Inside quickSort (e suo caso medio)

corso di laurea in **Matematica**

Informatica Generale, **Ivano Salvo**

Lezione **10(a)** [**31/10/23**]

SAPIENZA
UNIVERSITÀ DI ROMA

Alcune intuizioni su quickSort (1)

- **Domanda**: Quante volte viene chiamata ***`partiziona`*** in ***`quickSort`***?

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

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

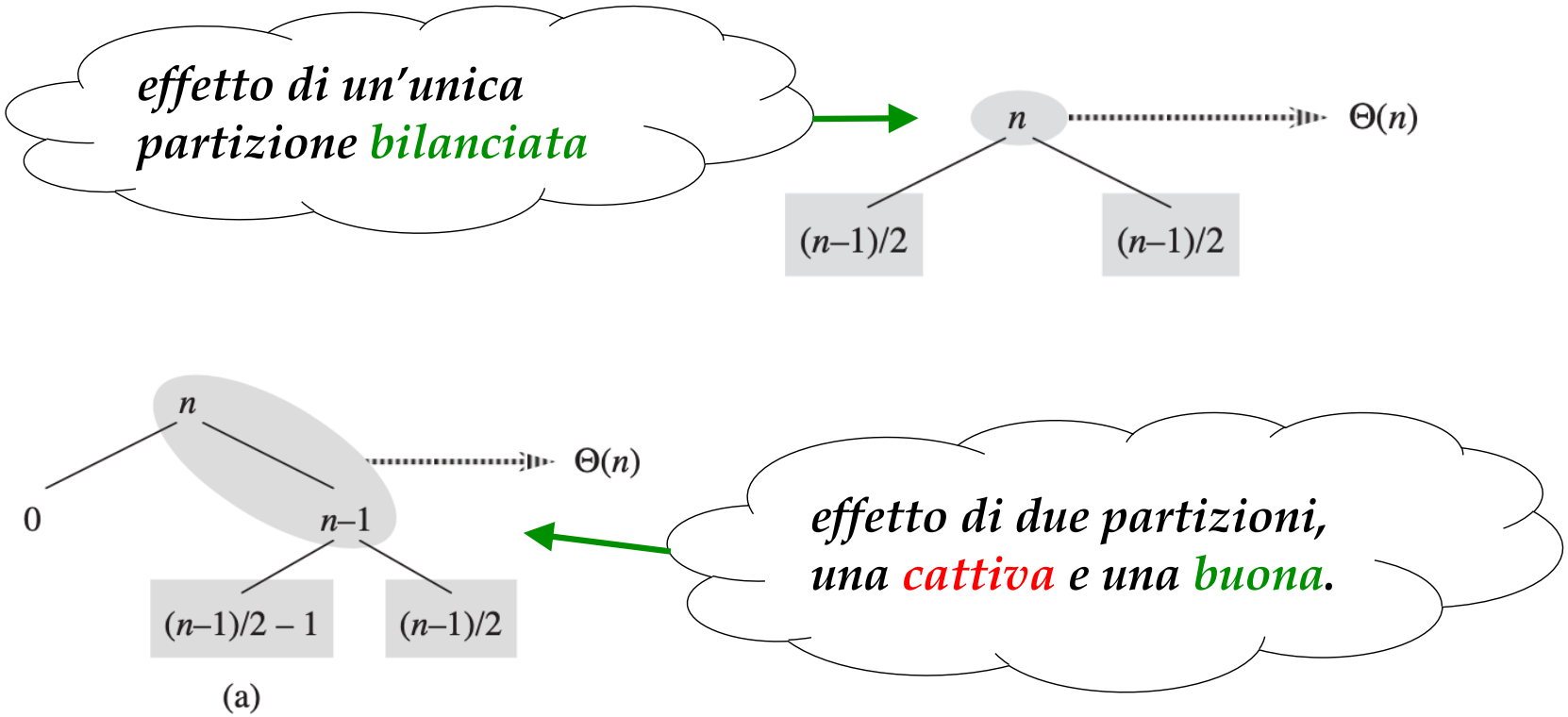
Il perno viene confrontato con **tutti gli altri elementi della partizione** (al primo giro $\mathcal{O}(n)$ confronti). Tuttavia se due elementi finiscono in partizioni diverse, non saranno **mai più** confrontati.

Quindi, se la partizione è bilanciata, **elimino di colpo** $(n/2)^2$ **confronti** dagli $n^2/2$ possibili. **Nessun confronto viene ripetuto**. Se la partizione esce sbilanciata, elimino solo n - 1 confronti.

Altre intuizioni su quickSort

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

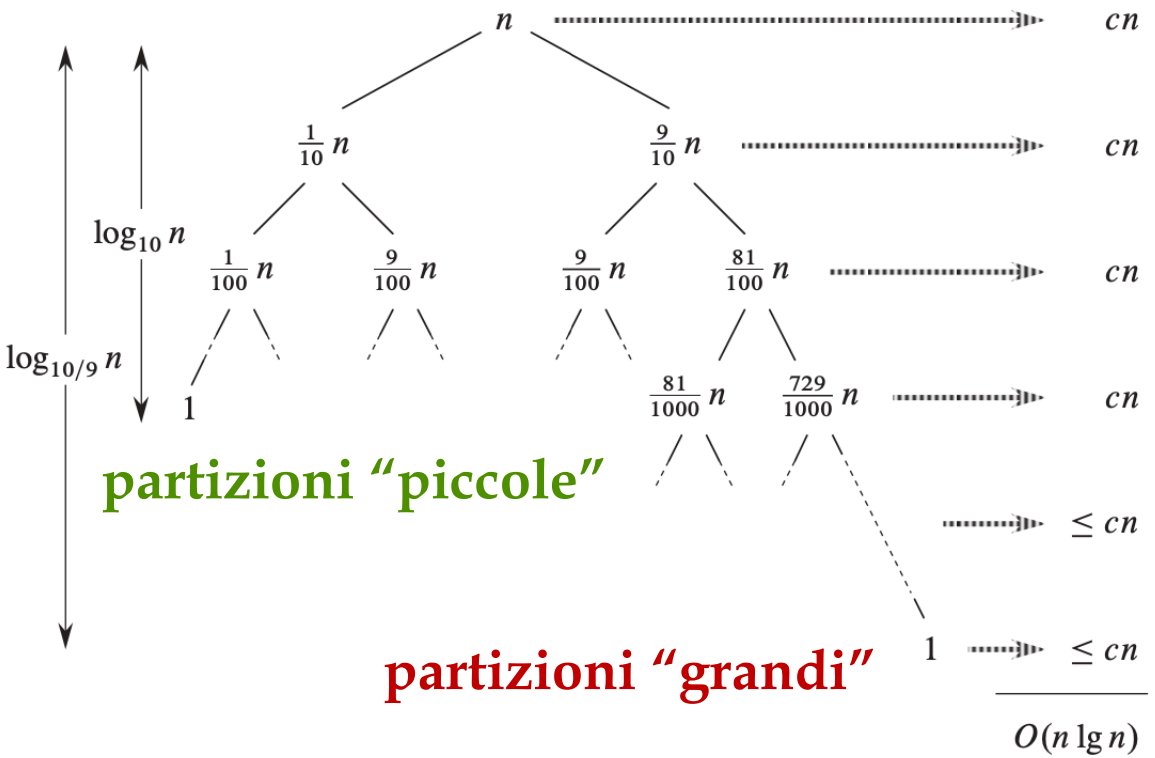
Niente. Divide in 3 sotto-vettori di cui uno vuoto e due bilanciati, come se **la partiziona buona "assorbisse" subito l'errore dovuto a quella cattiva**.



Ultime intuizioni su quickSort

Domanda: Cosa succede se ***`quickSort`*** partiziona il vettore **sistematicamente** in due parti di lunghezze 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…



k-mediana: partizioni εn / $(1 - \varepsilon)n$

Supponiamo di avere partizioni **sempre nelle stesse proporzioni** εn e $(1 - \varepsilon)n$, con $0 < \varepsilon < ½$ e di dover cercare la k-mediana sempre **nella più grande**. La relazione di ricorrenza diventa:

$$T(n) = T((1 - \varepsilon)n) + \Theta(n) \qquad T(1) = \Theta(1)$$

Applichiamo il metodo iterativo, cercando una maggiorazione:

$$\begin{aligned} T(n) &\leq cn + T((1 - \varepsilon)n) \\ &\leq cn + (1 - \varepsilon)cn + T((1 - \varepsilon)^2 n) \\ &\leq cn + (1 - \varepsilon)cn + (1 - \varepsilon)^2 cn + T((1 - \varepsilon)^3 n) \leq \\ &\leq \ldots = \\ &\leq \sum_{i=0,\ldots,k} (1 - \varepsilon)^i cn \\ &\leq cn \sum_{i=0,\ldots,\infty} (1 - \varepsilon)^i = \{\text{serie geometrica convergente}\} \\ &= cn\,(1/\varepsilon) = \mathcal{O}(n) \end{aligned}$$

Ottenendo **sempre** una **complessità lineare** che **cresce allo sbilanciarsi** delle partizioni (al decrescere di ε).

Partiziona e kMediana randomizzate

Immaginiamo ora che ***partiziona*** **scelga in modo casuale** il pivot (**algoritmo randomizzato**), ottenendo ***kMedianaRand***.

Fissiamo una qualsiasi costante ε, per comodità $\varepsilon = 1/4$ e diciamo che l'algoritmo **è in fase *j*** se la **dimensione della partiziona** in cui si sta cercando è **compresa tra** $(3/4)^j n$ **e** $(3/4)^{j+1} n$. Qual è **il numero medio** di esecuzioni in cui ***kMedianaRand*** in **ciascuna fase**?

C'è probabilità $1/2$ di scegliere un buon pivot (che divide lo spazio di ricerca in due parti $1/4\ n$ e $3/4\ n$): il valor medio delle chiamate di ***partiziona*** per passare dalla fase j alla fase j+1 è $\sum_{i \in [0, k)} i \cdot 1/2^i \leq \sum_{i \in [0, \infty)} i \cdot 1/2^i =$ **2** (vedi problema del **confrontatore binario**).

Qual è il numero di passi fa ***kMedianaRand*** randomizzata? Sarà la somma di $X_1, \ldots, X_h$, dove X_j è il numero di passi fatti in fase j.

Il **valor medio** di passi fa ***kMedianaRand*** sarà $E[\sum_j X_j] =$ (per linearità del valor medio) $\sum_j E[X_j] = \sum_j 2c\,(3/4)^j n = 2cn \sum_j (3/4)^j =$ ***8cn*** (ancora serie geometrica convergente a $1/(1 - 3/4) = 4$). Abbiamo quindi:

Teorema. *Il valor medio atteso della costo computazionale di* `kMedianaRand` *è* $O(n)$.

quickSort randomizzato

Modifichiamo **`quickSort`** come segue: si partiziona random finché non c'è un **perno buono**, che divida il vettore in almeno $^1/_4\,n$ - $^3/_4\,n$.

Questa operazione ha senso se il vettore ha almeno 4 elementi, quindi consideriamo **caso base i vettori di al più 3 elementi**.

Se il perno non è "buono", **buttiamo via la partizione** e ripetiamo la stessa chiamata a **`quickSort`** con gli stessi parametri (osservare che questa procedura **non termina**, con **probabilità 0**).

```
def quickSortRand(v, inf, sup):
    if sup - inf <= 3: # caso base: 3 elem.
            ordina3(v)
            return
    m = partizionaRand(v, inf, sup)
    if inf + (sup – inf) / 4 > m
          or inf + 3 * (sup – inf) / 4 < m:
       # butto via tutto e rifaccio
       quickSortRand(v, inf, sup)
    quickSortRand(v, inf, m)
    quickSortRand(v, m+1, sup)
```

Conseguenze: caso medio quickSort

Osserviamo che:

- come prima, in media **troviamo un buon perno ogni 2 tentativi**
- ogni **valore** giocherà da **perno 1 sola volta**
- in fase j, ci sono al più $(4/3)^{j+1}$ istanze di dimensione $(3/4)^j n$

La complessità attesa della fase j è globalmente $\mathcal{O}(n)$ [come nell' albero con partizioni 1/10 – 9/10] e si attraversano **al più $\log_{4/3} n$ fasi** cioè $\Theta(\log n)$, quindi il **tempo di esecuzione atteso** è $\Theta(n \log n)$.

Osservate che qualsiasi versione **randomizzata** o **deterministica** su **vettore** con valori distribuiti uniformemente **causale** **che però non butta via le partizioni calcolate** ci mette un **tempo inferiore** (è comunque meglio una cattiva partizione, che rifare la partizione).

Quindi, **`quickSortRand`** (che è essenzialmente uno **strumento concettuale**) stabilisce un **limite superiore** al **valor medio** del tempo di esecuzione di **`quickSort`**. Quindi:

Teorema. *Il valor medio atteso del costo computazionale di* `quickSort` *è* $\mathcal{O}(n \log n)$.

That's all Folks!

corso di laurea in **Matematica**

Informatica Generale, **Ivano Salvo**

Lezione **10(a)** [**31/10/22**]

SAPIENZA
UNIVERSITÀ DI ROMA