Università degli Studi di Torino
Corso di Laurea Magistrale in Informatica
Algoritmi e Strutture dati

Relazione esercitazione 1

Alessandro Clocchiatti
Matricola 909105

Consegna
Implementare una libreria che offre i seguenti algoritmi di ordinamento:

- Insertion-sort;
- Quick-sort

Il codice che implementa ciascun algoritmo deve essere generico. Inoltre, ogni algoritmo deve permettere di specificare (cioè deve accettare in input) il criterio secondo cui ordinare i dati.

Unit Testing
Implementare gli unit-test degli algoritmi secondo le indicazioni suggerite nel documento Unit Testing.

Uso della libreria di ordinamento implementata
Il file *records.csv* contiene 20 milioni di record da ordinare. Ogni record è descritto su una riga e contiene i seguenti campi:

- id: (tipo intero) identificatore univoco del record;
- field1: (tipo stringa) contiene parole estratte dalla divina commedia, potete assumere che i valori non contengano spazi o virgole;
- field2: (tipo intero);
- field3: (tipo floating point).

Il formato è un CSV standard: i campi sono separati da virgole; i record sono separati da \n.
Usando ciascuno degli algoritmi implementati, si ordinino i *record* (non è sufficiente ordinare i singoli campi) contenuti nel file *records.csv* in ordine non decrescente secondo i valori contenuti nei tre campi "field" (cioè, per ogni algoritmo, è necessario ripetere l'ordinamento tre volte, una volta per ciascun campo).
Si misurino i tempi di risposta e si produca una breve relazione in cui si riportano i risultati ottenuti insieme a un loro commento. Nel caso l'ordinamento si protragga per più di 10 minuti potete interrompere l'esecuzione e riportare un fallimento dell'operazione. I risultati sono quelli che vi sareste aspettati? Se sì, perché? Se no, fate delle ipotesi circa il motivo per cui gli algoritmi non funzionano come vi aspettate, verificatele e riportate quanto scoperto nella relazione.

---

1. Risultati

I tempi di esecuzione degli algoritmi ottenuti sono i seguenti:

	String	*Integer*	*Float*
Insertion-sort	FAIL	FAIL	FAIL
Quick-sort	FAIL	$\sim 20\ sec$	$\sim 26\ sec$

FAIL indica che il tempo di esecuzione è superiore a 10 minuti

I risultati ottenuti rispettano le ipotesi fatte e le stime di complessità degli algoritmi utilizzati. Infatti, siccome il Quick-sort ha una complessità di $O(n \log n)$ mentre l'Insertion-sort ha una complessità di $O(n^2)$, l'esecuzione del primo risulta essere molto più veloce rispetto al secondo.

Nel caso del Quick-sort, la similarità tra le tempistiche di ordinamneto di dati interi o floating point dipende dal fatto che le funzioni di confronto hanno complessità $O(1)$. La leggera differenza può dipendere dalla maggiore complessità della funzione di confronto dei dati floating point.

Il fallimento dell'ordinamento di stringhe tramite Quick-sort può dipendere in parte della funzione di confronto, la quale ha complessità $O(m)$ dove m è la lunghezza della stringa più breve.
Inoltre, l'ordinamento di stringhe richiede un tempo maggiore anche a causa della presenza di stringhe ripetute in record diversi. Ogni volta che una stringa viene presa come perno, le stringhe maggiori o uguali vengono spostate nella partizione destra. La successiva chiamata ricorsiva su questa partizione genererà delle partizioni non bilanciate (caso peggiore), in quanto il nuovo perno sarà l'elemento minore tra quelli presenti nella partizione. In questo modo il numero di chiamate ricorsive aumenta, rallentando la durata dell'ordinamento.

Una possibile soluzione, non implementata, è quella di introdurre un ulteriore gruppo in cui partizionare l'array:

- elementi con valore minore del perno;
- elementi con valore uguale al perno;
- elementi con valore maggiore al perno.

In questo modo le chiamate ricorsive vengono fatte sulle partizioni dell'array non contenenti elementi uguali al perno.