



16 LUGLIO 2014


ANALISI TEMPORALE ALGORITMI DI ORDINAMENTO

RELAZIONE ESERCIZIO 5

MAURO SABATINO

MATRICOLA 736724

Corso B



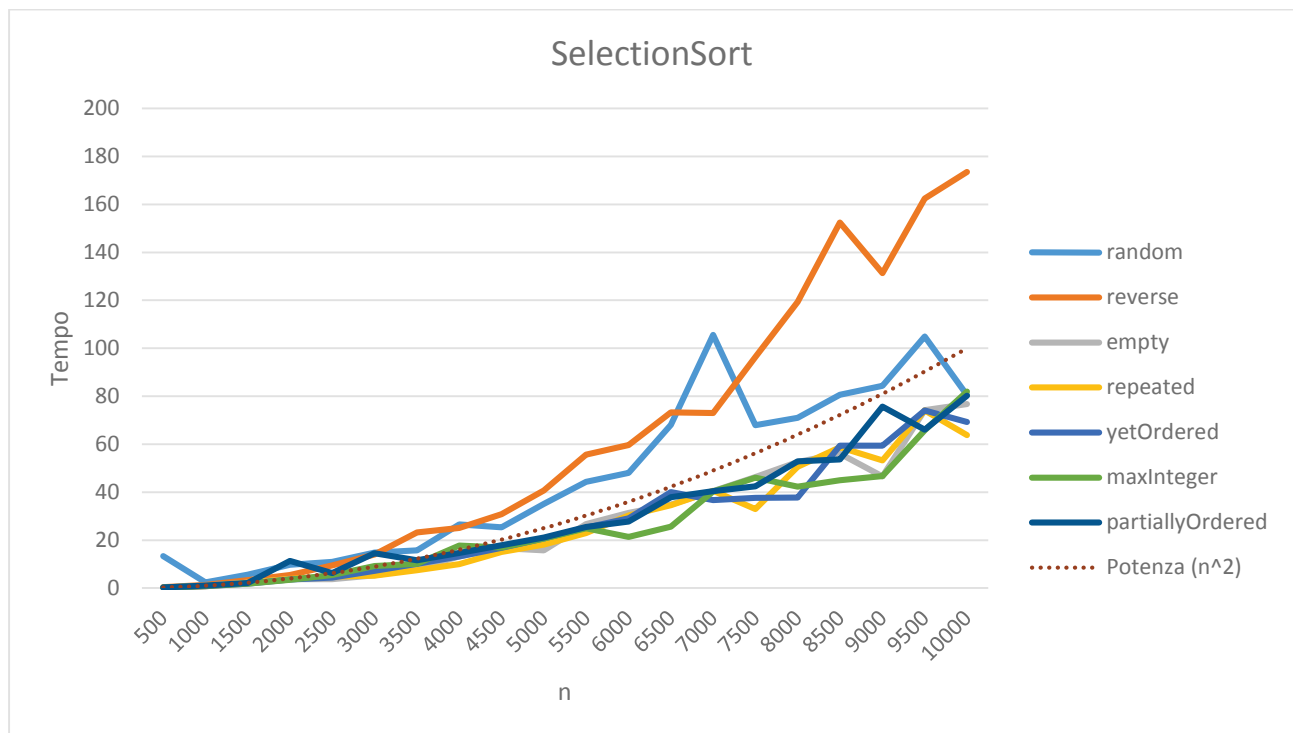
Il compito 5 consiste nell'implementare una versione a scelta dei 4 principali algoritmi di ordinamento per confronti studiati all'interno del corso. Ho analizzato il comportamento dei seguenti algoritmi nei casi:

- Array riempito con elementi casuali generati dalla funzione random
- Array di elementi casuali, ordinato con la classe di java Arrays.sort() e poi l'ho invertito
- Array con un elemento ripetuto
- Array vuoto inizializzato da java con tutti 0
- Array riempito con l'elemento Integer.MAX_VALUE
- Array di elementi casuali già ordinato con la classe java Arrays.sort()
- Array parzialmente ordinato

Ho analizzato array di lunghezza da 500 a 10000 con step di 500 elementi aggiunti ad ogni iterazione. Dopo aver prodotto con un metodo java un file .csv in cui registravo il tempo di esecuzione dei vari algoritmi per le varie dimensioni ho prodotto dei grafici per osservarne il comportamento a livello asintotico. L'analisi è supportata da grafici realizzati con uno spreadsheet (excel).

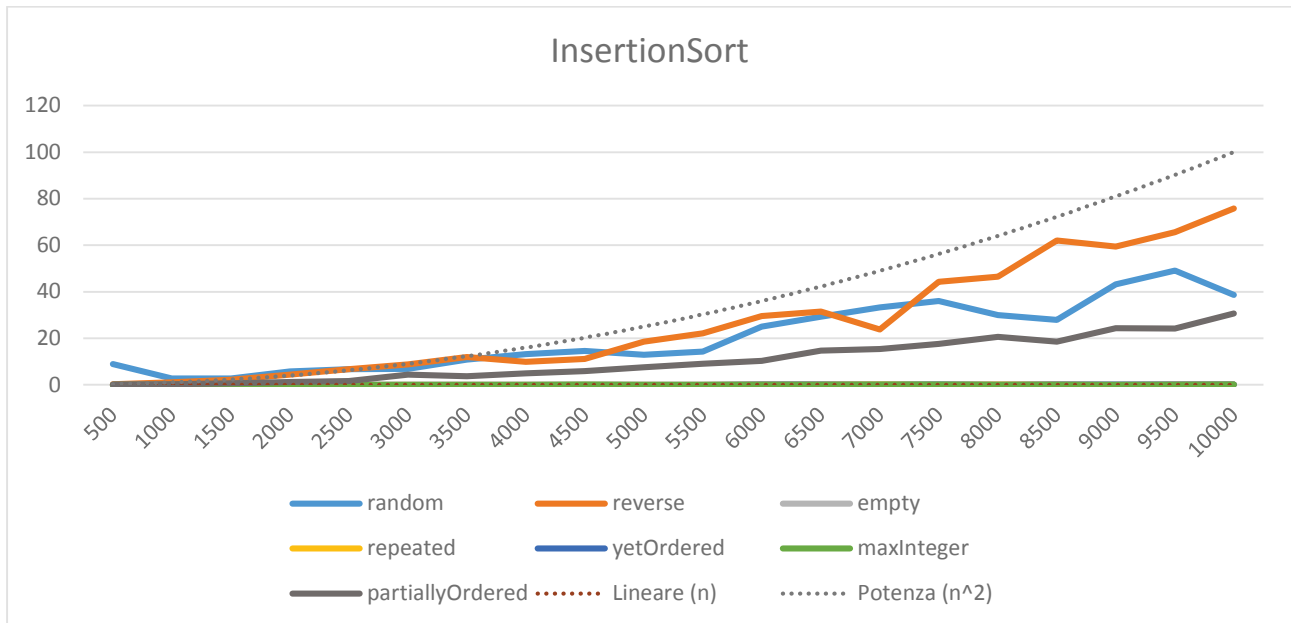
Analisi per algoritmo

Selection Sort



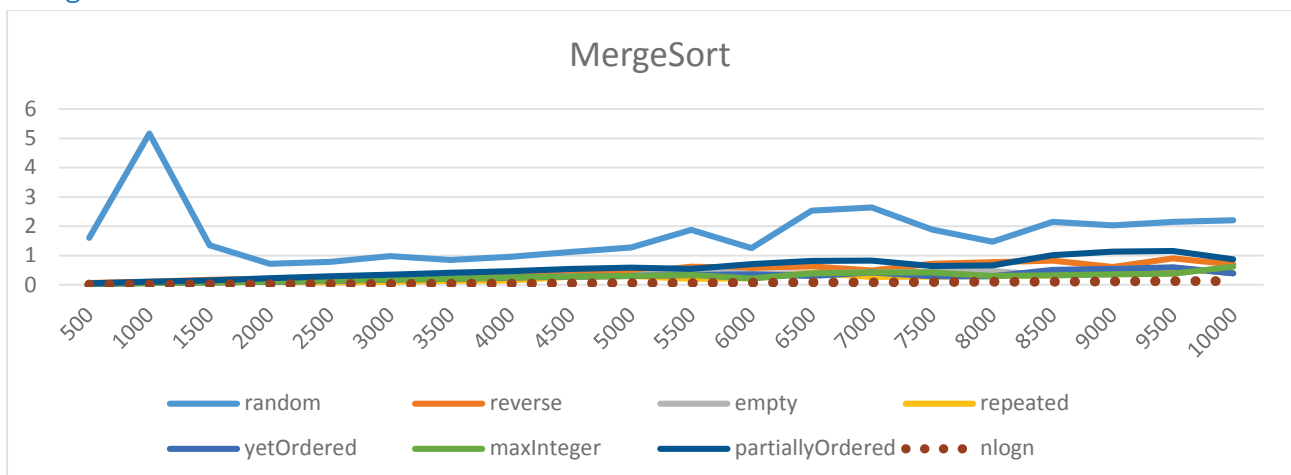
Durante il corso abbiamo studiato il comportamento asintotico del selectionSort e sappiamo che in tutti i tre casi ($T_{best}(n)$, $T_{avg}(n)$, $T_{worst}(n)$) l'algoritmo ha un comportamento asintotico $\Theta(n^2)$. Questo viene anche confermato dal grafico, infatti per ogni tipologia di vettore fornito, il comportamento è sempre assimilabile ad un andamento quadratico.

InsertionSort



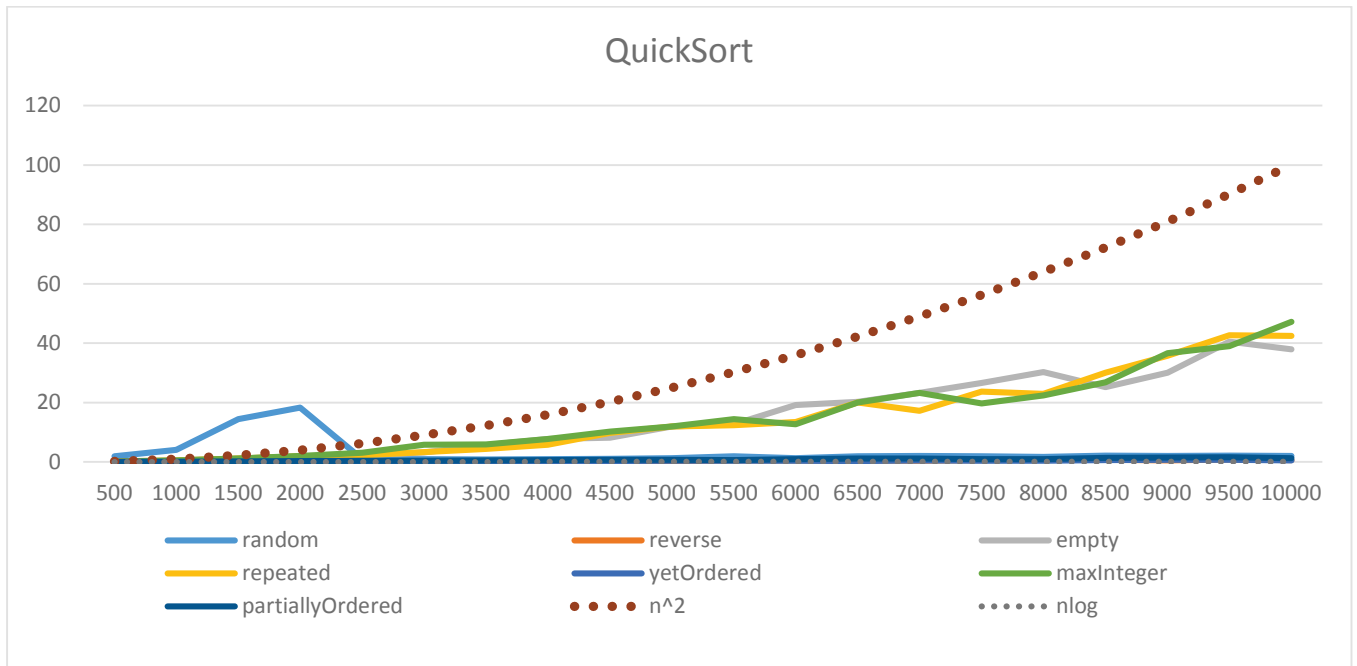
Durante il corso abbiamo studiato l'algoritmo di ordinamento per inserimento, soffermandoci sullo studio della complessità asintotica, esaminando un caso particolare di array da ordinare già ordinato. Questo è stato catalogato come il caso migliore possibile, ogni valore da inserire è maggiore o uguale all'ultimo elemento della parte già ordinata, quindi ogni inserimento costa un solo passo, per n valori farà n passi, quindi la sua complessità nel caso migliore è lineare $\Theta(n)$. Dal grafico si evince chiaramente che gli array contenenti un solo elemento ripetuto oppure quello già ordinato seguendo perfettamente l'andamento lineare. I casi peggiori e medio invece crescono in modo quadratico e si evince chiaramente dall'array random, partiallyOrdered e reverse. Il caso dell'array ordinato al contrario è proprio un caso peggiore, ad ogni inserimento il valore da inserire sarà confrontato con tutti gli elementi della parte ordinata.

MergeSort



Lo studio del mergeSort ha prodotto un andamento asintotico pari a $\Theta(n \log n)$ in tutti e tre i casi (peggiore, medio, migliore). E nel grafico è abbastanza chiaro questo comportamento simile in ogni caso per ogni tipo di array (fatta eccezione per l'array random, ma deve esser causato da qualche altro fattore durante l'esecuzione e non all'implementazione dell'algoritmo, in quanto in tutti gli altri casi è abbastanza regolare). In questa implementazione dell'algoritmo non abbiamo caso peggiore, medio e migliore (ho deciso per questo esercizio di implementare il mergesort "ecologico" che quindi non apporta alcuna miglioria temporale in determinate condizioni).

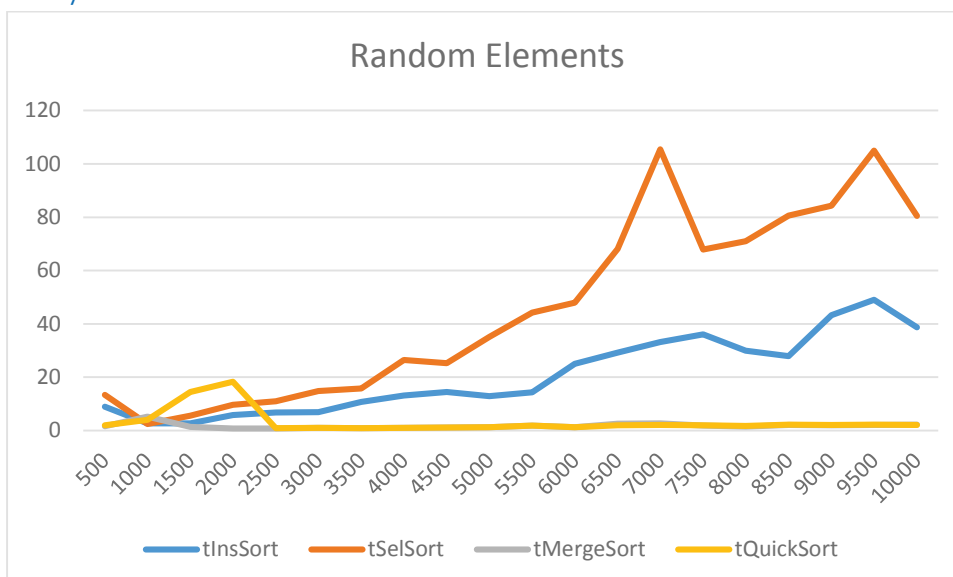
Quicksort



Per questo esercizio ho deciso di implementare la versione Crescenzi del quicksort, il quale mette in entrambe le parti gli elementi uguali al pivot, ma non risolve il caso di array composti da elementi tutti uguali (perché li mette tutti subito nella prima partizione rendendo l'array totalmente sbilanciato). Infatti come si evince dal grafico gli array con elementi ripetuti seguono proprio un andamento $\theta(n^2)$. Il caso di elementi ripetuti è però un caso limite, di sicuro è un gap di questo algoritmo, ma normalmente si ordinano array disordinati e con questi tipi di array si comporta in maniera ottimale $\theta(n \log n)$. Da notare come in questa versione il caso peggiore di una prima implementazione del quicksort, ovvero l'array già ordinato, si comporti come un caso medio, anche dovuto alla scelta del pivot in maniera casuale. Inoltre un importante risultato emerge da questa analisi temporale, generalmente il quicksort è più veloce del mergesort, ma in queste due particolari implementazione è mediamente più veloce il mergesort.

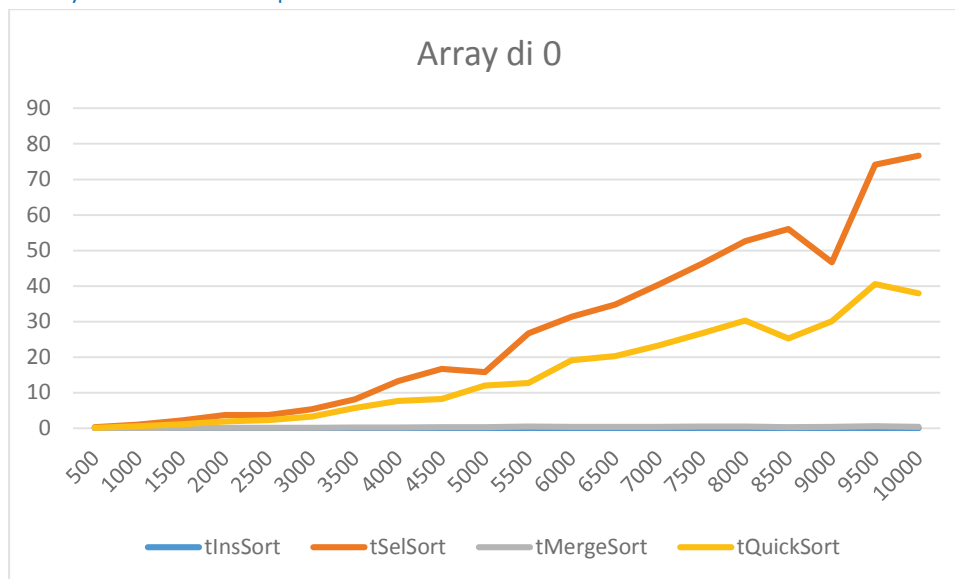
Analisi per tipologia di sequenza in input

Array di elementi casuali

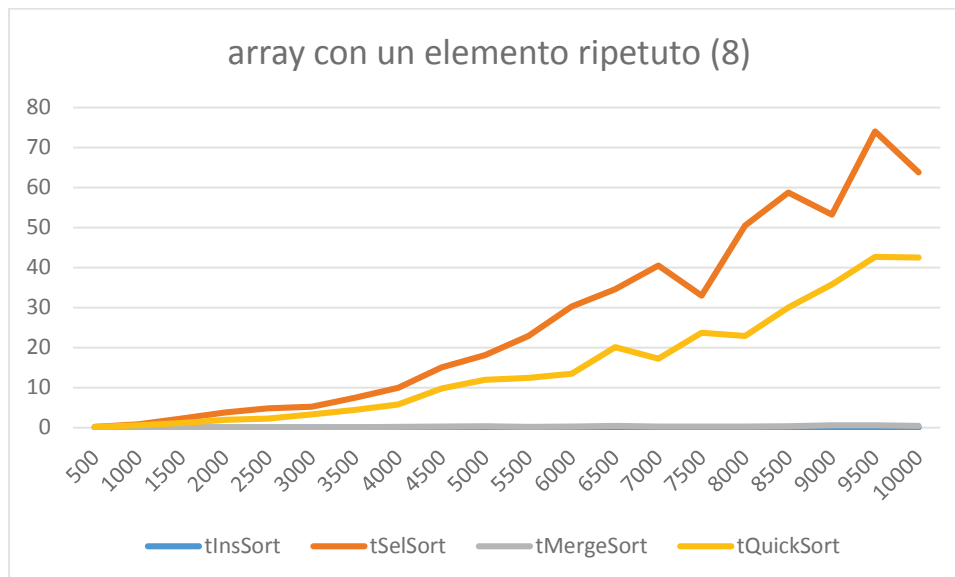


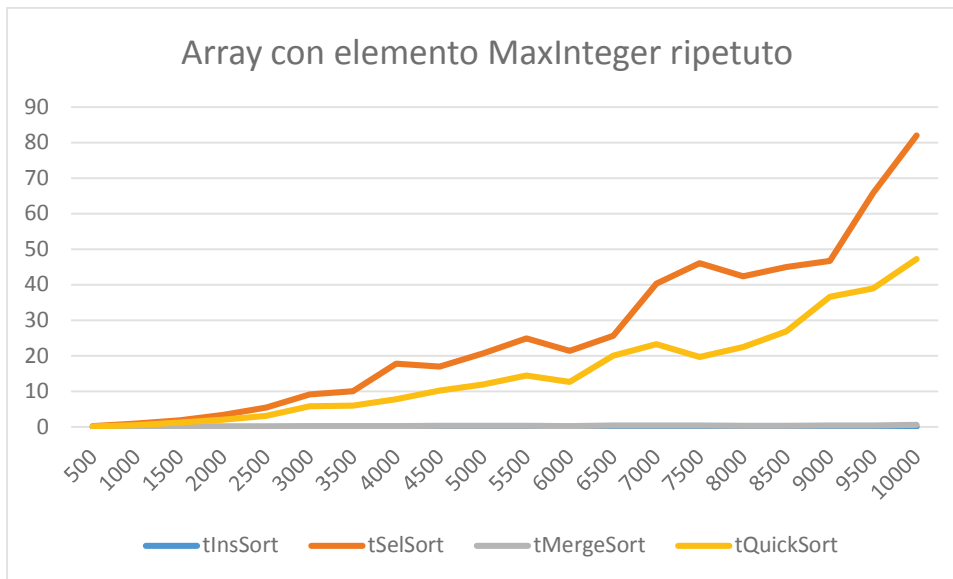
Il quicksort e il mergesort sono nettamente più veloci degli altri due algoritmi, anche studiando questo caso che è un caso medio per tutti e 4, le tesi teoriche non possono che esser confermate.

Array di elementi ripetuti



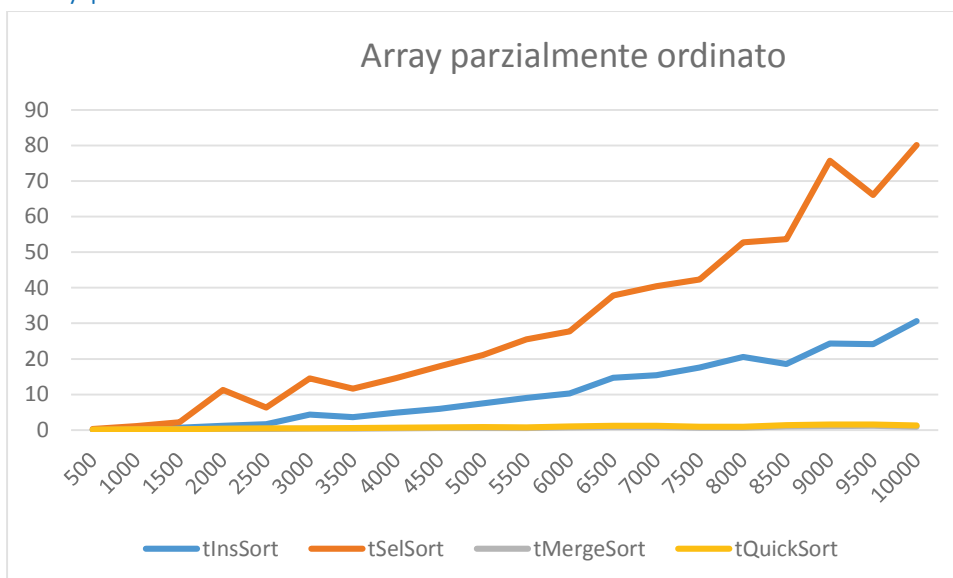
Questo è il caso di elementi ripetuti tutti uguali e si evince chiaramente come sia il caso peggiore del quicksort, ma soprattutto il caso migliore per l'insertion. Il mergesort e selection sort confermano il loro comportamento asintotico $\theta(n \log n)$ e $\theta(n^2)$.





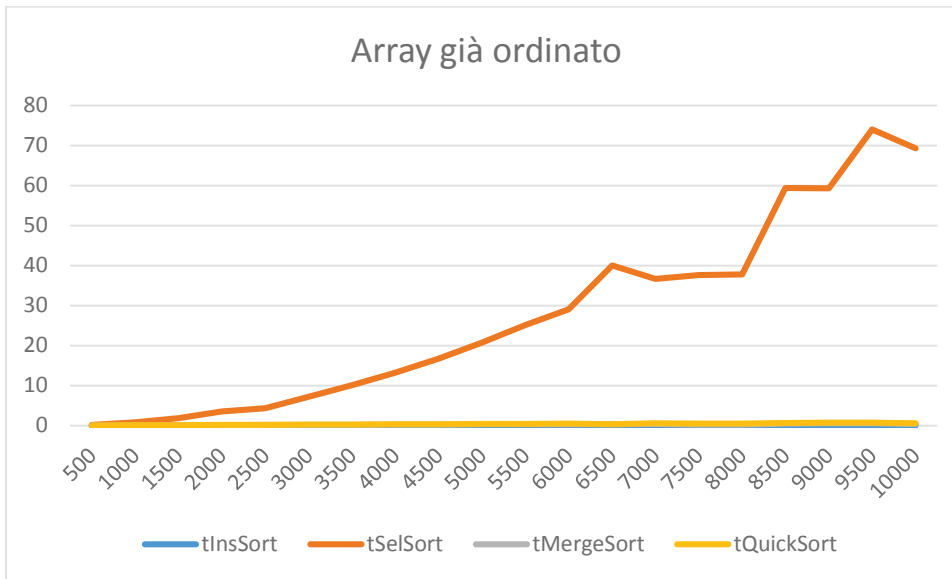
Altri esempi di array con elementi ripetuti e i grafici sono pressoché identici.

Array parzialmente ordinato



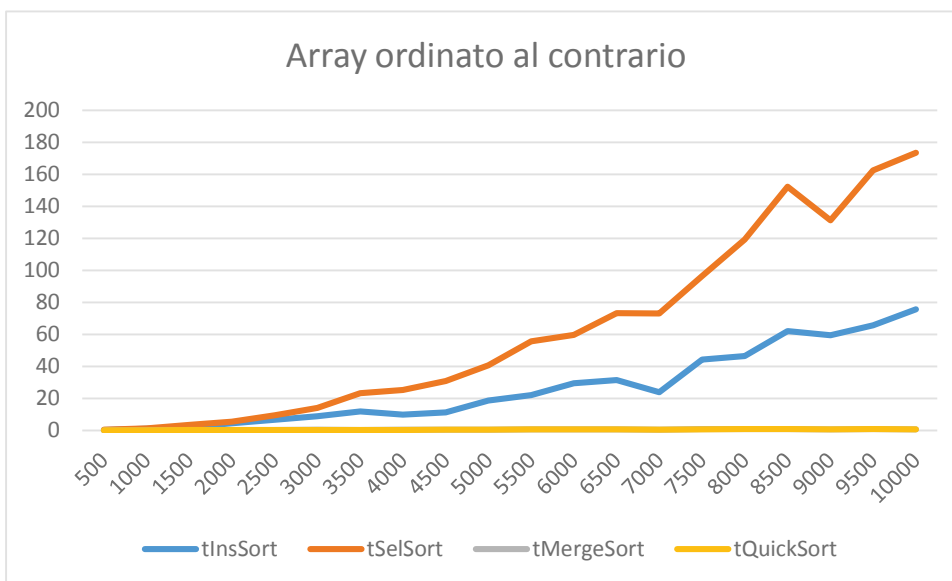
Anche questo può essere considerato un caso medio, in quanto i comportamenti asintotici rispecchiano proprio quelli delle funzioni $T_{avg}(n)$ dei 4 algoritmi.

Array già ordinato



Questo è il caso migliore dell'insertionsort e trova riscontro anche nelle misurazioni effettuate. Invece il quicksort si comporta in maniera ottimale perché nell'ottimizzazione consistente nella scelta del pivot in maniera casuale, si evita di farlo diventare un caso peggiore.

Array ordinato al contrario



Questo caso non provoca nessun caso diverso da quello medio per ognuno dei 4 algoritmi.