Relazione su Merge Sort e Quick sort

$Autore : Georgiev\ David\ mat : 1043306$

Introduzione generale

In questa relazione vengono esaminate e confrontate le prestazioni degli algoritmi di ordinamento quick sort e merge sort implementati in modo da poter ordinare array che contengono elementi con un tipo di dati generico. In particolare si è fatto uso di un file di tipo **CSV** contenente dei record con una struttura ben definita:

- **id** : un numero intero univoco per ogni record
- **field 1** : una stringa
- **field 2** : un numero intero
- **field 3** : un numero in virgola mobile

Questi campi sono separati da una virgola e ogni record finisce con un carattere a capo (’\n’).

I test di confronto sono stati eseguiti su un numero diverso di record per esaminare il tempo di risposta con l’aumentare della dimensione dell’input. Scendendo nel dettaglio, sono stati esaminati gli ordinamenti di array di record con dimensione 1.000.000, 2.500.000, 5.000.000, 7.500.000, 10.000.000, 12.500.000, 15.000.000, 17.500.000 e infine 20.000.000 elementi.

Nota importante: I test sono stati eseguiti su un computer solo quindi i tempi di risposta non sono da considerare come un indicatore assoluto per misurare la differenza tra i due algoritmi. Inoltre, questi tempi sono anche influenzati dallo stato interno della macchina come per esempio il numero di processi attivi nel sistema; ciò implica che per avere una migliore stima converrebbe fare un analisi qualitativa degli algoritmi basandosi sul numero di operazioni eseguite non sui tempi di risposta.

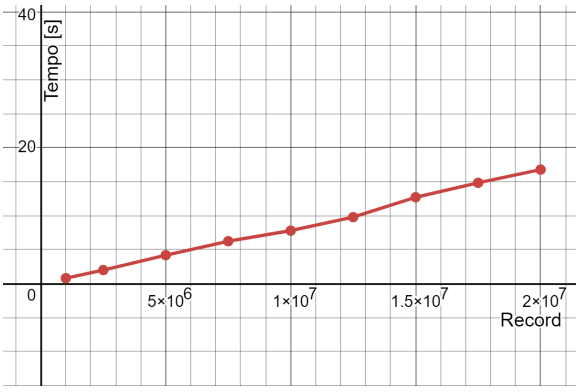
Ordinamento di Stringhe

Esito: Positivo

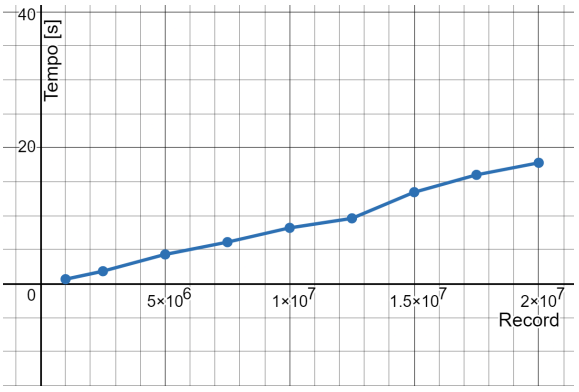
L'ordinamento di stringhe è stato completato in meno di 20 secondi da entrambi gli algoritmi. Con l'aumentare del numero di record si è notato un lieve peggioramento del quick sort rispetto al merge sort sebbene non sia un calo di prestazioni particolarmente rilevante. (vedi figure 1a e 1b).

Nota aggiuntiva: L'ottimizzazione applicata al quick sort impedisce ad esso di degenerare nel caso peggiore ovvero con una complessità temporale di O(n^2) dove n indica la dimensione dell'array.

Questa ottimizzazione è in grado di ignorare i sottoinsiemi di elementi tutti uguali che l'algoritmo quick sort esaminerebbe uno ad uno se non fosse controllato. In sostanza il costo maggiore è dato da un confronto semplice in più per ogni chiamata di partizione quindi, anche se c'è un leggero degrado per array di elementi tutti distinti, questa ottimizzazione aumenta le prestazioni col crescere della dimensione dell'array considerando che la probabilità di incontrare elementi ripetuti aumenta con l'aumentare della dimensione stessa dell'array. (a meno che non si tratti di un array contenente valori unici tra di loro). D'altra parte è da sottolineare anche che con l'aumentare del dominio dei valori nell'array anche la probabilità che gli elementi si ripetano diminuisce.



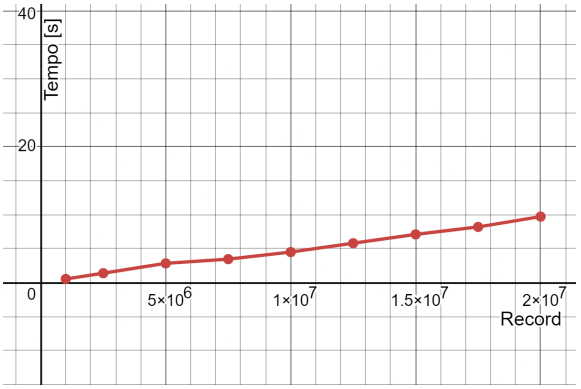
1a. Tempi di risposta del Merge sort (Stringhe)



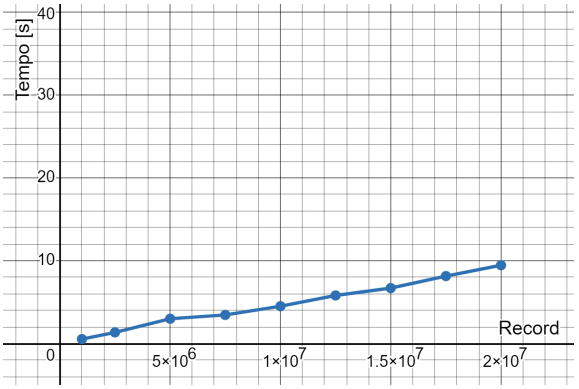
1b. Tempi di risposta del Quick sort (Stringhe)

Ordinamento di Interi

L’ordinamento di interi è stato completato in meno di 10 secondi da entrambi gli algoritmi. I tempi di risposta sono risultati quasi uguali per tutti gli array testati con una leggera differenza in favore del quick sort per l’ordinamento di pochi record e una differenza in favore del merge sort per l’ordinamento di molti record. (Vedi figure 2a e 2b)



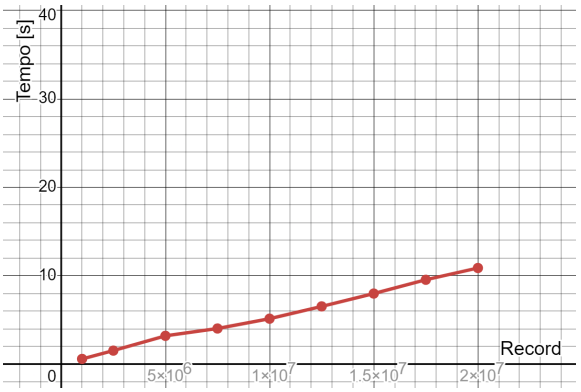
2a. Tempi di risposta del Quick sort (Interi)



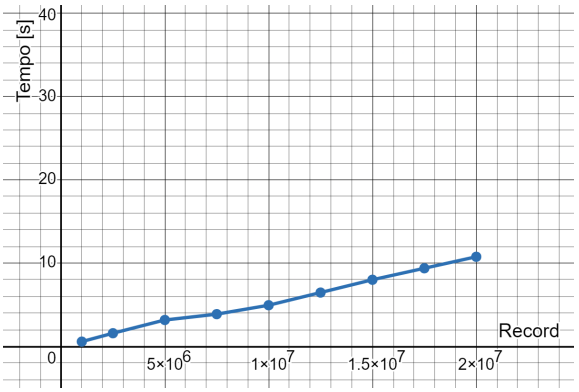
2b. Tempi di risposta del Quick sort (Interi)

Ordinamento di Numeri in virgola mobile

L’ordinamento di float è stato completato in poco più di 10 secondi da entrambi gli algoritmi. I tempi di risposta sono risultati pressoché uguali per tutti gli array testati.



3a. Tempi di risposta del Merge sort (Float)



3b. Tempi di risposta del Quick sort (Float)

Osservazioni aggiuntive

Siccome il tipo di dato stringa è un array di per sé, l'ordinamento di questo tipo di dato è stato il più lento in confronto all'ordinamento dei tipi di dato primitivi. Da ciò si può prematuramente dedurre che i tempi di risposta degli algoritmi dipendono molto anche dal tipo di dato che devono ordinare; più il dato è complesso da gestire e confrontare, più sarà impegnativo l'ordinamento da parte degli algoritmi. (Notare come strutture dato che occupano tanti byte in memoria influenzano anche i tempi siccome possono occupare più blocchi di memoria in memoria secondaria quindi "sprecare" più tempo per il trasferimento in memoria primaria).

Per gli ordinamenti dei tipi di dato primitivo, ovvero Interi e float in questo caso, l'algoritmo ha dato tempi di risposta molto simili per via della semplicità con cui si possono confrontare questi tipi. Tuttavia è da notare un lieve aumento nei tempi di risposta dell'ordinamento dei numeri in virgola mobile visto che le operazioni con tali numeri richiedono più tempo di CPU rispetto a dei semplici interi.

Per concludere si può notare una somiglianza con i vari grafici di tutti gli ordinamenti; l'andamento della funzione sembra essere quasi lineare il che è un possibile indice di conferma che la complessità temporale di entrambi gli algoritmi è O(n * log(n)) a meno di costanti moltiplicative.