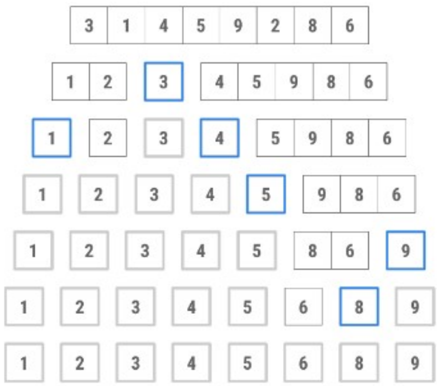
COMPLESSITA' QUICK SORT

QuickSort



Idea base di quickSort: partizionare rispetto a un perno ("pivot")

- Prendi un elemento dell'array (il "pivot")
- Partiziona gli elementi della sottosequenza su cui quickSort viene chiamata in modo tale che
  - quelli a sinistra del pivot siano minori del pivot
  - quelli a destra siano maggiori (o uguali) del pivot
- Dopo queste operazioni, il pivot è nella sua posizione finale
- Richiama quickSort sulle due parti della sottosequenza ottenute

complessità

L'efficienza del quick sort dipende da come scegliamo il pivot

- CASO MIGLIORE:
  $\Theta(n \log n)$ → è un caso ipotetico in quanto si sceglie come pivot il mediano che però non possiamo sapere senza prima riordinare la sequenza. Essendo che il pivot è sempre l'elemento mediano, l'array verrà diviso dalla partition in due sottosequenze di lunghezza circa $n/2$. Quindi, come nel caso del merge sort, ottengo un albero binario bilanciato che ha altezza $\log_2(n)$, che è il numero di iterazioni della funzione ricorsiva
- CASO MEDIO:
  $\Theta(n \log n)$ →
  - n deriva dal numero di operazioni svolte ad ogni livello dell'albero della ricorsione: al livello J si effettuano 2^J chiamate partition, ciascuna su una porzione di array lunga $n/2^J$. Ciascuna di queste chiamate ha complessità lineare nella dimensione della porzione di array su cui viene chiamata, quindi $O(n/2^J)$. Ad ogni livello faccio $2^J \cdot O(n/2^J)$ operazioni che determinano $O(n)$
  - $\log n$ deriva dal numero di livelli dell'albero della ricorsione
- CASO PEGGIORE:
  $\Theta(n^2)$ → quando si sceglie come pivot l'elemento minore o maggiore della sequenza su cui partition viene chiamata o quando la sequenza è già ordinata. Questa è data dalla partition poiché a livello 0 costerà n, a livello 1 costerà $n-1$, a livello 2 costerà $n-2$ e così via, quindi si ha che $n+(n-1)+(n-2)+(n-3)\ldots = n \cdot \frac{(n+1)}{2} = n^2$. Quindi questo calcolo si sviluppa nella sommatoria per "i" che va da 1 a n in $O(n^2)$
- QUICKSORT RANDOM:
  Per ogni array di dimensione n, il tempo di esecuzione del quicksort randomizzato nel caso medio è $\Theta(n \log n)$

quick sort esempio esame

- QUICKSORT CASO PEGGIORE + PIVOT MAGGIORE / MINORE
  Sequenza: 30 5 20 15 50 45 40 35
  Nel caso peggiore quicksort ha complessità $\Theta(n^2)$ e si cade in questo caso quando si sceglie come pivot l'elemento minore o maggiore della sequenza su cui partition viene chiamata.
  Infatti per calcolare la complessità si sommano le operazioni fatte ad ogni chiamata ricorsiva. Al primo "giro" (livello 0 dell'albero) si effettuano n operazioni, al secondo (livello 1) se ne fanno $n-1$ e via dicendo fino ad arrivare all'ultimo livello che esegue 1 sola operazione.
  Questo calcolo si sviluppa nella sommatoria per "i" che va da 1 a n: $n+(n-1)+(n-2)+(n-3)\ldots = n^2$
- ESEMPIO SU SEQUENZA DATA:
  N.B. i valori tra parentesi sono già nella posizione finale e non verranno considerati alla

- **ESEMPIO SU SEQUENZA DATA:**

N.B. i valori tra parentesi sono già nella posizione finale e non verranno considerati alla prossima chiamata ricorsiva.

1) Scelgo come pivot o il minore o il maggiore → 15 e sistemo i minori a sinistra e i maggiori a destra

SEQUENZA: (15) 30 25 20 50 45 40 35

Il pivot è nella posizione finale (in questo caso primo elemento) chiamo quick sort su sequenza restante.

2) Scelgo come pivot o il minore o il maggiore → 50 e sistemo i minori a sinistra e i maggiori a destra

SEQUENZA: (15) 30 25 20 45 40 35 (50)

Il pivot in questo caso è nella posizione finale (ultimo elemento) chiamo quick sort su sequenza restante

- **QUICK SORT CASO MIGLIORE + PIVOT SEMPRE ULTIMO ELEMENTO:**

La complessità nel caso migliore è: $\Theta(n \log n)$

Questo però è un caso ipotetico in quanto si sceglie come pivot sempre l'elemento mediano, che però non possiamo sapere senza prima riordinare la sequenza.

L'array verrà quindi diviso dalla partition in due sottosequenze di lunghezza circa $n/2$ ottenendo quindi, come nel caso del mergesort, un albero binario che ha altezza $\log_2(n)$ ossia il numero di iterazioni della funzione ricorsiva.

Ad ogni livello J dell'albero la partition viene effettuata su $(2^J)\cdot(n/2^J)$ elementi e ha quindi complessità $\Theta(n)$. La complessità finale sarà dunque $\Theta(n \cdot \log_2(n))$

SIMULAZIONE QUICK SORT CON PIVOT SEMPRE ULTIMO ELEMENTO

SEQUENZA: 10 20 30 40 50 60 70 80 90 100

Dato che la sequenza è già ordinata e che il pivot scelto è sempre l'ultimo elemento, sappiamo già di essere nel caso peggiore.

giro n	pivot	sequenza
1	100	10 20 30 40 50 60 70 80 90 (100)
2	90	10 20 30 40 50 60 70 80 (90) (100)
3	80	10 20 30 40 50 60 70 (80) (90) (100)
⋮	⋮	⋮
10	10	(10) (20) (30) (40) (50) (60) (70) (80) (90) (100)

• quick sort con pivot per "magia"

SEQUENZA: 12 7 1 2 3 4 9 6 8 10

in blu il pivot, in rosso gli elementi nella posizione finale

12 7 1 2 3 4 9 6 8 10

(1 2 3) 4 (12 7 9 6 8 10)

(1) 2 (3) 4 (7 6 8) 9 (12 10)

1 2 3 4 6 7 8 9 10 12

1 2 3 4 6 7 8 9 10 12

• come scegliere un buon pivot

Idea scelta casuale del pivot:

Ad ogni chiamata ricorsiva, scegliamo come pivot uno dei numeri dell'array a caso. Con questo approccio alla scelta del pivot, abbiamo una versione di quick sort randomizzata, nella quale anche se due esecuzioni diverse sullo stesso input possono svolgersi in modo diverso, i risultati delle due esecuzioni deve essere lo stesso.

Per ogni array di dimensione n, il tempo di esecuzione del quick sort randomizzato nel caso medio è $O(n \log n)$

esempio domande esame quick sort aulaweb

1) **SCRIVERE LA COMPLESSITÀ DEL QUICK SORT NEL CASO PEGGIORE**

complessità: $\Theta(n^2)$

Si ricade nel caso peggiore quando partition chiamata sull'array tra "inizio" e "fine" seleziona sempre come pivot l'elemento maggiore o minore tra quelli compresi tra "inizio" e "fine".

La complessità si calcola nel seguente modo:

Si sommano le operazioni effettuate ad ogni livello dell'albero delle chiamate ricorsive. Tali operazioni sono dovute alla chiamata di partition, che al livello 0 vale n (effettua n operazioni), al livello 1 si fanno n-1 operazioni, livello 2 n-2 e così via fino ad arrivare all'ultimo livello nel quale viene effettuata una sola operazione. Questo calcolo si sviluppa nella sommatoria per "i" che va da 1 a n: $n+(n-1)+(n-2)+(n-3)\ldots = n^2 \rightarrow \Theta(n^2)$

2) **5 6 8 7 9 10 3 4 2 1**

in blu il pivot, in rosso gli elementi nella posizione finale

5 6 8 7 9 10 3 4 2 1

(5 6 3 4 2 1) 7 (8 9 10)

(2 1) 3 (4 5 6) 7 8 9 10

1 2 3 4 5 6 7 8 9 10

→ Corrisponde al caso migliore di quick sort in quanto ad ogni chiamata di partition, il pivot è il mediano degli elementi nella porzione dell'array tra inizio e fine. Ovviamente questo caso è teorico in quanto non si può calcolare l'elemento mediano senza riordinare prima gli elementi.

complessità: $\Theta(n \log n)$