Relazione sugli algoritmi di ordinamento

Matteo Querini
G. Bearzi, Udine
matteo.queriniallievi.bearzi.it

INTRODUZIONE

Durante l'anno scolastico 2023-2024 abbiamo studiato diversi algoritmi di ordinamento: il Bubble sort, il Selection sort, l'Insertion sort, il Merge sort e il Quick sort; Ogni algoritmo ha caratteristiche uniche che li rendono adatti a determinati contesti e situazioni. In questa introduzione analizzeremo tutte le caratteristiche di questi diversi algoritmi di ordinamento:

- **Bubble sort**
  Il Bubble sort è uno degli algoritmi di ordinamento più semplici, utilizzato maggiormente per strutture dati di dimensioni ridotte. La sua complessità computazionale varia, nel caso migliore è di O(n) mentre nel caso peggiore è di $O(n^2)$. Questo algoritmo si basa sul confronto degli elementi all'interno della struttura dati. Durante l'esecuzione, gli elementi vengono scambiati se si trovano nell'ordine sbagliato, spostando gradualmente gli elementi più grandi verso la fine della struttura. Questo algoritmo richiede un grande numero di iterazioni per ordinare completamente la struttura dati, il che lo rende meno efficiente per strutture dati di grandi dimensioni.

- **Selection sort**
  Il Selection Sort è un algoritmo di ordinamento con complessità di $O(n^2)$. Questo algoritmo divide la struttura dati in due parti: una parte già ordinata e una ancora da ordinare. Inizialmente, l'intera sequenza è considerata non ordinata e durante ogni iterazione, l'algoritmo trova l'elemento più piccolo nella parte non ordinata mediante confronti e lo scambia con il primo elemento non ordinato. Questo processo continua fino a quando tutti gli elementi sono stati spostati nella parte ordinata.

- **Insertion sort**
  L'Insertion sort è un algoritmo di ordinamento con complessità di $O(n^2)$. Questo algoritmo è in place, perché non usa delle strutture dati di supporto, bensì lavora sulla struttura dati stessa. Il suo funzionamento consiste nell'avere due indici, uno punta all'elemento da ordinare, mentre l'altro indica l'elemento immediatamente precedente infine confronta i due elementi e li scambia se si trovano nell'ordine sbagliato. L'Insertion non è molto efficace, infatti viene usato unicamente nelle strutture dati parzialmente ordinati, nelle quali riesce ad ordinare più velocemente.

- **Merge sort**
  Il Merge sort è un algoritmo di ordinamento con complessità θ (n log n),questo algoritmo è stabile, perché mantiene l'ordine relativo fra gli elementi con la stessa chiave, inoltre, a differenza dell'Insertion sort, non è in place, infatti il Merge sort usa strutture dati di supporto, infine, la caratteristica principale di questo algoritmo è il fatto che è ricorsivo, ciò significa che viene richiamato nella sua stessa funzione creando degli ambienti. Il Merge sort si vivide in due funzioni: il Merge sort, che si richiama ricorsivamente e il merge, il quale ordina effettivamente la struttura dati tramite dei confronti.
  Il funzionamento del Merge Sort consiste nel dividere la struttura dati non ordinata in due sottostrutture dati, ciascuna con dimensioni dimezzate rispetto l'originale. Questa suddivisione continua finché non si arriva a un singoletto, il quale è per forza ordinato. Successivamente, si passa al confronto e all'unione in ordine crescente dei singoletti in una nuova struttura dati.
  Grazie a queste caratteristiche, il Merge sort, viene usato principalmente in strutture dati con grandi dimensioni ma soprattutto quando si vuole mantenere l'ordine relativo degli elementi.

- **Quick sort**
  Il Quick sort l'algoritmo di ordinamento più utilizzato, è basato sul confronto ricorsivo, come il Merge sort si richiama ricorsivamente, è in place, cioè non usa strutture dati di supporto, a differenza col precedente algoritmo,

il Quick sort non è stabile, infatti non mantiene l'ordine relativo degli elementi con la stessa chiave, infine, la sua complessità computazionale parte da $\Omega(n \log n)$ e aumenta progressivamente fino a raggiungere $O(n^2)$ quando la struttura dati è ordinata in modo inverso.
Il suo funzionamento consiste nell'assegnare a un valore della struttura dati una variabile pivot, la quale dividerà quest'ultima in due parti. In seguito, viene ordinata la struttura dati in modo che i valori maggiori del pivot siano alla sua destra e quelli minori a sinistra. Dopo questa operazione, l'indice pivot viene scambiato di posto con l'elemento dei valori più alti; queste operazioni vengono eseguite ricorsivamente fino a raggiungere l'ordinamento della struttura dati.

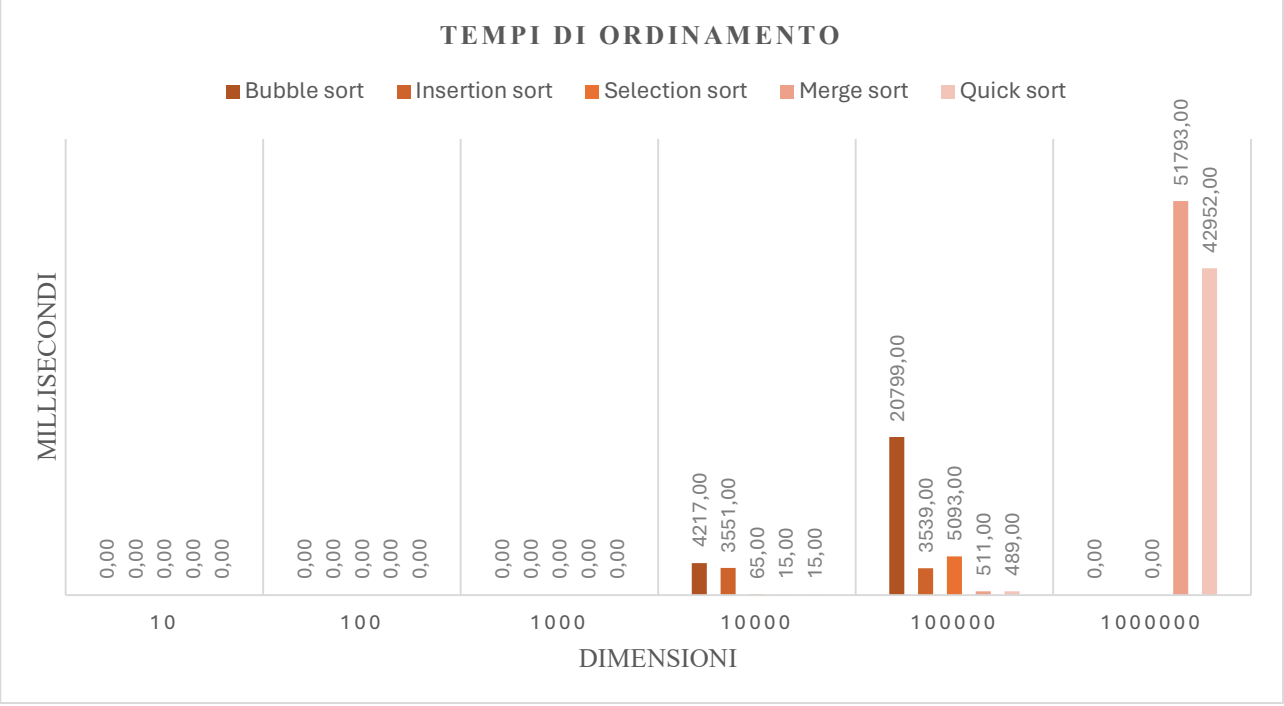
IPOTESI

In base alle informazioni e alle caratteristiche degli algoritmi di ordinamento riportate nell'introduzione, possiamo formulare alcune ipotesi sui risultati riguardanti il tempo di ordinamento del Bubble sort, Selection sort, Insertion sort, Merge sort e Quick sort:

- **Bubble sort**: Ci si aspetta che sia l'algoritmo più lento tra quelli studiati, poiché richiede un gran numero di confronti e scambi, specialmente su strutture dati di grandi dimensioni.
- **Selection sort**: Pur avendo una complessità simile a quella del Bubble sort, potrebbe eseguire un numero di scambi inferiore a quest'ultimo, ma comunque non sarà veloce su grandi strutture dati.
- **Insertion sort**: Potrebbe essere più efficiente degli altri su strutture dati già parzialmente ordinate, ma meno efficiente su strutture dati completamente disordinate
- **Merge sort**: Si calcola di essere più efficiente rispetto agli algoritmi precedenti, specialmente su strutture dati di grandi dimensioni, grazie alla sua complessità θ (n log n).
- **Quick sort**: Anche se può raggiungere una complessità pari a $O(n^2)$ nel caso peggiore, si stima che sia molto efficiente, specialmente su strutture dati disordinate.

RISULTATI

Stando ai risultati delle misurazioni degli algoritmi di ordinamento del grafico 1, si può vedere come le prestazioni variano al variare delle dimensioni degli array. Algoritmi come Bubble, Insertion e Selection sort sono veloci per input piccoli ma con input più grandi richiedono molto più tempo per ordinare. Al contrario, il Merge e Quick sort mantengono prestazioni migliori anche se le dimensioni incrementano.



Graf. 1 Riportamento grafica dei tempi di ordinamento

DISCUSSIONE

Stando ai dati ottenuti dalle misurazioni degli algoritmi di ordinamento, e alle ipotesi formulate in precedenza si conferma che il Bubble sort è l'algoritmo più lento tra quelli studiati. Questo è coerente con le sue caratteristiche, infatti richiede un elevato numero di confronti e scambi, rendendolo inefficace su strutture dati di dimensioni elevate. La sua lentezza aumenta notevolmente all'aumentare delle dimensioni, come previsto.
Riguardo a Selection sort, pur avendo una complessità simile al Bubble sort, i tempi di esecuzione confermano che è molto lento in strutture dati di grandi dimensioni. Anche se esegue un numero inferiore di iterazioni rispetto al Bubble sort.
Per quanto riguarda Insertion sort, l'ipotesi che sia più efficiente su strutture dati parzialmente ordinati si conferma nei risultati ottenuti, tuttavia, come per gli altri due algoritmi risulta meno efficiente su strutture dati completamente disordinati, confermando le precedenti ipotesi.
Il Merge sort, con la sua complessità $\theta (n \log n)$, si rivela efficiente su strutture dati di grandi dimensioni, come previsto. I risultati ottenuti mostrano che mantiene tempi di esecuzione costantemente inferiori rispetto agli altri tre algoritmi, confermando l'ipotesi formulata.
Infine, il Quick sort si dimostra efficiente, specialmente su strutture dati disordinati, nonostante possa raggiungere una complessità $O(n^2)$ nel caso peggiore

CONCLUSIONE

Abbiamo effettuato delle misurazioni su degli algoritmi di ordinamento (Bubble sort, Insertion sort, Selection sort, Merge sort e Quick sort) su degli array di dimensioni crescenti. I risultati mostrano che Bubble sort, Selection sort e Insertion sort diventano inefficaci su strutture dati di grandi dimensioni, confermando l'ipotesi formulata che prevedeva la loro lentezza su array estesi. Al contrario, Merge sort e Quick sort mantengono prestazioni efficienti anche con strutture dati di dimensioni maggiori, confermando l'ipotesi formulata riguardo alla loro efficienza su array di dimensioni maggiori.