Valutazione performance InsertionSort e MergeSort

Relazione esercizio 1

Laboratorio di Algoritmi e Strutture Dati

Gemma Vaggelli
6348717

Ingegneria Informatica
Università degli studi di Firenze

1 Introduzione

Il problema dell'ordinamento è un problema di tipo algoritmico in cui, dato in input una sequenza di n numeri $< a_1; a_2; ... ; a_n >$, genera in output una permutazione $< a_1^{'}; a_2^{'}; ... ; a_n^{'} >$ della sequenza di input tale che $< a_1^{'} \leq a_2^{'} \leq ... \leq a_n^{'} >$.
Tra i molti algoritmi di ordinamento in questo esercizio verranno trattate e valutate le performance degli algoritmi InsertionSort e MergeSort.
Vogliamo valutare attraverso dei test quale dei 2 algoritmi è più efficiente; ciò verrà effettuato valutando i 2 algoritmi in base a due fattori: la velocità di esecuzione all'aumentare della lunghezza degli array e la medesima in base alla conformazione dell'array in input.

2 Caratteristiche

I due algoritmi presentano delle differenze sostanziali nel trattamento dell'array in input.

2.1 Caratteristiche InsertionSort

L'insertionSort è un algoritmo di ordinamento iterativo che si basa sull'inserimento di un elemento all'interno di un array.Il suo funzionamento è paragonabile all'operazione che si svolge quando si ordinano le carte in mano, partendo da una mano vuota e tutte le carte in tavola si pesca una carta alla volta e si inserisce nella posizione 'giusta', ovvero in maniera ordinata rispetto alle carte che si hanno in mano in quel momento.

2.2 Caratteristiche MergeSort

L'algoritmo MergeSort d'altra parte è un algoritmo ricorsivo Divide et Impera e come tale, si compone di 3 fasi:

- **Divide** che consiste nel dividere a metà l'array;
- **Impera** che ordina, chiamando ricorsivamente la funzione MergeSort, i 2 sotto-array;
- **Combina** che, all'interno della funzione Merge, prende i 2 sotto-array ordinati e forma un singolo array ordinato.

3 Prestazioni

3.1 Prestazioni InsertionSort

Dall'analisi teorica dell'algoritmo InsertionSort possiamo vedere che le prestazioni dell'algoritmo cambiano al variare della disposizione degli elementi all'interno dell'array, perciò bisogna testare le prestazioni dell'algoritmo nel caso migliore, nel caso medio e nel caso peggiore.
Il caso migliore è un array di partenza già ordinato, in questo caso il tempo di esecuzione dell'algoritmo T(n) è una funzione lineare di n.
Il caso peggiore è un array di partenza ordinato al contrario, in questo caso T(n) è una funzione quadratica di n.
Nel caso medio l'algoritmo nel ciclo while fa circa la metà dei controlli rispetto a quelli che fa nel caso peggiore però T(n) rimane una funzione quadratica di n.

3.2 Prestazioni MergeSort

Dall'analisi teorica dell'algoritmo MergeSort si nota che le prestazioni dell'algoritmo sono pressoché le stesse qualsiasi sia la disposizione degli elementi all'interno dell'array, quindi non vi è un caso migliore o peggiore.
Come per gli altri algoritmi ricorsivi Divide et Impera si valuta il costo del MergeSort attraverso l'equazione di ricorrenza.

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ 2T(\frac{n}{2}) + \Theta(n) & \text{se } n > 1. \end{cases}$$

L'equazione di T(n) in forma chiusa con notazione asintotica per il MergeSort è T(n) $= \Theta(n)$

4 Documentazione

Il programma è formato dalle funzioni che eseguono gli algoritmi **InsertionSort** e **MergeSort**, dalla funzione **Client** che si occupa di svolgere i test e di calcolare il tempo di esecuzione di ognuno di essi e dalla funzione **BGenerator** che si occupa della creazione dei vari array su cui applicare i due algoritmi di ordinamento.
La funzione BGenerator prende in input la lunghezza dell'array che si vuole generare e una stringa che determina la modalità di creazione del medesimo.La funzione può generare array casuali, ordinati o ordinati al contrario.

All'interno della funzione Client, oltre a ciò che ho specificato sopra, viene creato un file csv per visualizzare i dati elaborati dalla funzione quali la dimensione dell'array da ordinare e il tempo impiegato (in millisecondi) da ciascun algoritmo di ordinamento.

Hardware I test sono stati eseguiti su un PC Desktop con sistema operativo Windows 10 Home a 64 bit, un processore Intel Core i5-7200U con 8Gb di RAM e Visual Studio Code come IDE.

5 Prestazioni risultati sperimentali

5.1 Prestazioni risultati sperimentali caso Random

Dim	Merge	Ins
0	0	0
200	4	4
400	4	24
600	4	44
800	8	68
1000	8	104
1200	16	224
1400	12	284
1600	20	392
1800	20	480
2000	28	540
2200	20	600
2400	24	640
2600	28	992
2800	28	936
3000	32	1088
3200	48	1172
3400	32	1440
3600	48	1644
3800	36	1972
4000	48	2136
4200	56	2596
4400	48	2512
4600	68	2872
4800	48	3432
5000	56	3112

5.2 Prestazioni risultati sperimentali best case

Dim	Merge	Ins
0	0	0
200	4	0
400	0	0
600	4	0
800	8	0
1000	8	0
1200	12	0
1400	12	0
1600	16	4
1800	16	0
2000	16	0
2200	20	0
2400	20	0
2600	24	0
2800	28	0
3000	40	4
3200	42	0
3400	34	4
3600	54	0
3800	42	4
4000	36	4
4200	34	4
4400	44	0
4600	44	4
4800	56	4
5000	69	4

5.3 Prestazioni risultati sperimentali worst case

Dim	Merge	Ins
0	0	0
200	0	8
400	4	32
600	4	76
800	4	140
1000	8	216
1200	8	476
1400	20	492
1600	16	764
1800	20	748
2000	16	920
2200	20	1296
2400	28	1296
2600	20	1804
2800	24	2096
3000	28	2244
3200	36	2740
3400	28	2684
3600	32	3408
3800	32	3520
4000	52	4244
4200	40	4536
4400	36	5032
4600	40	5148
4800	40	6052
5000	40	6168

5.4 Grafici

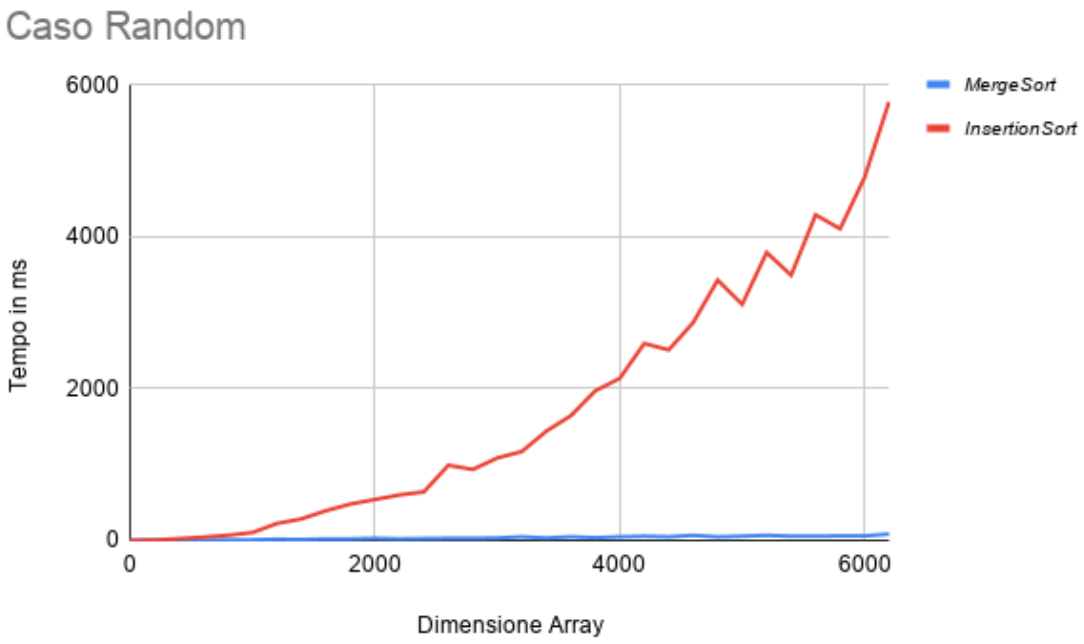

Figura 1: Ordinamento di array causale

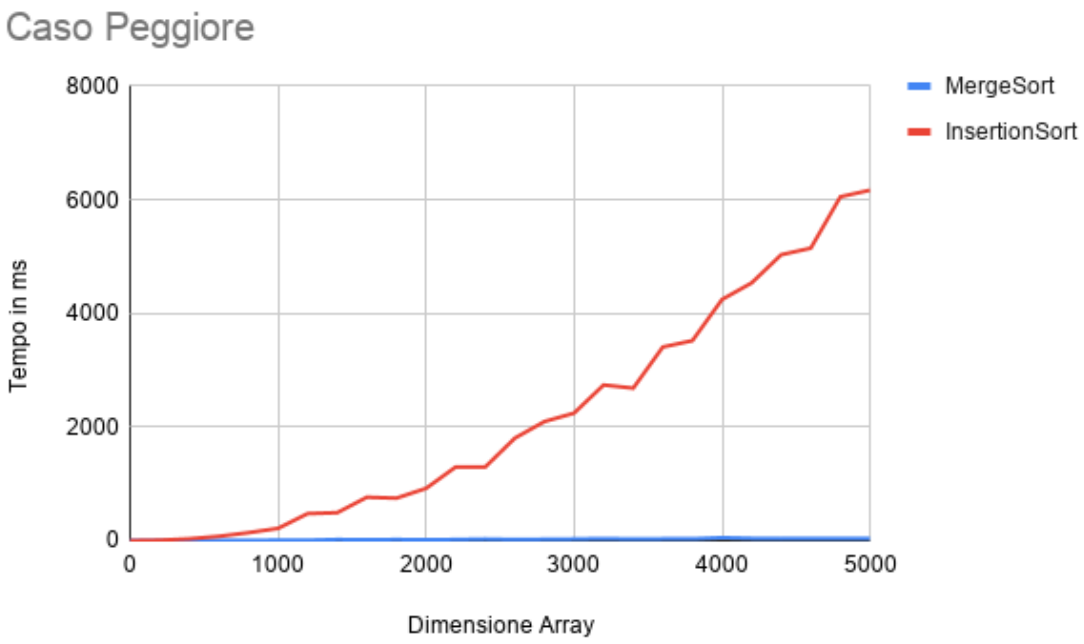

Figura 2: Ordinamento di array ordinato al contrario

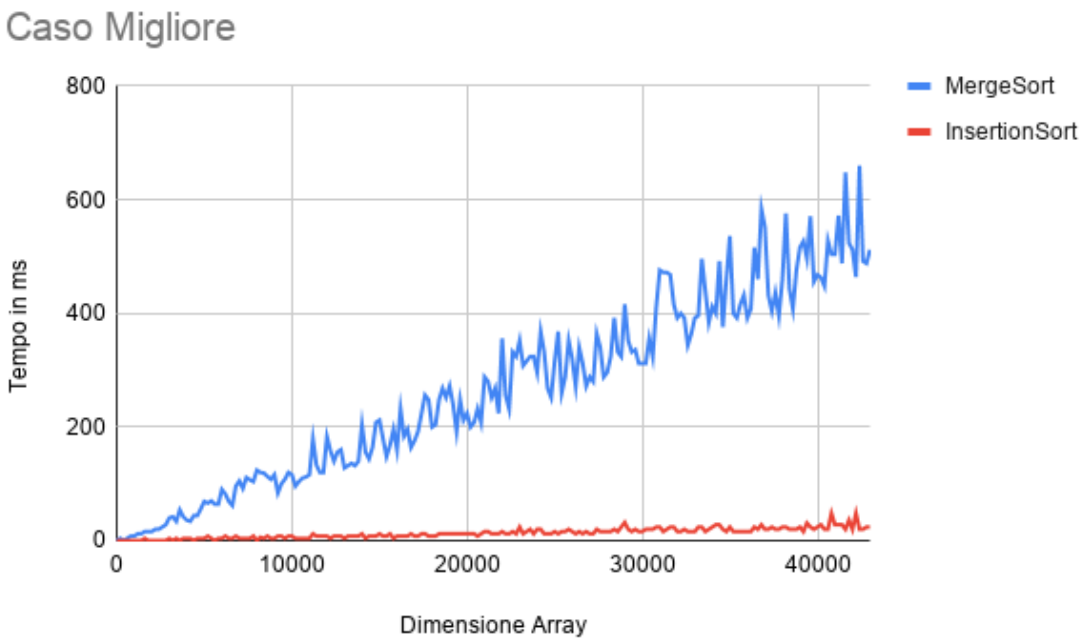

Figura 3: Ordinamento di array ordinato

6 Analisi risultati sperimentali

6.1 Analisi risultati sperimentali InsertionSort

Per qualsiasi dimensione degli array i risultati sono simili ed in linea con le analisi teoriche e le prestazioni attese.
In ogni test eseguito il tempo di esecuzione del Best-case risulta molto più veloce rispetto a qualsiasi altro tempo di esecuzione degli altri array con la stessa dimensione, infatti prendendo come esempio un array di 19800 elementi (ultimo dato riportato in tabella) si ha che il tempo di esecuzione del Best-case è circa 12ms, mentre il tempo di esecuzione di un array randomico di 6200 elementi supera i 5 secondi a seguito dei quali il programma non riesce ad elaborare l'array successivo nel tempo massimo concesso.
In ogni test eseguito il tempo di esecuzione del Worst-case risulta circa il doppio più lento rispetto al tempo di esecuzione degli array random della medesima dimensione.
Il best-case invece risulta sempre molto più veloce anche del caso medio, infatti analizzando il grafico si vede che ha un andamento lineare rispetto al caso medio e al

Worst-case.
Per array di 10mila elementi il tempo di esecuzione cresce in modo drastico e c'è molta differenza tra i 3 casi. Il tempo di esecuzione del Best-case rimane molto basso invece del caso medio e del Worst-case aumenta drasticamente. Questo andamento continua anche per array crescenti.

6.2 Analisi risultati sperimentali MergeSort

Anche in questo caso per qualsiasi dimensione degli array i risultati sono simili ed in linea con le analisi teoriche e le prestazioni attese.
In ogni test con array della stessa dimensione il tempo di esecuzione degli array è molto simile tra loro. Nel MergeSort non esistono dei casi migliori e peggiori ma ogni array della stessa dimensione ha circa lo stesso tempo di esecuzione degli altri infatti anche quegli array che erano il Best-case e il Worst-case nell'InsertionSort, nel MergeSort hanno circa lo stesso tempo di esecuzione di un altro array generato random.
Come si può vedere dal grafico, per array fino a 10mila elementi il tempo di esecuzione è compreso tra 0 e 1 secondo, si inizia a vedere un aumento del tempo di esecuzione per array di 100mila elementi che è di poco meno di 2 secondi.

7 Conclusione

In generale il MergeSort è un algoritmo molto più efficiente dell'InsertionSort poiché nel caso medio il MergeSort è estremamente più veloce dell'InsertionSort. Come abbiamo dimostrato praticamente nell'esercizio si ha infatti che per array molto grandi la differenza del tempo di esecuzione dei 2 algoritmi è notevole, se si deve invece ordinare array di circa 100 elementi o inferiori la differenza è appena percettibile.