Studio e analisi di algoritmi di ordinamento: Merge-Sort e Insertion-Sort a confronto

Leonardo Gori

January 25, 2021

Abstract

L'ordinamento degli elementi all'interno di un insieme è uno dei problemi che più spesso viene affrontato in campi come computer science e data analysis. Per questo motivo esistono più algoritmi di ordinamento che sono di fondamentale utilizzo per garantire ottimalità in termini di ottimizzazione di calcoli computazionali e di risparmio di risorse. In questa relazione verranno presentati gli algoritmi di ordinamento Merge-sort e Insertion-sort per l'ordinamento di liste, in particolare verranno messi in confronto i tempi di ordinamento di liste di dimensione e input diversi.

1 Introduzione

Un algoritmo di ordinamento posiziona gli elementi di un insieme secondo una relazione d'ordine, in particolare devono essere garantite due proprietà fondamentali:

1. Gli elementi dell'insieme in uscita devono essere ordinati in modo non decresecnte (secondo la relazione d'ordine)
2. Gli elementi dell'insieme in uscita devono essere una permutazione degli elementi dell'insieme in entrata

In questo documento la relazione d'ordine sarà di tipo numerico ascendente e gli insiemi saranno rappresentati da liste numerate.

2 Algoritmi di ordinamento

2.1 Insertion-Sort

L'Insertion-Sort è un algoritmo di ordinamento basato sui confronti che opera in modo elementare, come una persona qualsiasi ordinerebbe un mazzo di carte: seleziona uno alla volta tutti gli elementi della lista partendo dal secondo e lo confronta con quelli precedenti posizionandolo al suo posto secondo la relazione d'ordine. È un tipo di algoritmo che si definisce *in place* per la caratteristica di non aver bisogno di utilizzare una lista di appoggio per la sua esecuzione.

```
def insertion_sort(self):

    for j in range(1, len(self.array)):
        key = self.array[j]
        i = j - 1
        while i >= 0 and self.array[i] > key:
            self.array[i + 1] = self.array[i]
            i = i - 1
        self.array[i + 1] = key
```

Listing 1: Frammento di codice dell'algoritmo di ordinamento *Insertion-Sort*

2.2 Merge-Sort

Merge-Sort è un algoritmo di ordinamento basato sui confronti che opera secondo l'approccio *Divide et Impera*, dividendo ricorsivamente il problema in sottoproblemi della stessa natura e di ordine inferiore, per poi unirli dopo essere stati risolti (fase di merge). Nel caso presentato la lista in input viene divisa in modo ricorsivo in sottoliste di ordine sempre minore, le sottoliste di ordine minimo vengono ordinate e successivamente unite fino a generare la lista ordinata in output.

```
def merge_sort(self, a, b):

    if b - a > 0:
        q = math.floor((a + b) / 2)

        self.merge_sort(a, q)
        self.merge_sort(q + 1, b)

        self.merge(a, q, b)

def merge(self, a, q, b):

    l = []
    r = []

    for i in range(a, q + 1):
        l.append(self.array[i])

    for i in range(q + 1, b + 1):
        r.append(self.array[i])

    l.append(len(self.array)+1)
    r.append(len(self.array)+1)

    j = 0
    k = 0

    for i in range(a, b + 1):
        if l[j] <= r[k]:
            self.array[i] = l[j]
            j = j + 1
        else:
            self.array[i] = r[k]
            k = k + 1
```

Listing 2: Frammento di codice dell'algoritmo di ordinamento *Merge-Sort*

3 Analisi delle prestazioni

Secondo la teoria a seconda del tipo di input i due algoritmi possono risultare più o meno efficienti in termini di complessità computazionale. In questa relazione verranno presentati tre casi riassuntivi: lista ordinata, ordinata al contrario e riempita in modo casuale.

Input	Merge-Sort	Insertion-Sort
Ordinato	$O(n \log n)$	$O(n)$
Ordinato al contrario	$O(n \log n)$	$O(n^2)$
Riempito in modo casuale	$O(n \log n)$	$O(n^2)$

Quanto è riportato nella tabella descrive l'aspettativa sui tempi medi di ordinamento degli algoritmi a seconda della dimensione e del tipo di input. Come si può vedere l'algoritmo di

ordinamento Merge-Sort risulta ordinare con lo stesso tempo medio per tutti i tipi di input, mentre l'Insertion-Sort è più efficiente nell'ordinamento di liste parzialmente ordinate ma meno ottimale per liste ordinate in modo decrescente o completamente casuali.

4 Test delle prestazioni

Il progetto allegato alla relazione si occupa di testare i due algoritmi utilizzando liste che si comportano come array, mostrando i grafici dei tempi di ordinamento applicato ai tre casi di input introdotti precedentemente per liste di dimensione crescente che partono da 200 elementi fino a raggiungere i 4000 elementi con un passo di 200 elementi. Per garantire la correttezza del risultato, per ogni dimensione fissata e per ogni tipo di lista, vengono registrati i tempi di ordinamento di 100 casi e successivamente calcolata la media. Gli esperimenti sono stati eseguiti su una piattaforma con le seguenti specifiche:

- Processore: Intel® Core™ i7-1065G7 (frequenza di base 1,3 GHz, fino a 3,9 GHz con tecnologia Intel® Turbo Boost, 8 MB di cache, 4 core)
- Ram: SDRAM DDR4-2666 da 16 GB (1 x 16 GB)
- Sistema Operativo: Windows 10 Home 64

Di seguito vengono presentate tre tabelle che confrontano i tempi di ordinamento dei due algoritmi per stesso tipo di input, e un grafico che mostra come si comportano i due algoritmi per tipi di input diversi.

4.1 Inserimento Crescente

Dimensione	Merge-Sort	Insertion-Sort
200	0.0008	4.10e-05
400	0.0018	8.12e-05
600	0.0025	0.0001
800	0.0034	0.0001
1000	0.0044	0.0002
1200	0.0052	0.0002
1400	0.0062	0.0003
1600	0.0071	0.0003
1800	0.0081	0.0003
2000	0.0091	0.0004
2200	0.0101	0.0004
2400	0.0110	0.0004
2600	0.0122	0.0005
2800	0.0131	0.0006
3000	0.0141	0.0006
3200	0.0153	0.0006
3400	0.0161	0.0007
3600	0.01715	0.0007
3800	0.0183	0.0008
4000	0.0194	0.0008

Table 1: Merge-Sort e Insertion-Sort a confronto per liste ordinate

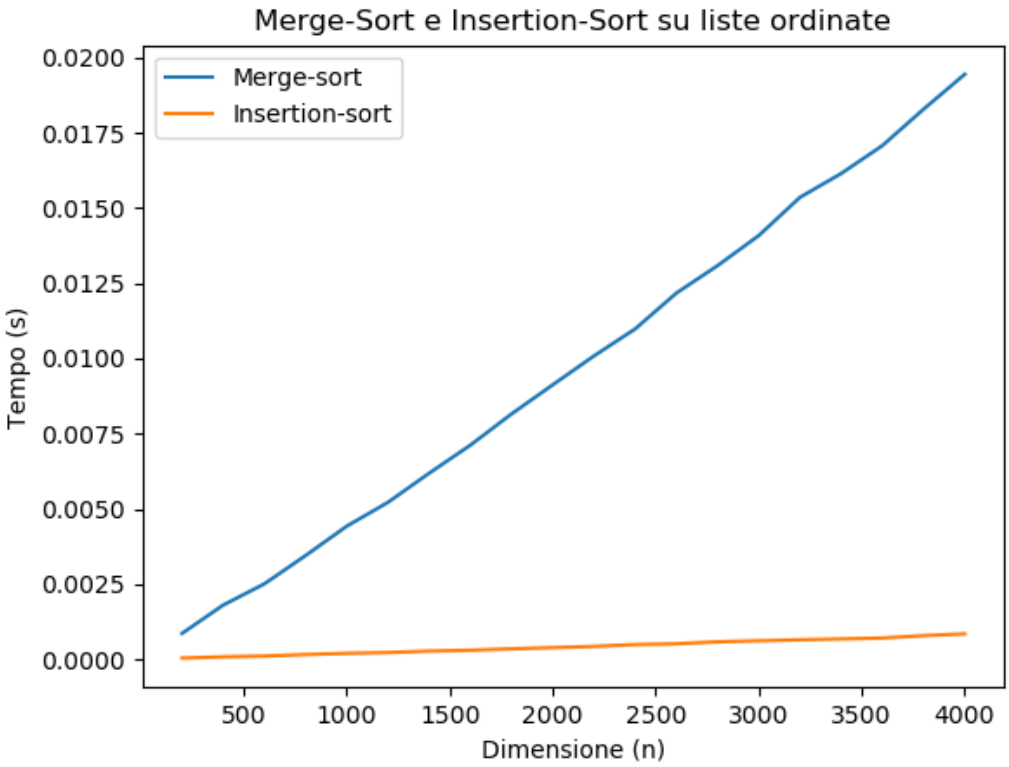

Figure 1: Grafico che descrive i tempi medi di ordinamento dei due algoritmi in relazione alla dimensione di liste ordinate in modo crescente

4.2 Inserimento Decrescente

Dimensione	Merge-Sort	Insertion-Sort
200	0.0008	0.0035
400	0.0018	0.0134
600	0.0025	0.0278
800	0.0034	0.0520
1000	0.0044	0.0794
1200	0.0051	0.1164
1400	0.0062	0.1598
1600	0.0072	0.2095
1800	0.0080	0.2655
2000	0.0091	0.3285
2200	0.0099	0.4022
2400	0.0110	0.4774
2600	0.0120	0.5608
2800	0.0129	0.6520
3000	0.0141	0.7543
3200	0.0151	0.8566
3400	0.0161	0.9625
3600	0.0171	1.0800
3800	0.0182	1.2072
4000	0.0192	1.3409

Table 2: Merge-Sort e Insertion-Sort a confronto per liste ordinate inversamente

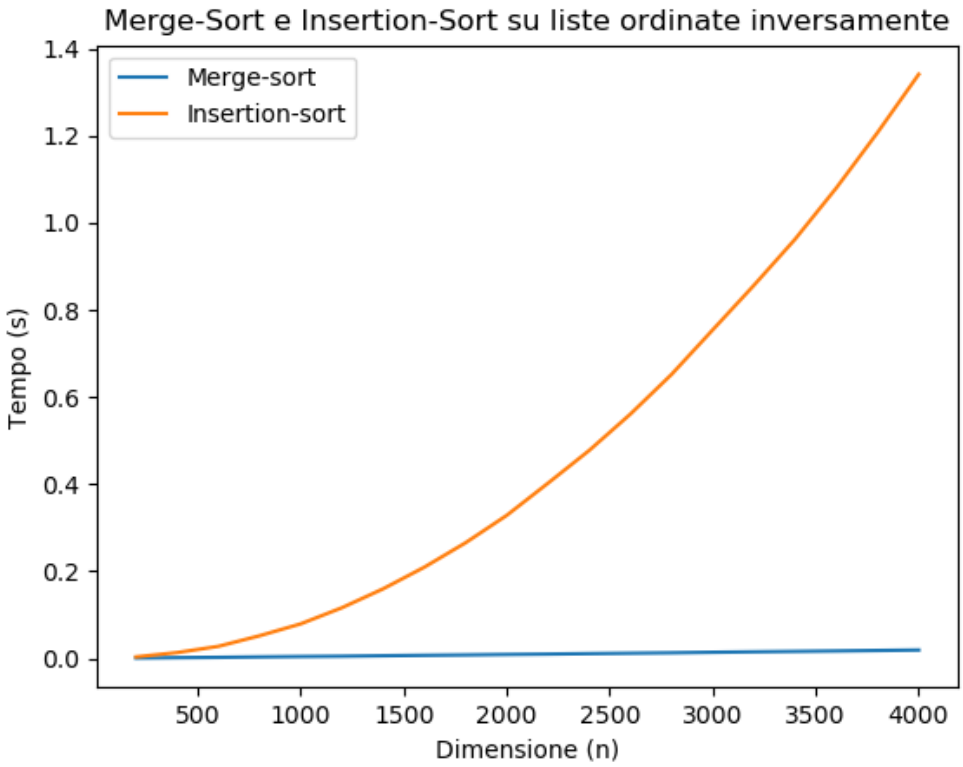

Figure 2: Grafico che descrive i tempi medi di ordinamento dei due algoritmi in relazione alla dimensione di liste ordinate in modo decrescente

4.3 Inserimento Casuale

Dimensione	Merge-Sort	Insertion-Sort
200	0.0008	0.0018
400	0.0018	0.0072
600	0.0025	0.0143
800	0.0035	0.0269
1000	0.0044	0.0410
1200	0.0052	0.0602
1400	0.0063	0.0829
1600	0.0072	0.1082
1800	0.0082	0.1370
2000	0.0092	0.1685
2200	0.0102	0.2067
2400	0.0112	0.2466
2600	0.0121	0.2874
2800	0.0131	0.3340
3000	0.0143	0.3843
3200	0.0153	0.4363
3400	0.0162	0.4899
3600	0.0173	0.5502
3800	0.0183	0.6155
4000	0.0196	0.6821

Table 3: Merge-Sort e Insertion-Sort a confronto per liste casuali

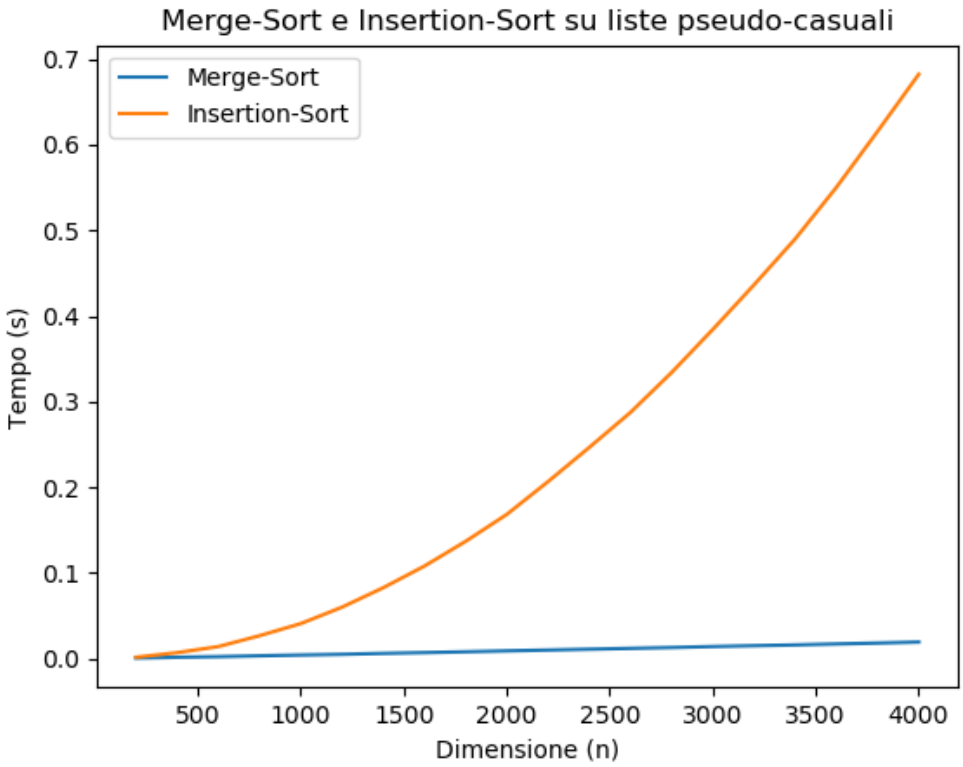

Figure 3: Grafico che descrive i tempi medi di ordinamento dei due algoritmi in relazione alla dimensione di liste costruite in modo pseudo-casuale

4.4 Risultati grafici

Si riporta il grafico descritto nel paragrafo precedente:

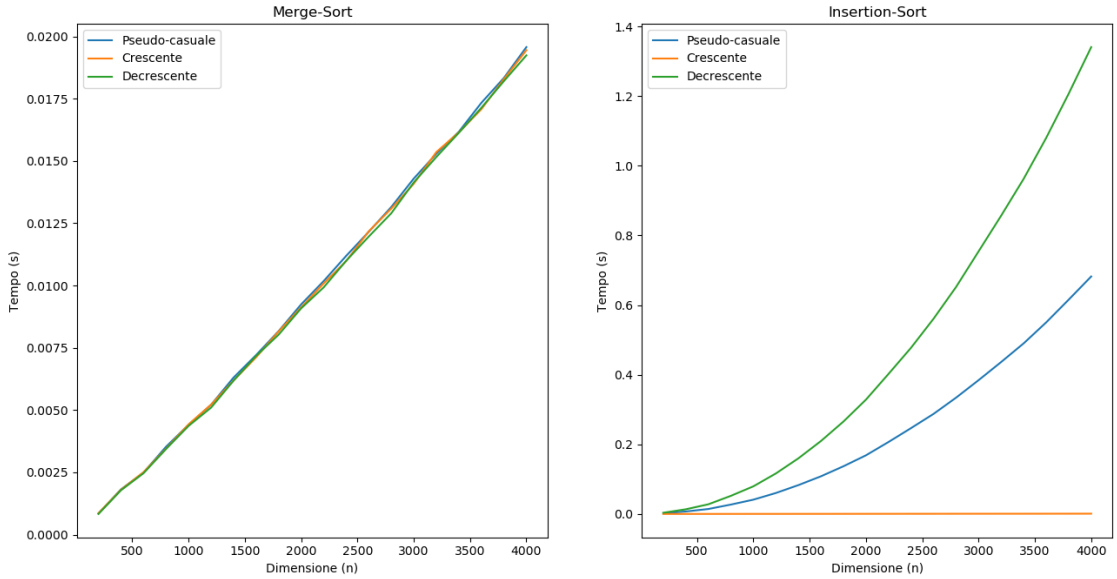

Figure 4: Grafici che descrivono il tempo di ordinamento in base alla dimensione di input, distinti in modo da evidenziare caso migliore, peggiore e medio.

5 Conclusioni

Come si evince dai grafici precedenti gli algoritmi di ordinamento considerati confermano le aspettative della teoria. Infatti l'algoritmo di ordinamento Merge-Sort ordina le liste con gli stessi tempi: non esiste quindi un caso migliore, e il tipo di input non influenza l'esito dei tempi di ordinamento. Al contrario l'algoritmo di ordinamento Insertion-Sort non è ottimizzato per liste ordinate al contrario e casuali, però per liste ordinate o parzialmente ordinate ha tempi di ordinamento inferiori rispetto al Merge-Sort.