Insertion sort vs Merge sort

Salvatore Baglieri

Luglio 2020

1 Introduzione

L'obiettivo è quello di valutare la performance di due particolari algoritmi di ordinamento, Merge sort e Insertion sort.
I test di seguito sono eseguiti in python.
Innanzitutto analizziamo gli algoritmi in questione:

1.1 Insertion sort

```
1  InsertionSort(A):
2    for j = 2 to A.length do
3      key = A[j]
4      i = j − 1
5      while i > 0 and A[i] > key do
6        A[i + 1] = A[i]
7        i = i + 1
8      end while
9      A[i + 1] = key
10   end for
```

L'insertion sort funziona in modo simile al modo in cui si ordinano le carte da gioco tra le mani. Si assume che la sequenza da ordinare sia partizionata in una sottosequenza già ordinata, all'inizio composta da un solo elemento, e una ancora da ordinare. Alla k-esima iterazione, la sequenza già ordinata contiene k elementi. In ogni iterazione, viene rimosso un elemento dalla sottosequenza non ordinata e inserito nella posizione corretta della sottosequenza ordinata, estendendola così di un elemento.

1.2 Merge sort

```
1  MergeSort(A,p,r):
2    if p < r then
3      q = ⌊(p + r)/2⌋
4      MergeSort(A, p, q)
5      MergeSort(A, q + 1, r)
6      Merge(A, p, q, r)
7    end if
```

```
1  Merge(A,p,q,r):
2    n = 1 − p + 1
3    m = r − q
4    Crea due nuovi array L[1...n + 1] e R[1...m + 1]
5    for i = 1 to n do
6      L[i] = A[p + i − 1]
7    end for
8    for j = 1 to m do
9      R[j] = A[q + j]
10   end for
11   L[n + 1] = ∞
12   R[m + 1] = ∞
13   i = 1
14   j = 1
15   for k = p to r do
16     if L[i] ≤ R[j] then
17       A[k] = L[i]
18       i = i + 1
19     else
20       A[k] = R[j]j = j + 1
21     end if
```

Il merge sort è un algoritmo basato su confronti che utilizza un processo di risoluzione ricorsivo, sfruttando la tecnica del Divide et Impera, che consiste nella suddivisione del problema in sottoproblemi della stessa natura di dimensione via via più piccola. L'algoritmo opera nel modo seguente:
La sequenza viene divisa (divide) in due metà. Ognuna di queste sottosequenze viene ordinata, applicando ricorsivamente l'algoritmo (impera) Le due sottosequenze ordinate vengono fuse (combina). Per fare questo, si estrae ripetutamente il minimo delle due sottosequenze e lo si pone nella sequenza in uscita, che risulterà ordinata.

2 Aspettative

Dato un array di una qualsiasi dimensioni, di sicuro l'output per entrambi gli algorimi sarà identico, cioè l'array ordinato. La sostanziale differenza tra i due sta nel tempo in cui questo array viene ordinato. Per quanto riguarda l'insertion sort, in caso di array già ordinato mi aspetto un tempo di computazione lineare n, mentre per il caso medio (array con valori casuali) un tempo proporzionale a n^2. Per quanto riguarda invece il merge sort, in entrambi i casi il tempo di computazione sarà proporzionale a $n \cdot logn$.

3 Test

I test sono scritti in python, applicando ogni algoritmo su un array di dimensioni crescenti da 100 a 5000, con incrementi di 100. Ogni test è la media di 10 iterazioni dello stesso problema.

3.1 Test array con elementi random

Analizziamo il caso di un array con n elementi random.
Di seguito viene riportata la tabella con i dati numerici forniti dal programma.

Numero elementi	Insertion sort	Merge sort
100	0.000274	0.000268
300	0.002092	0.000810
500	0.006351	0.001494
700	0.013533	0.002193
900	0.022560	0.002945
1100	0.033618	0.003580
1300	0.049300	0.004364
1500	0.063933	0.005125
1700	0.082095	0.005880
1900	0.104735	0.006566
2100	0.129992	0.007481
2300	0.155192	0.008224
2500	0.182831	0.009173
2700	0.233384	0.010692
2900	0.261519	0.013430
3100	0.292233	0.011984
3300	0.332240	0.012804
3500	0.373685	0.013692
3700	0.424808	0.014802
3900	0.465859	0.015432
4100	0.513574	0.016307
4300	0.564957	0.017198
4500	0.633343	0.018395
4700	0.677158	0.019096
4900	0.729514	0.019698
5000	0.764516	0.020342

Si nota facilmente che i tempi divergono velocemente. Infatti, con un array di 300 elementi vediamo già che i tempi differiscono di un ordine di grandezza.

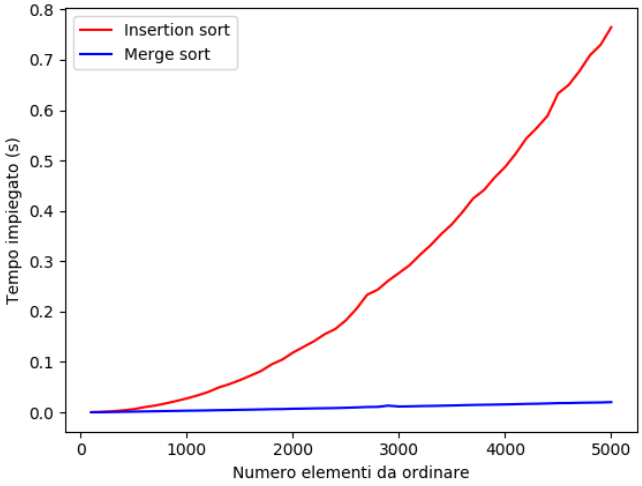

Figure 1: Prestazioni a confronto su un array con valori random

Dal grafico sopra riportato notiamo che i due algoritmi differiscono molto all'aumentare di n (numero degli elementi), tanto da far sembrare il tempo di Merge sort quasi nullo.

3.2 Test array ordinato

Consideriamo adesso un array già ordinato.

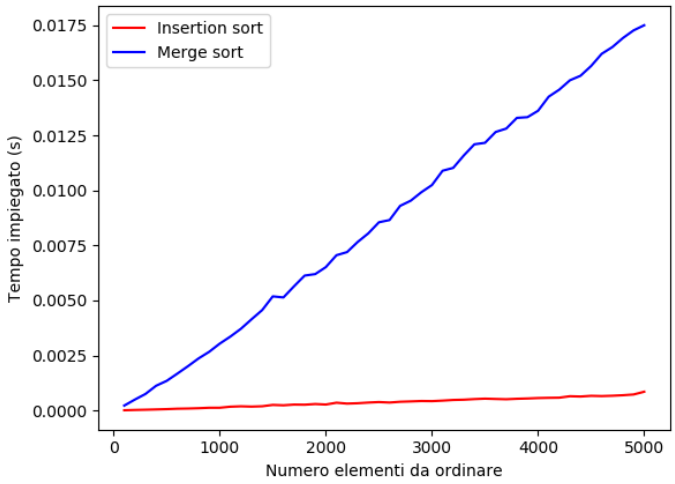

Figure 2: Prestazioni a confronto su un array ordinato

Notiamo subito che rispetto al caso precedente la situazione è cambiata. Adesso l'Insertion sort è più veloce del Merge sort, impiegando un tempo quasi nullo.
Di seguito viene riportata la tabella con i dati numerici dei test.

Numero elementi	Insertion sort	Merge sort
100	0.000012	0.000224
300	0.000036	0.000752
500	0.000063	0.001347
700	0.000090	0.002011
900	0.000125	0.002665
1100	0.000173	0.003358
1300	0.000178	0.004145
1500	0.000254	0.005184
1700	0.000269	0.005642
1900	0.000294	0.006196
2100	0.000354	0.007053
2300	0.000331	0.007656
2500	0.000384	0.008549
2700	0.000397	0.009295
2900	0.000431	0.009915
3100	0.000449	0.010890
3300	0.000491	0.011584
3500	0.000538	0.012159
3700	0.000512	0.012805
3900	0.000547	0.013323
4100	0.000576	0.014254
4300	0.000649	0.014998
4500	0.000667	0.015649
4700	0.000671	0.016507
4900	0.000728	0.017265
5000	0.000853	0.017499

4 Conclusione

Abbiamo visto nella figura 1 che in generale il Merge sort è più veloce dell'Insertion sort. In particolare, i test effettuati confermano le aspettative. Infatti nel caso di un array random abbiamo che il tempo di ordinamento per l'Insertion sort è esponenziale (come si nota dal grafico), mentre quello del Merge sort è quasi lineare. Un caso particolare si ha quando l'array è già ordinato. In questo caso il tempo di ordinamento per il Merge sort rimane invariato, mentre quello dell'Insertion sort è lineare.