Merico Michele, Montesi Dennis, Turcan Boris

PROGETTO LABORATORIO ASD
RELAZIONE ESERCIZIO 1: BINARY INSERTION SORT E QUICKSORT

BINARY INSERTION SORT

Il binary insertion sort è una versione migliore dell'insertion sort in cui si utilizza la ricerca dicotomica per determinare la posizione corretta in cui inserire gli elementi da ordinare. Anche se questa versione dell'algoritmo riduce il numero di confronti a $O(N \bullet lgN)$, la sua complessità è $O(n^2)$, cioè come quella dell'insertion sort.

QUICKSORT

Il quicksort è un algoritmo di ordinamento sul posto basato sul paradigma divide et impera. In particolare, i passaggi per un ordinare un array A[p..r] sono:

1. Divide: partizionare l'array A[p..r] in due sottoarray A[p..q-1] e A[q..r] (eventualmente vuoti) tali che, per ogni elemento dell'array, risulta che A[p..q-1] <= A[q] <= A[q..r]. L'indice q viene calcolato mediante una funzione partition();
2. Impera: ordinare i due sottoarray A[p..q-1] e A[q+1..r] chiamando ricorsivamente il quicksort;
3. Combina: poiché i due sottoarray sono ordinati sul posto non occorre alcun lavoro per combinarli, l'array A[p..r] è ordinato.

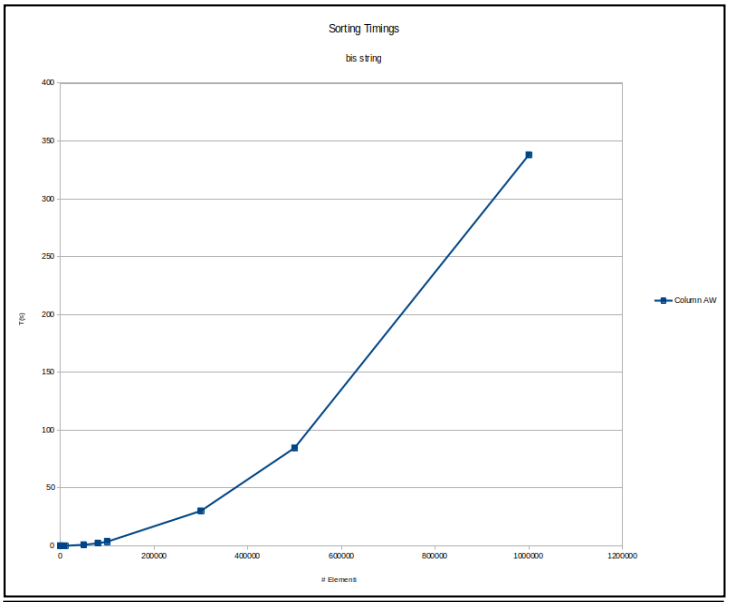
RISULTATI SPERIMENTALI

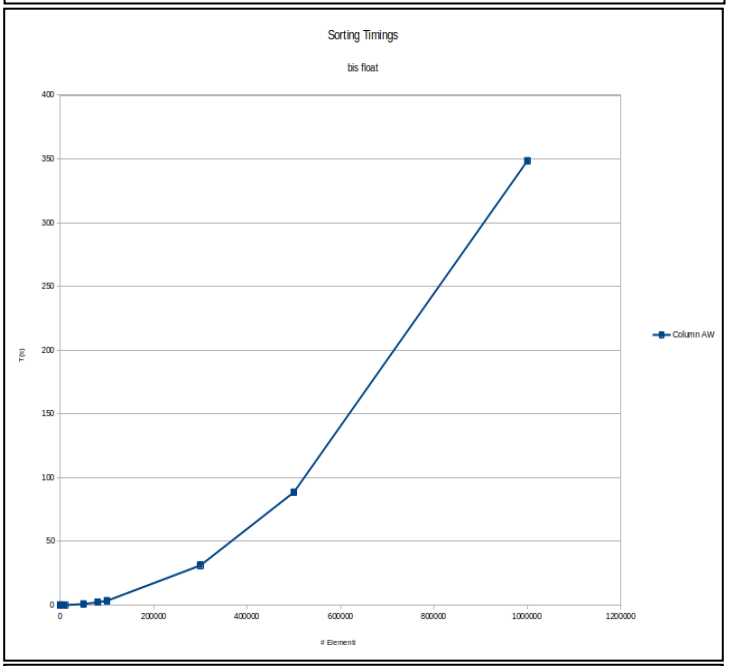
INPUT(integer)	TEMPO(s)				
	qsort(m)	qsirt(l)	qsort(r)	qsort(rand)	bis
1	0.000001	0.000001	0.000001	0.000001	0.000001
5	0.000001	0.000001	0.000001	0.000002	0.000001
10	0.000002	0.000002	0.000002	0.000002	0.000001
100	0.000016	0.000014	0.000015	0.000015	0.000014
1000	0.000194	0.000195	0.000179	0.000272	0.000602
10000	0.002389	0.002569	0.002341	0.003503	0.04411
50000	0.014953	0.015256	0.01483	0.02933	0.837973
80000	0.026015	0.027082	0.036095	0.03711	2.110918
100000	0.034364	0.034957	0.047967	0.04705	3.31349
300000	0.129528	0.136666	0.130834	0.174586	29.194715
500000	0.223309	0.254441	0.242589	0.418543	81.186401
1000000	0.543765	0.589116	0.548164	0.70123	326.91098

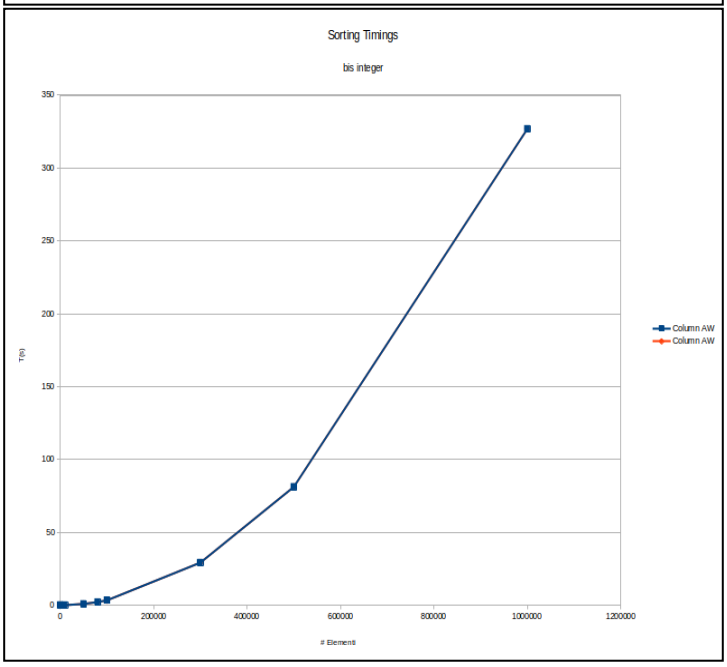
INPUT(string)	TEMPO(s)				
	qsort(m)	qsirt(l)	qsort(r)	qsort(rand)	bis
1	0.000001	0.000001	0.000001	0.000001	0.000001
5	0.000001	0.000001	0.000001	0.000001	0.000001
10	0.000002	0.000002	0.000002	0.000002	0.000001
100	0.000022	0.000018	0.000015	0.000012	0.000014
1000	0.000291	0.000327	0.000295	0.000156	0.000501
10000	0.003918	0.003911	0.003772	0.002078	0.037419
50000	0.023499	0.02398	0.0294	0.014409	0.930451
80000	0.040339	0.042059	0.040016	0.024283	2.176348
100000	0.052032	0.053817	0.053442	0.032245	3.615926
300000	0.181655	0.192386	0.181353	0.143183	30.231388
500000	0.318081	0.363198	0.320836	0.258353	84.70784
1000000	0.718089	0.830765	0.720255	0.528288	338.059814

INPUT(float)	TEMPO(s)				
	qsort(m)	qsirt(l)	qsort(r)	qsort(rand)	bis
1	0.000001	0.000001	0.000001	0.000002	0.000001
5	0.000001	0.000001	0.000001	0.000002	0.000001
10	0.000002	0.000002	0.000002	0.000013	0.000001
100	0.000022	0.000018	0.000015	0.000014	0.000014
1000	0.000204	0.000248	0.00019	0.000169	0.000447
10000	0.002566	0.002469	0.002517	0.003035	0.036547
50000	0.016101	0.016469	0.01539	0.015009	0.857285
80000	0.028387	0.027955	0.026318	0.033975	2.185798
100000	0.035708	0.037824	0.034789	0.038108	3.414952
300000	0.130774	0.141786	0.132877	0.13498	31.161753
500000	0.239143	0.270839	0.243468	0.250323	88.541183
1000000	0.529178	0.648904	0.573849	0.571434	348.542328

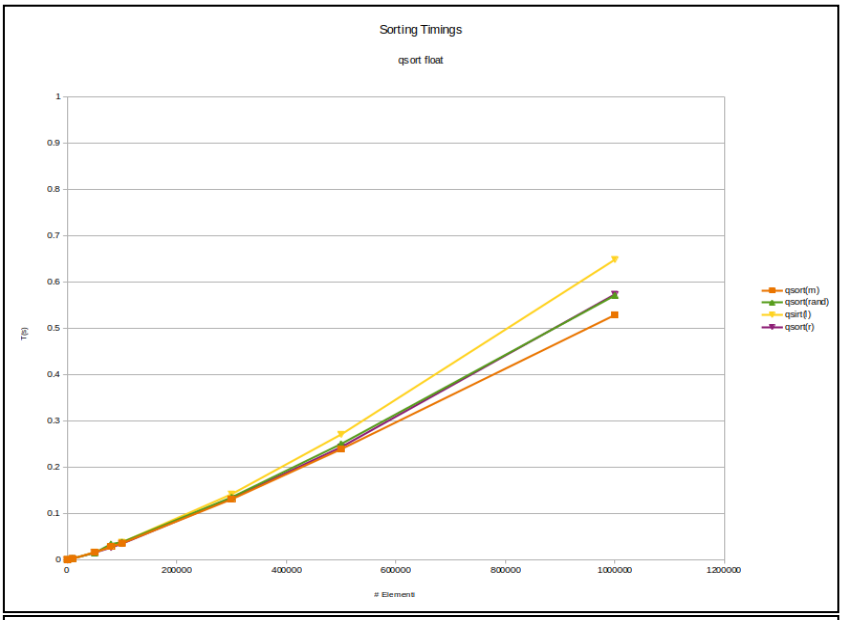
GRAFICI BINARY INSERTION SORT

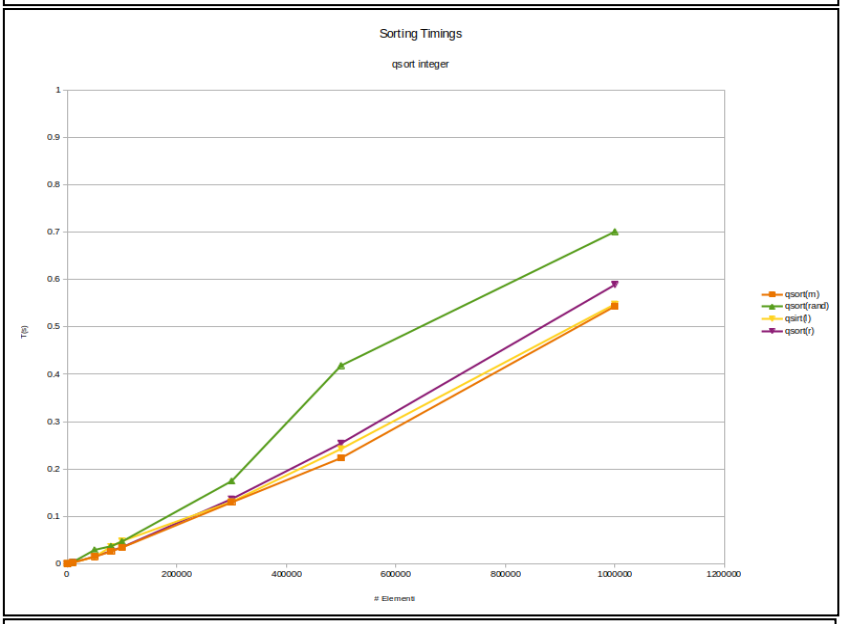


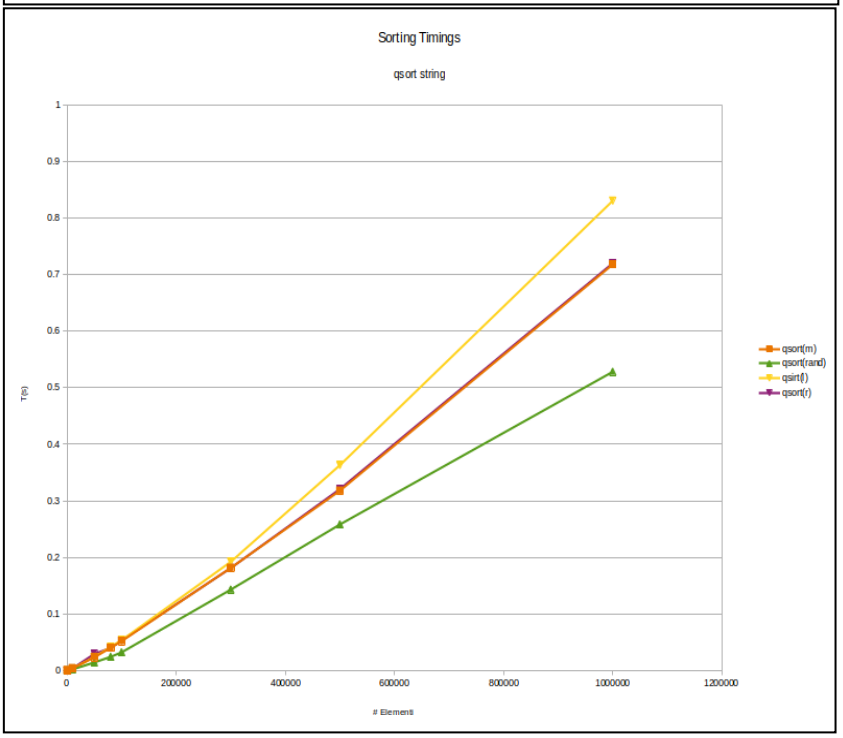




GRAFICI QUICKSORT







CONCLUSIONI

Binary Insertion Sort

Come si può notare dai grafici, l'algoritmo è particolarmente efficiente per ordinare una piccola quantità di elementi. Al crescere della dimensione dell'input la complessità dell'algoritmo tende a $O(n^2)$, ovvero non si nota unasignificativa differenza con il comportamento dell'insertion sort.

Quicksort

Il tempo di esecuzione del quicksort dipende dal fatto che il partizionamento sia bilanciato o sbilanciato, e questodipende a sua volta da quale elemento viene scelto come perno per il partizionamento.
Nella nostra sperimentazione abbiamo raccolto i dati sui tempi di esecuzione utilizzando come perno dell'array l'elemento più a sinistra, l'elemento centrale e l'ultimo elemento. Infine abbiamo testato una versione del quicksortche utilizza una partition randomizzata.
In particolare, abbiamo ottenuto i seguenti risultati:

- Scegliendo come perno il primo o l'ultimo elemento dell'array, il comportamento dell'algoritmo nel casomedio tende a $O(n^2)$: questo è dovuto al fatto che nella maggior parte dei casi le partizioni create da partition() non sono particolarmente bilanciate, rallentando quindi l'ordinamento;
- Scegliendo come perno l'elemento centrale dell'array, l'algoritmo nel caso medio si comporta come un algoritmo ottimale per l'ordinamento avente complessità $O(N \bullet lgN)$: questo è legato a una partizione nella maggior parte dei casi ben bilanciata che rende l'algoritmo più veloce;
- La versione del quicksort che utilizza una partition randomizzata ha un buon comportamento nel caso medio, non risulta essere ottimale ma potrebbe essere conveniente utilizzarla quando la dimensione dell'input è molto estesa.