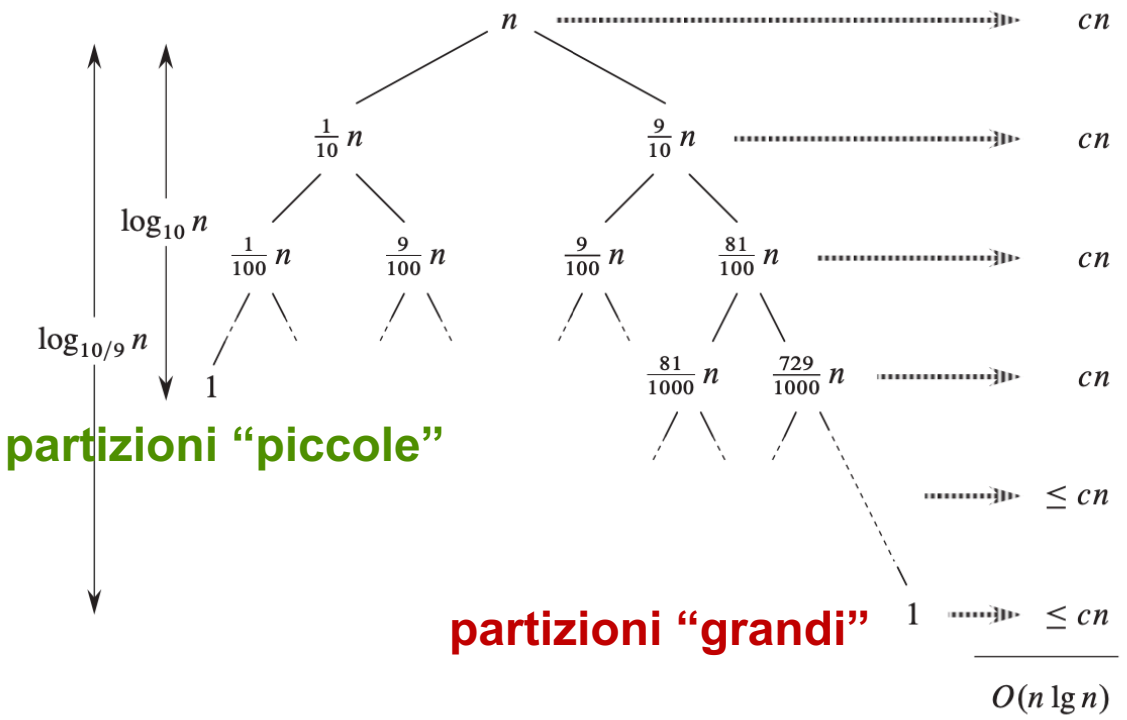
E quindi usciamo a riveder le stelle…

Domanda: Cosa succede se *quickSort* sistematicamente partiziona il vettore in due parti che hanno una lunghezza che sta in rapporto 9 a 1?

Occorre risolvere la relazione di ricorrenza:

$$T(n) = T(9n/10) + T(n/10) + cn$$

Impegnativo, ma aiutandosi col metodo dell'albero…

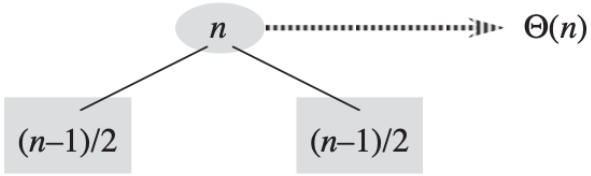


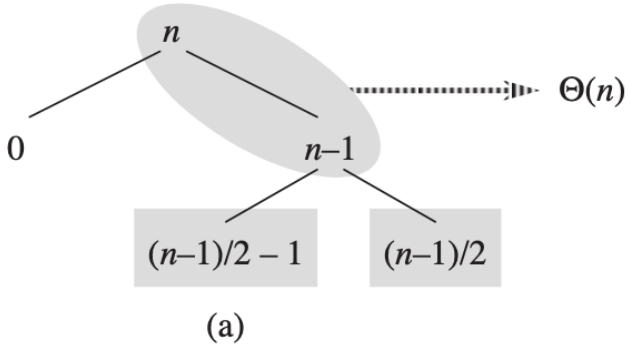
E quindi usciamo a riveder le stelle…

Domanda: Cosa succede se *quickSort* alterna una partizione bilanciata a una sbilanciata?

Niente. Divide in 3 sottovettori di cui uno vuoto e due ben bilanciati, come se **la partizione buone "assorbisse" l'errore di quella cattiva**.

effetto di un'unica partizione bilanciata





effetto di due partizioni, una cattiva e una buona

Quicksort - 17

Osservazioni

L'analisi ora fatta è valida nell'ipotesi che il valore del pivot sia equiprobabile e, quando questo è il caso, **il quicksort è considerato l'algoritmo ideale per input di grandi dimensioni**.

A volte però l'ipotesi di equiprobabilità non è soddisfatta (ad esempio quando i valori in input sono "poco disordinati") e le prestazioni dell'algoritmo degradano.

Per ovviare a tale inconveniente si possono adottare delle tecniche volte a ***randomizzare*** la sequenza da ordinare, cioè volte a disgregarne l'eventuale regolarità interna.

Tali tecniche mirano a rendere l'algoritmo indipendente dall'input, e quindi consentono di ricadere nel caso medio. Alcune di tali tecniche sono:

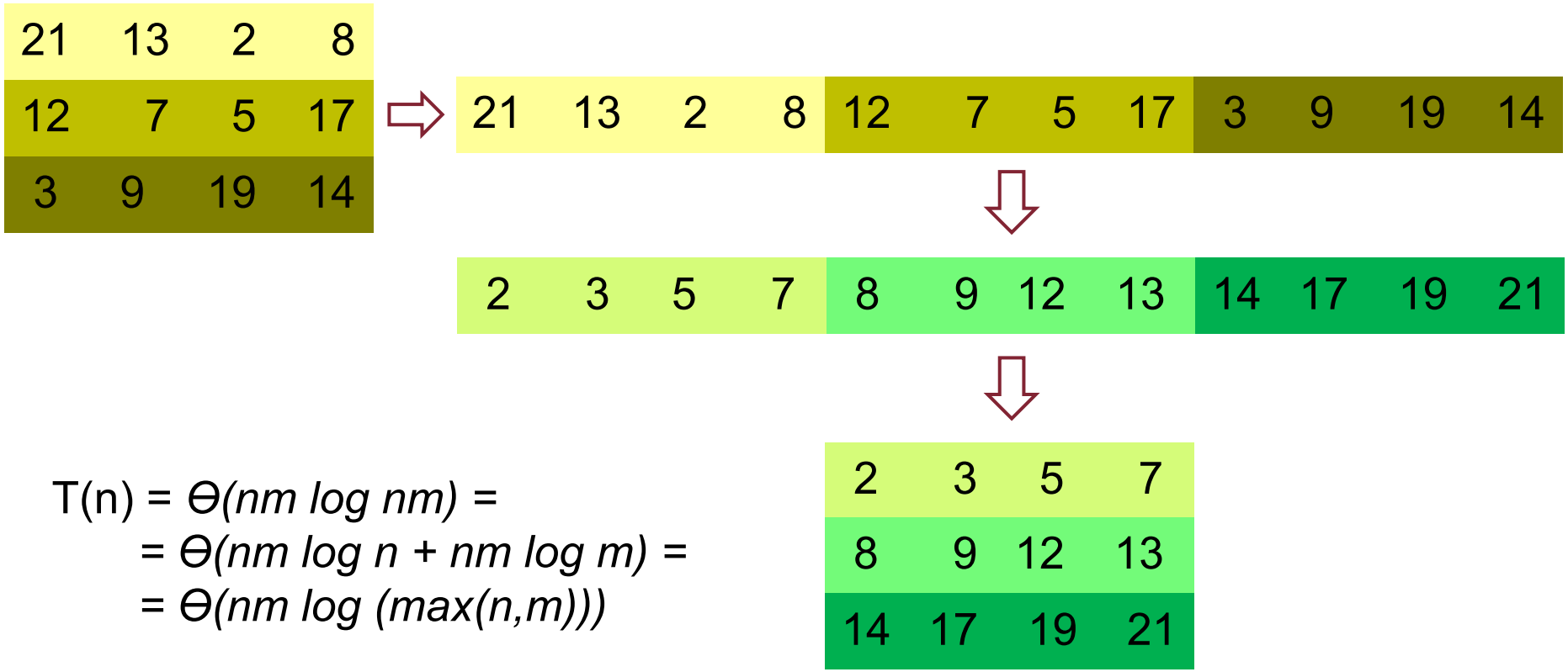
- prima di avviare l'algoritmo, alla sequenza da ordinare viene applicata una permutazione degli elementi generata casualmente;
- l'operazione di partizionamento sceglie casualmente come pivot il valore di uno qualunque degli elementi della sequenza anziché sistematicamente il valore di quello più a sinistra.

Esercizio svolto - 1

Data una matrice *m x n,* si vogliono rimescolare i suoi elementi in modo che tutti i vettori riga e tutti i vettori colonna siano ordinati in senso non decrescente (il costo computazionale dovrebbe essere non superiore a *nm log max{n,m}*).

Esercizio svolto - 2

Soluzione 1. Consideriamo la matrice come un unico lungo vettore e ordiniamolo con un algoritmo efficiente:



$T(n) = \Theta(nm \log nm) =$
$= \Theta(nm \log n + nm \log m) =$
$= \Theta(nm \log (\max(n,m)))$

Esercizio svolto - 3

Soluzione 2. Ordino prima tutte le righe separatamente e poi tutte le colonne separatamente con un algoritmo efficiente:

$$T(n)=\Theta(n\,(m \log m) + m\,(n \log n)) = \Theta(nm \log m + nm \log n)$$
$$= \Theta(nm \log (max(n,m)))$$

Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni a distanza

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi - 1

- Sia dato un vettore di lunghezza *n* contenente solo valori 0 e 2. Si progetti un algoritmo con costo computazionale lineare che modifichi il vettore in modo che tutte le occorrenze di 0 si trovino più a sinistra di tutte le occorrenze di 2.

- Si considerino i valori 0 1 2 3 4 5 6 7. Si determini una permutazione di questi valori che generi il caso peggiore per l'algortimo Quick sort.

- Calcolare il costo computazionale del Quick sort nel caso in cui il vettore contenga tutti elementi uguali ed in cui sia già ordinato da destra a sinistra.

Esercizi - 2

- Si progetti un algoritmo il più efficiente possibile per risolvere il seguente problema:
  - Data una matrice *n x n,* si vogliono rimescolare i suoi elementi in modo che tutti gli elementi posizionati al di sopra della diagonale principale siano minori o uguali di tutti gli elementi che giacciono sulla diagonale principale che, a loro volta, siano minori o uguali di tutti gli elementi posizionati al di sotto della diagonale principale.