Esercitazione 9
Algoritmi di Ordinamento

Corso di Fondamenti di Informatica II

BIAR2 (Ing. Informatica e Automatica) e BSIR2 (Ing. dei Sistemi)

A.A. 2010/2011

14 gennaio 2011

Sommario

Scopo di questa esercitazione è il confronto dei diversi algoritmi di ordinamento visti a lezione.

Algoritmi di Ordinamento

Tra i diversi algoritmi di ordinamento visti a lezione (Insertion Sort, Selection Sort, Heap Sort, Merge Sort e Quick Sort) si prenda in considerazione in particolare il *Merge Sort* e tre sue possibili ottimizzazioni.

Ottimizzazione 1 (Allocazione)**.** *Allocare un array è una operazione costosa ed il Merge Sort richede la creazione di un nuovo array ad ogni passo di merge. L'ottimizzazione proposta richiede di allorare un solo array di lavoro grande quanto quello da ordinare ed utilizzare sempre quello, stando attenti a gestire correttamente i limiti dei dati "significativi".*

Ottimizzazione 2 (Ricorsione)**.** *L'ottimizzazione proposta consiste nell'implementare una versiona iterativa del Merge Sort.*

Ottimizzazione 3 (Merge)**.** *Durante il passo di merge, l'algoritmo copia il sotto-array ottenuto dall'operazione di merge nello spazio dedicato al sotto-array originale. In realtà, si possono risparmiare delle operazioni, come mostrato in Figura, quindi l'ottimizzazione proposta è copiare (e spostare se necessario) soltanto la parte di sotto-array che ha subito una modifica.*

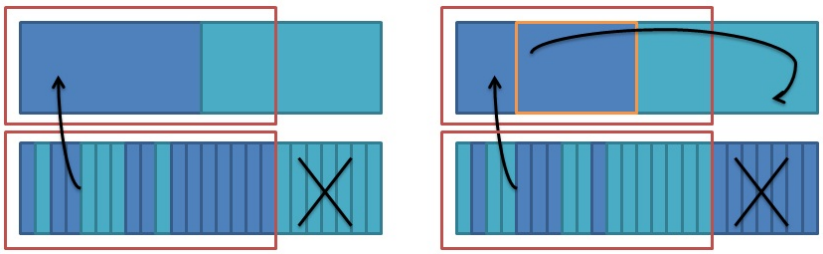

Figura 1: Operazione di merge.

1 Merge Sort

Si vuole implementare l'algoritmo Merge Sort con le tre ottimizzazioni viste a lezione e descritte nell'introduzione. Possibilmente, si implementino le tre ottimizzazioni in modo che siano "selezionabili", ovvero che sia possibile con un sistema a scelta dello studente (un parametro ad esempio) scegliere quali attivare e queli no.

Materiale di Supporto. Nel materiale di supporto è già presente la classe `Ordinamento` dove sono implementati tutti gli algoritmi di ordinamento, tra cui il Merge Sort senza ottimizzazioni.

Programma Java. Riscrivere il metodo statico della classe `Ordinamento` che implementa *Merge Sort* e se necessario cambiare il set di parametri.

```
public static void mergeSort(Comparable A[])
```

2 Sperimentazione

Si vuole procedere alla sperimentazione considerando, per ogni operazione di ordinamento, il tempo di esecuzione a fronte di array:

- con valori randomici in $[0, M]$ con M deciso opppurtunamente (per esempio se `a` è l'array si può avere $M = |a| - 1$);
- con valori ottenuti a partire da un array con valori randomici in $[0, M]$ ordinato per una percentuale di ordinamento decisa opportunamente.

Si consideri che un array ordinato al 70% è un array randomico ordinato al quale sono stati applicati un numero di scambi di elementi pari al 30% della sua dimensione totale.

Ad esempio un array ordinato al 100% non ha subito scambi, mentre un array di 10 elementi ordinato al 20% ha subito 8 scambi di posizione tra i suoi elelemnti.

Esempio. Un seplice main di sperimentazione è il seguente.

```
Integer[] pluto = ArrayUtil.getRandomArray(n, n, o);
long prima = System.currentTimeMillis();
Ordinamento.mergeSort(pluto);
long dopo = System.currentTimeMillis();
int min, sec, msec;
min  = (int) Math.floor((dopo-prima)/(1000*60));
sec  = (int) Math.floor((dopo-prima)/1000) % 60;
msec = (int) (dopo-prima)%1000;
salvaRisultati("Merge Sort", n, min, sec, msec);
```

Materiale di Supporto. Come si può vedere nell'esempio, oltre alla classe `Ordinamento`, viene fornita una classe `ArrayUtil` contenente i metodi

```
public static Integer[] getRandomArray(int size, int max, int
    ordine)
```

Restitusce un array di `Integer` di cardinalità `size`, con valori tra 0 e `max` e con una percentuale di ordinamento pari a `ordine`.

```
public static Integer[] getRandomArray(int size, int max)
```

Restitusce un array di `ITnteger` di cardinalità `size` e con valori tra 0 e `max`. La percentuale di ordinamento può essere qualsiasi.

Inoltre, nella classe `Esperimento` è già presente il main di sperimentazione dell'esempio e dei metodi di salvataggio su file dei risultati come:

```
private static void salvaRisultati(String alg, int n, int min,
    int sec, int msec)
```

Programma Java. Scrivere un nuovo `main` nella classe `Esperimento` che permetta di ottenere un file di risultati da cui sia possibile effettuare una valutazione significativa degli algoritmi di ordinamento. Il file, cha sarà stato scritto con una opportuna formattazione (modificando se necessario il metodo `salvaRisultati`), sia fatto infine processare da un programma di visualizzazione dati (Excel, Calc, gnuplot) e si proceda con l'analisi dei risultati.

Suggerimento: dimensioni dell'input. In questo contesto, una dimensione sicificativa dell'array da ordinare può essere compresa tra 10^6 e 10^7.

Suggerimento: problemi di misura sperimentale del tempo di esecuzione. Il metodo `System.currentTimeMillis`, proposto per la misura del tempo di esecuzione, restituisce un `long` contenente il numero di millisecondi trascorsi dal 1 gennaio 1970 e l'istante in cui viene invocato. Questo metodo non permette di ottenere una grande risoluzione di misura, ovvero quando invocato due volte molto vicine nel tempo, può sbagliare di molto, e ad esempio misurare una differenza di 1 millisecondo a fronte di 10 o viceversa. Si consiglia dunque di misurare il tempo di esecuzione di un algoritmo come la media dei tempi ottenuti su sa serie di esecuzioni (ad esempio 20).