Soluzione Algoritmi di Ordinamento

QuickSort Caso Peggiore + Pivot maggiore/minore

> **(1/2 punteggio)**: Si scriva quanto vale la complessità di **quicksort** nel caso peggiore e si illustri dettagliatamente quando si ricade nel caso peggiore e come si calcola la complessità di quicksort nel caso peggiore, senza fare uso del Teorema Master (non è parte di asd). Rispetto alla sequenza riportata sotto, si illustrino i primi due pasaggi dell'esecuzione di QS nel caso peggiore.

Sequenza: 30 25 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 (o livello dell'albero). Al primo "giro" (livello 0 dell'albero) si effettuano **n** operazioni, al secondo (lv. 1) se ne faranno **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) ... = n^2

Esempio su sequenza data:

> 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 SX e i maggiori a DX:
   - Seq = (15) 30 25 20 50 45 40 35

   Il pivot è nella posizione finale (in questo caso primo elemento), chiamo QS su sequenza restante
2. Scelgo come pivot o il minore o il maggiore —> 50 e sistemo i minori a SX e i maggiori a DX:
   - Seq = (15) 30 25 20 45 40 35 (50)

   Il pivot è nella posizione finale (in questo caso ultimo elemento), chiamo QS su sequenza restante
3. Scelgo come pivot o il minore o il maggiore —> 45 e sistemo i minori a SX e i maggiori a DX:
   - Seq: (15) 30 25 20 40 35 (45) (50)

   Il pivot è nella posizione finale (in questo caso penultimo elemento), chiamo QS su sequenza restante
4. Scelgo come pivot o il minore o il maggiore —> 20 e sistemo i minori a SX e i maggiori a DX:
   - Seq: (15) (20) 30 25 40 35 (45) (50)

   Il pivot è nella posizione finale (in questo caso ultimo elemento), chiamo QS su sequenza restante
5. Scelgo come pivot o il minore o il maggiore —> 25 e sistemo i minori a SX e i maggiori a DX:
   - Seq: (15) (20) (25) 30 40 35 (45) (50)

   Il pivot è nella posizione finale (in questo caso ultimo elemento), chiamo QS su sequenza restante
6. Scelgo come pivot o il minore o il maggiore —> 30 e sistemo i minori a SX e i maggiori a DX:
   - Seq: (15) (20) (25) (30) 40 35 (45) (50)

   Il pivot è nella posizione finale (in questo caso ultimo elemento), chiamo QS su sequenza restante
7. Scelgo come pivot o il minore o il maggiore —> 35 e sistemo i minori a SX e i maggiori a DX:

- Seq: (15) (20) (25) (30) (35) 40 (45) (50)

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

8. Farebbe forse ancora un giro ma abbiamo capito ...

QuickSort 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 lunghezze 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 $(2^j) * (n/2^j)$ elementi e ha quindi complessità Θ(n).
La complessità finale sarà dunque $\Theta(n * log_2(n))$.

Simulazione QS con Pivot sempre ultimo elemento

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

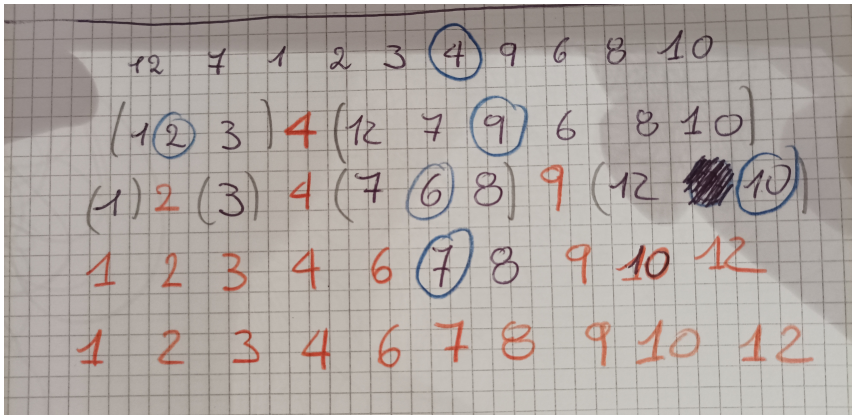
Dato che la sequenza è gia ordinata e che il pivot scelto è sempre l'ultimo elem, sappiamo già di essere nel caso **peggiore**. (I valori tra parentesi sono quelli nella posizione finale)

Giro n.	Pivot	Sequenza
1	100	10 20 30 40 50 60 70 80 90 (100) [a SX val<100; a DX val>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)
4	70	10 20 30 40 50 60 (70) (80) (90) (100)
...	...	...
10	10	(10) (20) (30) (40) (50) (60) (70) (80) (90) (100)

QuickSort con Pivot per "magia"

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

In blu il pivot; in rosso gli elementi nella posizione finale.



Come scegliere un buon pivot?

Idea scelta casuale(rand) 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 Quicksort randomizzata, nella quale anche se due esecuzioni diverse sullo stesso input possono svolgersi in modo diverso, il risultati delle due esecuzioni deve essere lo stesso.

Per ogni array di dimensione n, il tempo di esecuzione del quicksort randomizzato nel caso medio è **O(nlogn).**

Mergesort

> **(1/2 punteggio)**: Si simuli un' esecuzione della chiamata di **mergesort** sulla sequenza riportata sotto. Questa situazione coincide con il caso migliore, peggiore o medio ? Qual è la complessità di mergesort in questa situazione?

Sequenza: 50 45 40 35 30 25 20 15

Simulazione di chiamata mergesort sulla sequenza data:

0. 50 45 40 35 30 25 20 15
1. 50 45 40 35 | 30 25 20 15
2. (50 45) | (40 35) | (30 25) | (20 15)
3. 50 | 45 | 40 | 35 | 30 | 25 | 20 | 15
4. (45 50) | (35 40) | (25 30) | (15 20)
5. 35 40 45 50 | 15 20 25 30
6. 15 20 25 30 35 40 45 50

Vale $\Theta(n\ log\ n)$sia nel caso migliore che peggiore, infatti **non essendo un algoritmo adattivo**, effettua tutte le chiamate ricorsive necessarie, e tutti i confronti, anche nel caso in cui l'array preso in considerazione sia già ordinato.

La complessità dell' algoritmo deriva dal calcolo dell'espressione: **n° livelli * costo operazioni di ogni livello** . E ancora, il costo operazioni singolo livello si calcola come: **numero problemi * dimensione problema**.

Livello (j)	Numero problemi (2^j)	Dimensione problema (n/2^j)	Sequenza
0	1	n	50 45 40 35 30 25 20 15
1	2	n/2	S1: 50 45 40 35 S2: 30 25 20 15
2	4	n/4	S1: 50 45 S2: 40 35 S3: 30 25 S4: 20 15
3	8	n/8	S1: 50 S2: 45 S3: 40 S4: 35 S5: 30 S6: 25 S7: 20 S8: 15
4	16	n/16	Ogni sequenza è formata da un solo elemento e da qui in poi si "torna in su" nell'albero della ricorsione, ogni volta sistemando minore e maggiore

- **Costo operazioni di ogni livello:**

  Ad ogni livello **j** si hanno 2^j sottoproblemi di tipo `merge`, ognuno lungo $n/2^j$ e quindi risolvibile in $\theta(n/2^j)$. Per conoscere il costo di ogni livello, bisogna **moltiplicare il costo di `merge` per il numero di volte che viene chiamato:**

  $2^j * n/2^j = n \rightarrow \theta(\mathbf{n})$ (costo di ogni livello di ricorsione).

- **Numero di livelli:**

  Sapendo che all'ultimo livello della ricorsione, i sottoproblemi assumono dimensione **1**, e che su ogni livello "j", i sottoproblemi sono di dimensione $n/2^j$, per quale j si ha che $n/2^j = 1$? ——> $j = log_2(n)$

  Quindi il numero di livelli è: **log2 n + 1** [+1 perche si parte dal livello zero]

In conclusione il **costo di MergeSort** è dato da: $\theta(\mathbf{n}) \times \mathbf{log_2\ n} \rightarrow \theta(\mathbf{n\ log\ n})$ sia nel caso migliore che nel caso peggiore.