Esercizio 1

Alessandro D'Amico

3 Luglio 2019

Indice

1 Introduzione al problema

Nel seguente esperimento viene preso in considerazione il problema dell'ordinamento, per la cui soluzione utilizziamo due differenti algoritmi: Insertion Sort e Merge Sort. Lo scopo e' ottenere un array (una lista di numeri) i cui elementi che lo compongono (inizialmente disposti in modo casuale) siano disposti in ordine crescente.

2 Caratteristiche teoriche degli algoritmi utilizzati

3 Prestazioni attese

Un array ordinato in modo crescente rappresenta il miglior caso per Insertion Sort, mentre un array ordinato in modo decrescente rappresenta il suo caso peggiore.

Algoritmo	Tempo di esecuzione caso peggiore	Tempo di esecuzione atteso / caso medio
Insertion Sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$

4 Esperimenti

Sono stati effettuati i test con piu set di numeri:
- numeri **random**
- numeri ordinati in modo **crescente**
- numeri ordinati in modo **decrescente**

I set sono ulteriormente suddivisi in due tipologie:
- **Small**: contengono array piccoli (fino a 200 elementi)
- **Big**: contengono array di molto grandi (fino a 100000 elementi)

Essendo difficile simulare caso migliore e peggiore di Merge Sort, ci limitiamo ad analizzare i casi peggiore, migliore e medio di Insertion Sort su entrambi gli algoritmi (il caso medio e' simulato con numeri random)

5 Documentazione del codice

5.1 DataSetGenerator.py

Questo file ha l'unica funzione di generare i Dataset. **random_vect(B)**: Restituisce un array di B numeri generati randomicamente (Ricevendo B in ingresso).
incr_vect(B): Restituisce un array di B+1 numeri interi in ordine crescente e con passo 1 (da 0 a B)
decr_vect(B): Restituisce un array di B+1 numeri interi in ordine decrescente e con passo 1 (da B a 0)
multiple_random_vect(MultipleNumberVect, step1, multi): Riceve in

ingresso un array vuoto e restituisce un array di array contenenti numeri random. Gli array di numeri hanno dimensione pari al prodotto tra i e step1 dove i e' compreso tra 0 e multi.
multiple_incr_vect(MultipleNumberVect, step1, multi): Riceve in ingresso un array vuoto e restituisce un array di array contenenti numeri ordinati in modo crescente. Gli array di numeri hanno dimensione pari al prodotto tra i e step1 dove i e' compreso tra 0 e multi.
multiple_decr_vect(MultipleNumberVect, step1, multi): Riceve in ingresso un array vuoto e restituisce un array di array contenenti numeri ordinati in modo decrescente. Gli array di numeri hanno dimensione pari al prodotto tra i e step1 dove i e' compreso tra 0 e multi.

5.2 InsertionVsMergeBenchmark.py

Questo file contiene il main ed i test.
Funzioni:
Insertion_sort(A): e' l'algoritmo Insertion Sort, in ingresso riceve l'array A, contenente i numeri da ordinare. Come valore di ritorno ha il tempo di esecuzione dell'algoritmo stesso.
MergeSort(A, p, r): e' l'algoritmo Merge Sort, in ingresso riceve l'array A contenente i numeri da ordinare, gli indici degli estremi dell'array da ordinare (p ed r). La funzione si chiama ricorsivamente fino ad ottenere problemi di dimensione unitaria ed in
Merge(A, p, q, r): e' il cuore del Merge Sort (la parte "Impera").
random_vect(B): Restituisce un array di B numeri generati randomicamente (Ricevendo B in ingresso).
incr_vect(B): Restituisce un array di B+1 numeri interi in ordine crescente e con passo 1 (da 0 a B)
decr_vect(B): Restituisce un array di B+1 numeri interi in ordine decrescente e con passo 1 (da B a 0)
MergeSortMask(A, p, r): e' una chiamata a MergeSort, la sfrutto per avere un timer esterno. Il valore restituito e' il tempo d'esecuzione di Merge Sort registrato dal timer.
testComparison(rep,Setfile): effettua i test sul dataset passato in ingresso Setfile ed esegue un numero di ripetizioni rep (per rendere i test indipendenti da picchi di carico del processore). Restituisce un grafico (con 2 linee) dei tempi risultanti.
mergeTestComparison(rep): calcola il tempo di esecuzione di Merge Sort per ogni array di ognuno dei 3 dataset: il primo e' un dataset (di array di varie dimensioni) di numeri random, il secondo e' un dataset (di array di varie dimensioni) di numeri ordinati in modo crescente ed il terzo e' un dataset (di array di varie dimensioni) di numeri ordinati in modo decrescente. Restituisce un grafico (con 3 linee) dei tempi risultanti.
insertionTestComparison(rep): Lo stesso di mergeTestComparison, ma con Insertion Sort.

6 Risultati

6.1 Dataset di numeri in ordine crescente

Dalla Figura 1 traspare che per ordinare meno di 80 numeri Insertion Sort risulta piu' veloce (cio' e' dovuto principalmente al tempo impiegato al caricamento delle funzioni sulla stack).

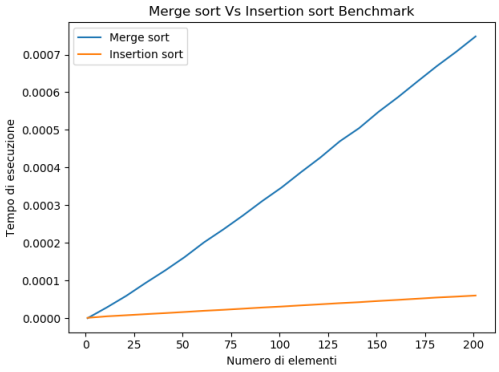

Figura 1: Merge Sort vs Insertion Sort per pochi input (ordinati crescentemente)

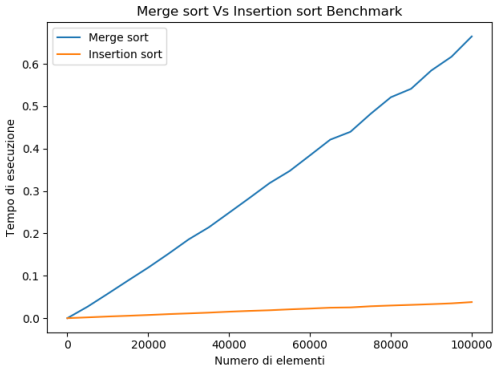

Figura 2: Merge Sort vs Insertion Sort per molti input (ordinati crescentemente)

6.2 Dataset di numeri in ordine decrescente

Grazie alla figura 4 e' facilmente deducibile che per grandi set di numeri da ordinare Merge Sort e' sensibilmente piu' rapido rispetto a Insertion Sort

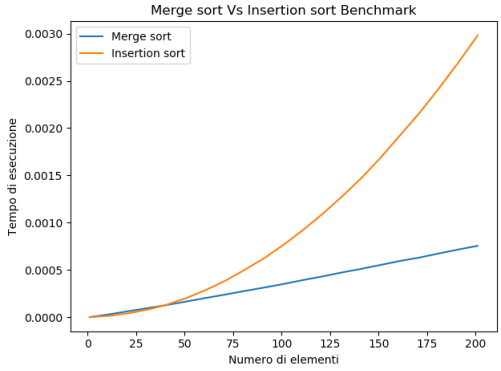

Figura 3: Merge Sort vs Insertion Sort per pochi input (ordinati decrescentemente)

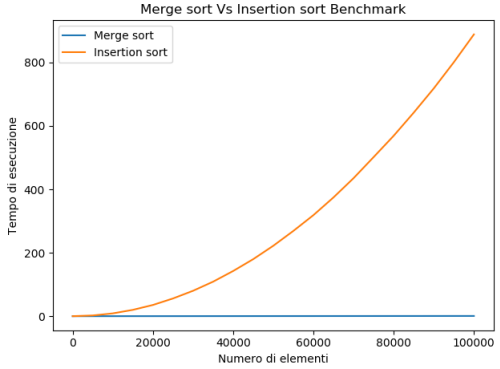

Figura 4: Merge Sort vs Insertion Sort per molti input (ordinati decrescentemente)

6.3 Dataset di numeri in ordine random

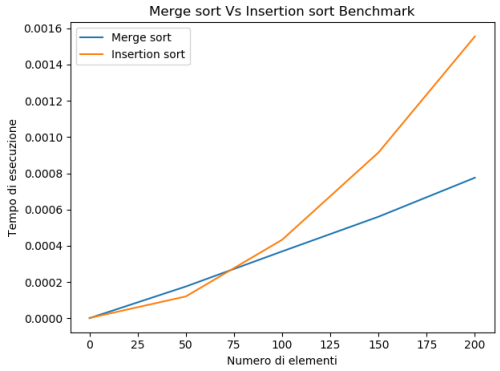

Figura 5: Merge Sort vs Insertion Sort per pochi input (random)

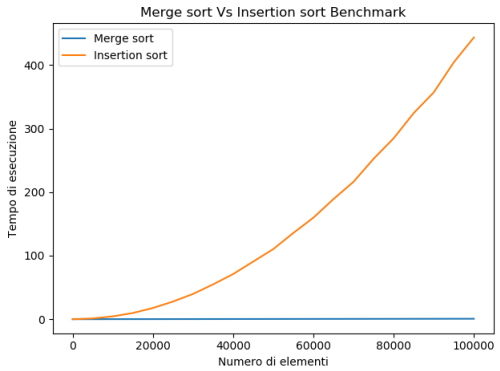

Figura 6: Merge Sort vs Insertion Sort per molti input (random)

6.4 Merge Sort per dataset delle varie tipologie

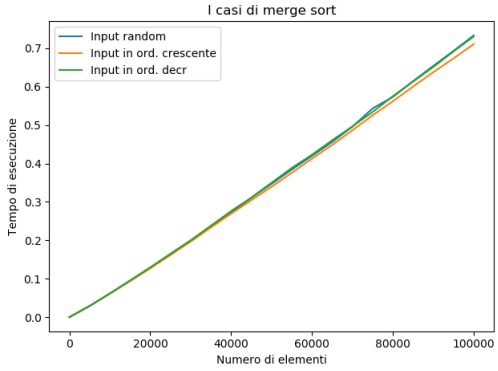

Figura 7: Merge Sort per varie tipologie di dataset

6.5 Insertion Sort per dataset delle varie tipologie

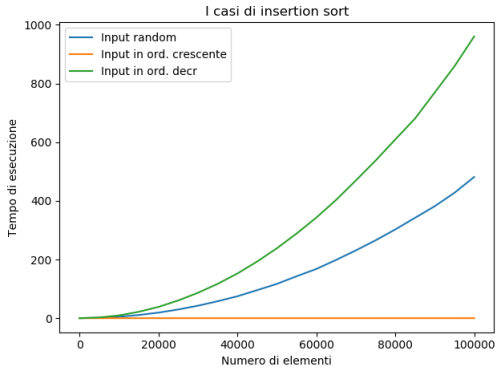

Figura 8: Insertion Sort per varie tipologie di dataset

7 Conclusioni

E' stato verificato il comportamento dei due algoritmi al variare della tipologia di dataset utilizzati. Possiamo concludere che per una quantita' molto ridotta di numeri da ordinare (pari a circa 80) e' conveniente utilizzare Insertion Sort (essendo Merge Sort ricorsivo, l'operazione di caricamento della funzione sulla stack richiede un tempo, che per pochi input puo' incidere sensibilmente sul tempo finale), ma escluso tale caso ed il caso in cui tutti i numeri risultino ordinati, Merge Sort risulta notevolmente piu' veloce.