Confronto tra algoritmi

Cosimo Michelagnoli

June 11, 2020

Abstract

In questa relazione vengono presentate le principali differenze tra due algoritmi di ordinamento, Insertion_Sort e Quick_Sort. I due algoritmi sono stati testati fornendo loro sequenze in input di dimensione e tipologia diverse per valutare le loro performance secondo i tempi di esecuzione, a parità di hardware & software.

1 Algoritmi

1.1 Insertion_Sort

L'algoritmo Insertion_Sort fa parte degli algoritmi iterativi. Questa tipologia di algoritmi itera una stessa azione più volte fino al verificarisi di una determinata condizione. L'Insertion_Sort è in particolare un algoritmo incrementale che preleva un generico elemento in posizione *k+1* e lo inserisce nella sua corretta posizione all'interno della sotto sequenza di lunghezza *k*.

1.2 Quick_Sort

Questo algoritmo appartiene alla famiglia degli algoritmi ricosivi(*divide et impera*). Questa tipologia di algoritmi divide il problema principale in sottoproblemi computazionalmente più semplici per poi combinare le soluzioni ottenute e risolvere il problema originario. Il Quick_Sort divide la sequenza in due sottosequenze ricorsivamente, fino ad avere sequenze di soli due elementi facile da ordinare. Infine, nell'algoritmo viene utilizzata la funzione merge che fonde le sottosequenze ordinate in un' unica sequenza di output.

1.3 Struttura dati

Per entrambi gli argoritmi è stata utilizzata la medesima struttura dati fornita da pyton, la lista. Questa struttura dati ha caratteristiche comuni, per quanto riguarda le operazioni per manipolare gli elementi, a quella del vettore di memoria.

2 Casi teorici:

2.1 Caso migliore

Le condizioni per cui si verificha il caso migliore nei due algoritmi sono differenti. Nel caso dell' Insertion_Sort il caso migliore si verifica quando la sequenza fornita è già ordinata. In tal caso la funzione di costo dell'algoritmo T(n) è uguale a $\theta(n)$. Per quanto riguarda l'algoritmo Quick_Sort il suo caso migliore si verifica quando la sequenza di input è casuale, perchè in questo caso i sotto array utilizzati risultano sempre bilanciati. Quando ciò accade la funzione T(n) è uguale a $\theta(nlgn)$. È già possibile notare come a livello teorico i due algoritmi nel loro caso migliore differiscano, ed è palese che nel caso migliore l'Insertion_Sort sia l'algoritmo più veloce.

2.2 Caso peggiore

I due algoritmi hanno condizioni agli antipodi per il verificarsi del caso peggiore. Per quanto riguarda l'algoritmo Insertion_Sort, il caso peggiore si verifica quando viene passata in ingresso una sequenza ordinata in modo decrescente; la funzione di costo T(n) in questo caso è uguale a $\theta(n^2)$. L'algoritmo di Quick_Sort, contrariamente a ció che ci aspetteremmo, verifica il caso peggiore se opera su una sequenza già precedentemente ordinata; la funzione di costo dell'algoritmo T(n) nel caso peggiore è comunque $\theta(n^2)$.

2.3 Caso medio

Il caso medio dell'Insertion_Sort coincide con il caso peggiore. Viceversa, il caso medio del Quick_Sort coincide con il suo caso migliore T(n). A livello teorico si è dunque portati a pensare che, in generale, il Quick_Sort sia meno costoso dell'Insertion_Sort.

3 Descrizione degli esperimenti

Dopo aver scritto le funzioni dei due algoritmi, è stata definita una funzione main che testasse entrambi gli algoritmi sulle stesse sequenze. Sono state creato tre differenti tipologie di sequenze con dimensioni che andavano da 1 a 1901. Su ogni dimensione è stato calcolato il tempo di esecuzionione di entrambi gli algoritmi per una trentina di volte. Per definire i grafici sono state utilizate le medie ottenute sperimentalmente.

4 Risultati dei test

Si riportano i risultati dei test eseguiti su i due algoritmi sopra descritti.

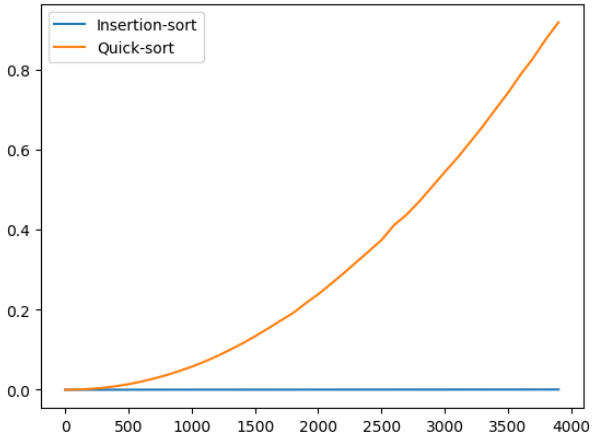

Figure 1: Confronto con sequenza ordinata

4.1 Sequenza ordinata

Dimensione	Insertion_Sort	Quick_Sort
1	0.0	0.0
201	0.0	0.0022
401	0.0	0.0088
601	0.0001	0.0203
801	0.0001	0.0366
1001	0.0001	0.0576
1201	0.0002	0.084
1401	0.0002	0.1157
1601	0.0002	0.1524
1801	0.0002	0.1914
2001	0.0003	0.2387
2201	0.0003	0.2909
2401	0.0003	0.3461
2601	0.0004	0.4115
2801	0.0004	0.4709
3001	0.0004	0.544
3201	0.0005	0.6182
3401	0.0005	0.7003
3601	0.0005	0.7874
3801	0.0005	0.876
3901	0.0005	0.9183

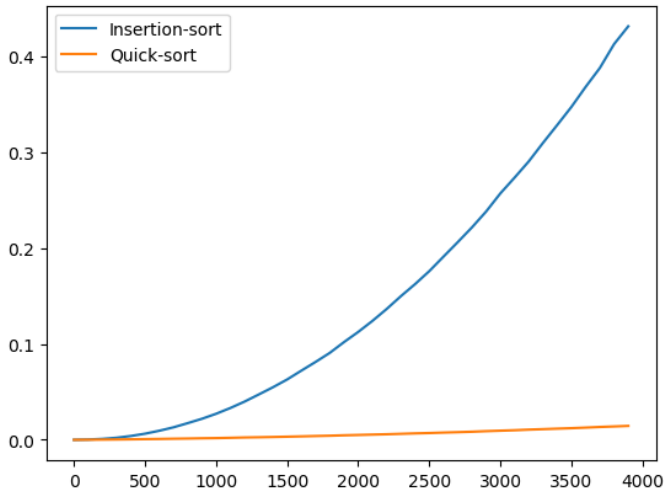

Figure 2: Confronto con sequenza random

4.2 Sequenza random

Dimensione	Insertion_Sort	Quick_Sort
1	0.0	0.0
201	0.001	0.0002
401	0.004	0.0005
601	0.0095	0.0009
801	0.0174	0.0013
1001	0.0272	0.0018
1201	0.0399	0.0024
1401	0.0549	0.0029
1601	0.0722	0.0036
1801	0.0908	0.0043
2001	0.1126	0.0051
2201	0.1366	0.0058
2401	0.1625	0.0067
2601	0.1912	0.0076
2801	0.2217	0.0084
3001	0.2572	0.0095
3201	0.2906	0.0106
3401	0.3286	0.0116
3601	0.3684	0.0127
3801	0.413	0.0139
3901	0.4316	0.0145

4.3 Sequenza decrescente

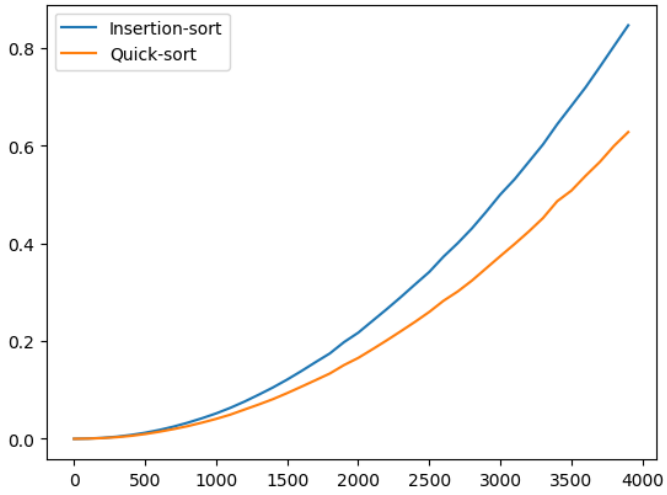

Figure 3: Confronto con sequenza decrescente

Dimensione	Insertion_Sort	Quick_Sort
1	0.0	0.0
201	0.0019	0.0015
401	0.0078	0.006
601	0.0181	0.0144
801	0.033	0.026
1001	0.052	0.0408
1201	0.0763	0.0599
1401	0.1052	0.0812
1601	0.1388	0.1068
1801	0.1749	0.1337
2001	0.2174	0.1657
2201	0.2655	0.2016
2401	0.3163	0.2396
2601	0.373	0.2827
2801	0.4309	0.3241
3001	0.5004	0.3743
3201	0.5671	0.4246
3401	0.6442	0.4866
3601	0.72	0.5387
3801	0.8045	0.6
3901	0.8469	0.6283

5 Conclusioni

A seguito dei risultati ottenuti, sono state confermate le aspettative teoriche su entrambi gli algoritmi. Infatti, nel caso di sequenze ordinate l'Insertion_Sort ha dominato il confronto con una funzione di costo lineare **Figure:**1. Appare evidente come giá con una dimensione di 500 elementi il Quick_Sort sia molto meno performante. Hanno, invece, esito opposto le prove effettuate su sequenze randomiche, dove, come previsto, l'Insertion_Sort ha le prestazioni del suo caso peggiore **Figure:**2. Questo è il risultato che rende il Quick_Sort un algoritmo più efficace. Questo perchè, come si nota da **Figure:**3, su sequenze randomiche la funzione di costo è logaritmica, come nel suo caso migliore. Essendo sempre possibile randomizzare le sequenze in ingresso, il caso peggiore del Quick_Sort è pressochè inesistente. L'ultimo caso è stato computazionalmente costoso per entramnbi gli algoritmi **Figure:**3, però su grandi dimensioni anche in questo caso ha prevalso il Quick_Sort .