ALGORITMI I

Relazione sugli algoritmi di ordinamento

**Descrizione del progetto**

Il progetto vuole dimostrare come il tempo di esecuzione di vari algoritmi di ordinamento si evolve con l’aumentare della mole di numeri da ordinare, indicati generalmente con la lettera ‘N’.

Il programma nasce da un problema affrontato in laboratorio, dove si chiedeva di consigliare a Pinocchio la scelta di investimento più saggia.

Scrivendo e mettendo in paragone le due equazioni che regolano la crescita degli zecchini sia nel caso di investimento nella Tree Bank (1 + 5k) sia in quello del Buoni del Tesoro ((1,20) ^k),dove k=numero di anni, è facile rendersi conto che con il passare del tempo l’investimento in Buoni del Tesoro porta ad un guadagno maggiore, poiché un’equazione esponenziale cresce più velocemente di una lineare.

Tuttavia, è possibile notare che se l’investimento durasse meno di 25/26 anni, converrebbe la Tree Bank.

Sulla base di questa idea è nato il seguente progetto, dove vengono messi a confronto tre algoritmi di ordinamento:

* **Bubble Sort**: con complessità quadratica Θ(N²);
* **QuickSort**: con complessità dell’ordine di nlogn Θ(nlogn);
* **HeapSort**: con complessità dell’ordine di nlogn Θ(nlogn);

**Scelte progettuali**

E’ stato deciso di riunire le funzioni utili al programma in un file a parte, sfruttando la compilazione separata. In **utils.c** è possibile trovare le funzioni di ordinamento e di gestione degli array.

Si è poi creato un file **utils.h** contente tutti i prototipi delle funzioni inserite nel file precedentemente menzionato, ed infine si è poi realizzato il programma vero e proprio (**ordinamenti.c**).

Nel codice del programma sono state definite due costanti: *numeroRun* e *ntest*.

La prima indica quante volte ripetere una run, una per ogni algoritmo di ordinamento, con lo stesso input, mentre la seconda costante identifica il numero di test da effettuare, pari alla dimensione dell’array *dimensioni*, dove ogni cella contiene il numero di elementi da ordinare volta per volta.

L’array da ordinare viene creato in maniera casuale e sempre diversa, e ciò è garantito dalla funzione *srand* presente nel main che prende come parametro l’orario del sistema in continua evoluzione.

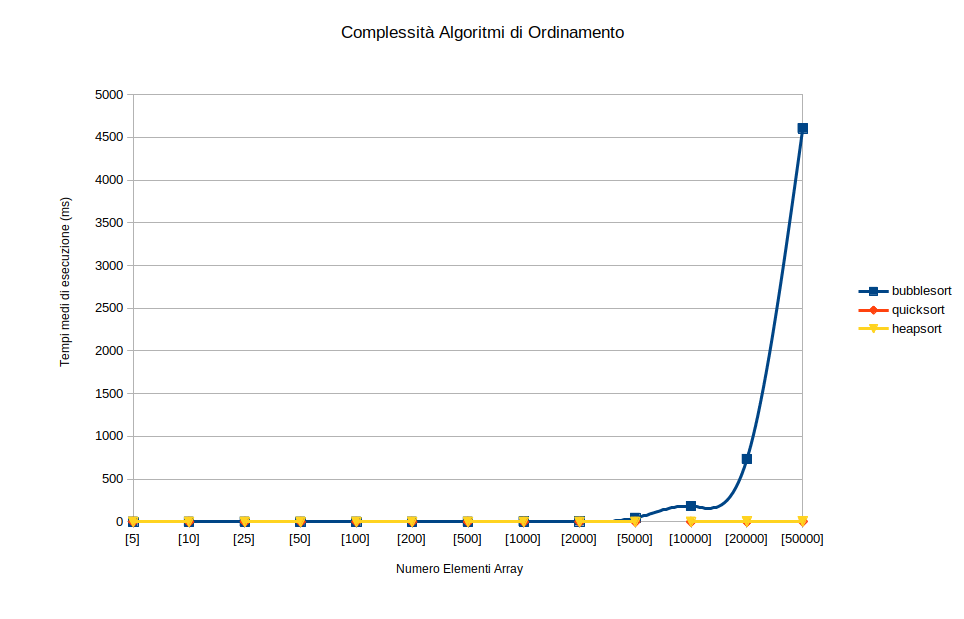
Si è deciso di incrementare la costante *ntest* e di aggiungere nuovi valori all’interno dell’array, in modo tale da evidenziare maggiormente l’efficienza ed inefficienza dei vari algoritmi di ordinamento.

Per il calcolo del tempo di esecuzione di ogni singolo algoritmo durante ogni singola run, salvato nella variabile *tempodiesecuzione*, è stata usata la funzione *gettimeofday* della libreria *<sys/time.h>* , in appoggio all’uso delle struct *timeval* ***T1*** e ***T2***.

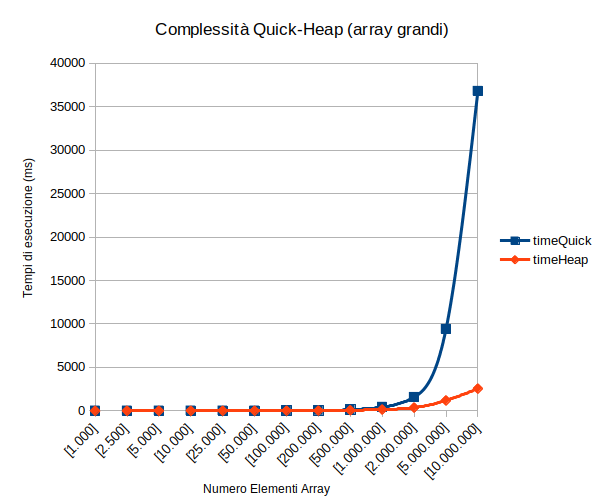
Al costo di una maggiore precisione, sono state implementate decisioni differenti nella descrizione ed uso dei grafici, descritti nella prossima sezione.

Il programma, infine, oltre a stampare i vari risultati ottenuti su console, salva i valori output di ogni singolo test in un file con estensione .csv contenente i risultati già divisi in tabelle.

Il nome del file viene preso come parametro di input da riga di comando.

**Grafici****

Considerando che il programma mostra i millisecondi come interi, non è possibile riportare con precisione il comportamento al di sotto di array con dimensione 1000(tutti restituiscono 0), tuttavia da questo punto in poi si può notare una differenza sostanziale: il tempo di esecuzione del Bubble Sort cresce esponenzialmente rispetto a quello del QuickSort e dell’HeapSort,i quali fino a N uguale a 50.000,restituiscono valori differenti di 1-2 millisecondi. Il comportamento di questi due ultimi algoritmi, con l’aumentare di N, è stato descritto nel grafico seguente.



Non riuscendo a descrivere in modo preciso l’andamento di questi due algoritmi con array di piccole dimensioni, si è deciso di testarli con array voluminosi.

I seguenti valori di N non sono stati aggiunti al programma, poiché ciò causerebbe tanto tempo prima della sua terminazione (soprattutto a causa del bubble che in questi test avrebbe prodotto risultati di attesa enormi), tuttavia è possibile aggiungerli all’array e modificare il valore della costante *ntest*.

**Considerazioni finali**

Possiamo quindi concludere che in linea generale l’HeapSort si è dimostrato l’algoritmo migliore ed il più efficiente, di conseguenza sarebbe anche quello che sceglieremmo in linea generale. Nel caso di N relativamente piccoli, una ipotetica scelta potrebbe ricadere sul Bubble Sort, poiché senz’altro più facile da realizzare rispetto ai suoi “avversari”. I due algoritmi aventi la stessa complessità asintotica, hanno dimostrato di crescere in maniera molto simile, tuttavia, come mostrato dall’ultimo grafico, l’HeapSort risulta essere più efficiente.

*Matteo Magrì* **20025172**

*Gabriele Ferrari* **20025045**