Relazione esercizio 1

Componenti del gruppo
Carlino Mattia (matricola: 913397)
Ferreri Federico (matricola: 929655)
Fontana Anna (matricola: 912716)

Implementazione Merge-BinaryInsertion Sort

Si misurino i tempi di risposta variando il valore di 'k' e si produca una breve relazione in cui si riportano i risultati ottenuti insieme a un loro commento. Dimostrare nella relazione come il valore di 'k' dovrebbe essere scelto nella pratica. Nel caso l'ordinamento si protragga per più di 10 minuti potete interrompere l'esecuzione e riportare un fallimento dell'operazione. I risultati sono quelli che vi sareste aspettati? Se sì, perché? Se no, fate delle ipotesi circa il motivo per cui l'algoritmo non funziona come vi aspettate, verificatele e riportate quanto scoperto nella relazione.

Osservazioni

Dalle osservazioni registrate variando il valore di 'k' della funzione è emerso che i tempi di esecuzione non cambiano molto fino ad una certa soglia, che è diversa in base al tipo di dato, ma che si può approssimare per tutti intorno al valore 'k' = 1000.

Quello che però effettivamente cambia è l'utilizzo della memoria che si viene a risparmiare se si preferisce l'implementazione del merge-binary-insertion-sort rispetto al classico merge-sort.
Il merge-sort "puro" ha infatti complessità spaziale pari a $O(n)$ dovuta alle due malloc presenti nella funzione di supporto 'merge'. L'utilizzo della chiamata al binary-insertion-sort evita quindi di ricorrere ad allocazioni superflue poiché la sua complessità spaziale è pari a O(1).

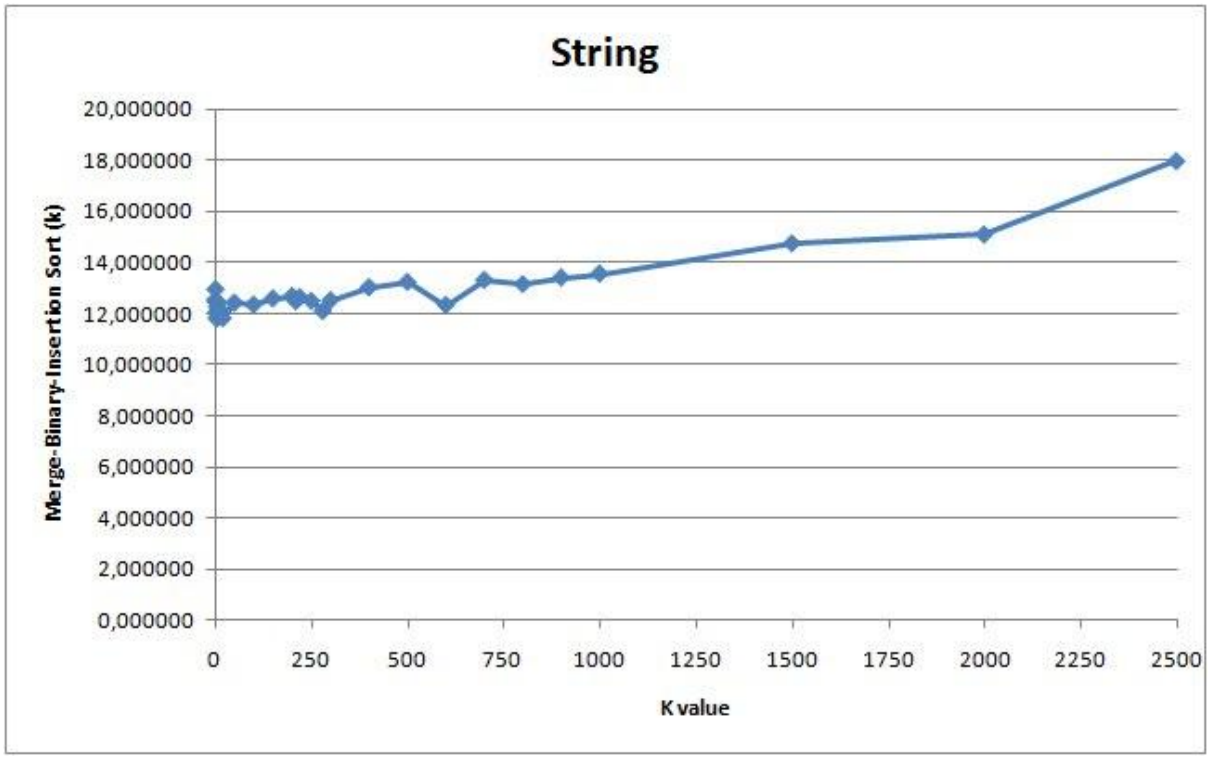
Inoltre, si può notare che usare il binary-insertion-sort riduce il numero di comparazioni rispetto all'insertion-sort, ma il numero di scambi che gli elementi devono effettuare non cambia: infatti, per spostare un elemento nella posizione corretta tutti gli elementi che si trovano tra la posizione individuata e l'elemento si devono spostare a destra di una posizione.

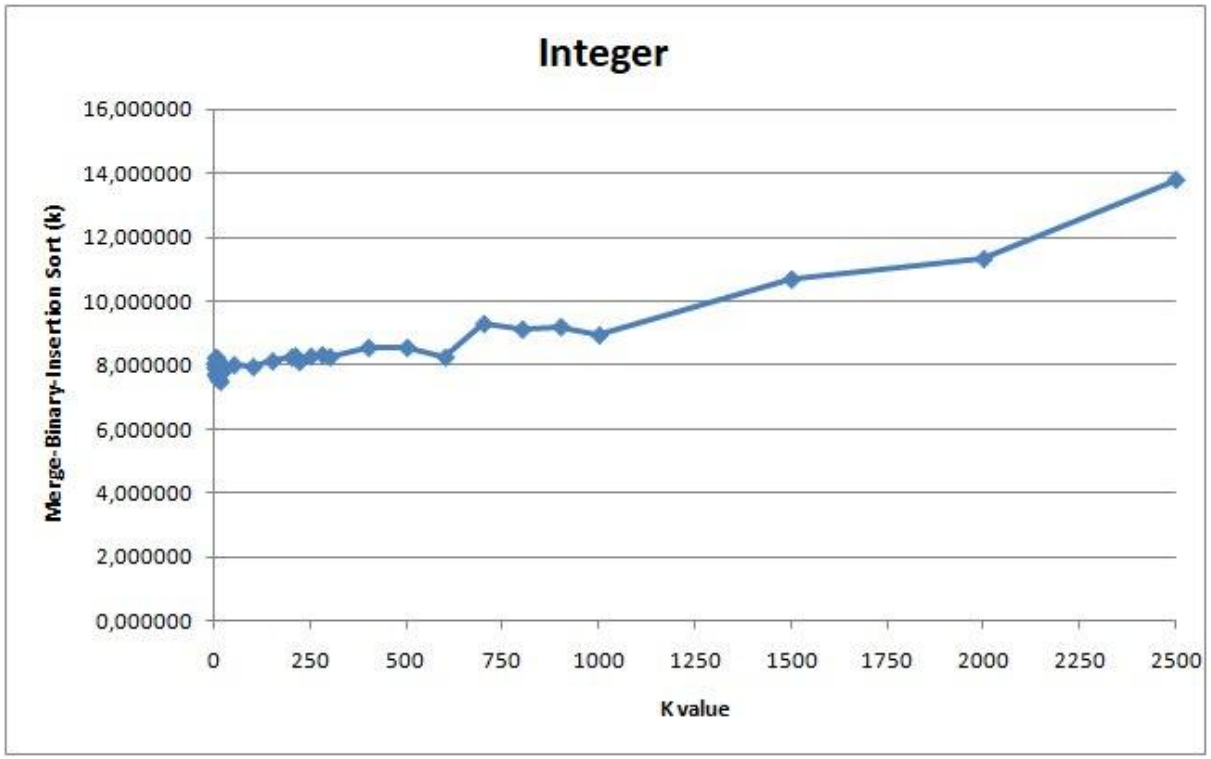
Dal punto di vista della complessità temporale il binary-insertion-sort non varia molto rispetto all'utilizzo del classico insertion-sort (che ha complessità nel caso peggiore $O(n^2)$): anche qui, nel caso peggiore, si avrà $O(n)$* ($O(n)$+$O(logn)$) = $O(n)$ * $O(n)$= $O(n^2)$ (con $O(logn$)complessità del binary_search).

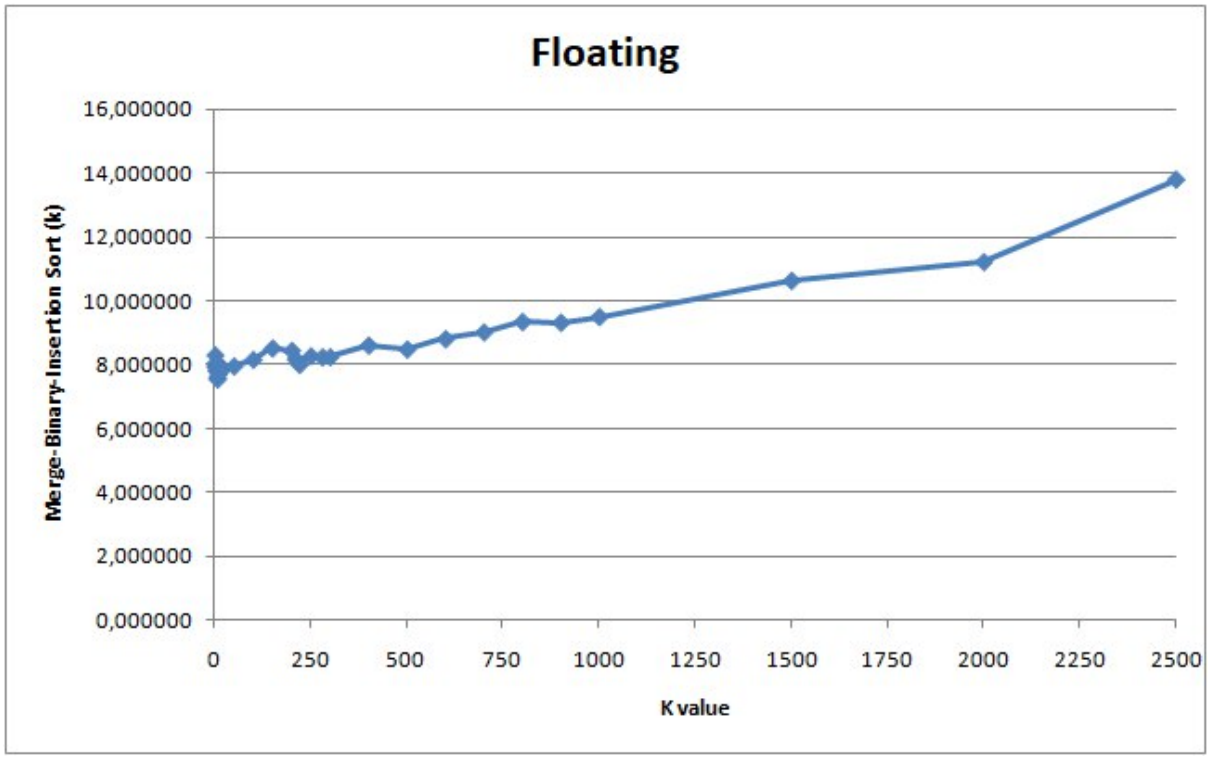
È comunque impossibile migliorare il tempo di ordinamento del merge-sort poiché questo algoritmo risulta essere già ottimo (ha complessità $O(n\ log\ n)$).

Si nota invece, come da noi atteso, un certo degrado nelle prestazioni quando il valore ‘k’ comincia ad aumentare considerevolmente, ad esempio, per k superiore a 1000. Nella pratica si può dunque ricorrere ad un ‘k’ < 1000 che non alteri troppo i tempi di esecuzione (come abbiamo visto, infatti, fino a 1000 i risultati sembrano essere identici) a favore di un uso ridotto della memoria utilizzata.

Di seguito riportiamo i grafici ottenuti al variare del parametro k:




Integer
16,000000
14,000000
12,000000
10,000000
8,000000
6,000000
4,000000
2,000000
0,000000
Merge-Binary-Insertion Sort (k)
0
250
500
750
1000
1250
1500
1750
2000
2250
2500
K value


Floating
16,000000
14,000000
12,000000
10,000000
8,000000
6,000000
4,000000
2,000000
0,000000
Merge-Binary-Insertion Sort (k)
0
250
500
750
1000
1250
1500
1750
2000
2250
2500
K value