Relazione algoritmo Merge-BinaryInsertion-Sort

1 Introduzione

La presente relazione descrive un'analisi dettagliata delle prestazioni dell'algoritmo di ordinamento "Merge-BinaryInsertion-Sort" al variare del valore di "k", che rappresenta il numero di elementi su cui verrà applicato l'algoritmo di ordinamento "BinaryInsertion-Sort" dopo la fase di scomposizione dell'array tramite l'algoritmo "Merge". Durante l'analisi, verranno esaminati tre diversi campi che possono essere utilizzati come chiave di ordinamento.

L'obiettivo di questa relazione è valutare il comportamento dell'algoritmo di ordinamento in base al valore di "k" e alle diverse chiavi di ordinamento utilizzate. Saranno misurati i tempi di risposta per ciascuna combinazione di "k" e campo chiave, al fine di determinare come questi fattori influenzino le prestazioni complessive dell'algoritmo.

2 Background

2.1 Merge-Sort: Il Merge-Sort è un algoritmo di ordinamento efficiente basato sulla tecnica "Divide and Conquer". L'algoritmo suddivide ricorsivamente l'array da ordinare in sotto-array più piccoli, fino a quando non si raggiunge una dimensione minima (solitamente un singolo elemento). Successivamente, combina i sotto-array ordinati per ottenere un array ordinato completo.

Dal punto di vista delle prestazioni, il Merge-Sort ha un tempo di esecuzione nel caso peggiore e nel caso medio di O(n log n). Ciò lo rende un algoritmo efficiente per gestire grandi insiemi di dati.

2.2 BinaryInsertion-Sort: Il BinaryInsertion-Sort è una variante dell'algoritmo di ordinamento Insertion-Sort che utilizza una ricerca binaria per trovare la posizione corretta di inserimento di ciascun elemento nell'array ordinato. Questa ricerca binaria riduce il numero di confronti necessari per posizionare un elemento nell'array ordinato, passando da O(n) a O(log n), migliorando le prestazioni dell'algoritmo.

Dal punto di vista delle prestazioni, il BinaryInsertion-Sort ha un tempo di esecuzione nel caso peggiore di O(n^2), dove "n" rappresenta la dimensione dell'array da ordinare.

3 Metodologia

Per valutare le prestazioni dell'algoritmo Merge-BinaryInsertion-Sort al variare del valore di "k" e dei diversi campi chiave di ordinamento, sono stati condotti una serie di test utilizzando tre tipi di dati differenti. I test sono stati eseguiti su tre diverse dimensioni di dataset, al fine di coprire una gamma di scenari di utilizzo realistici.

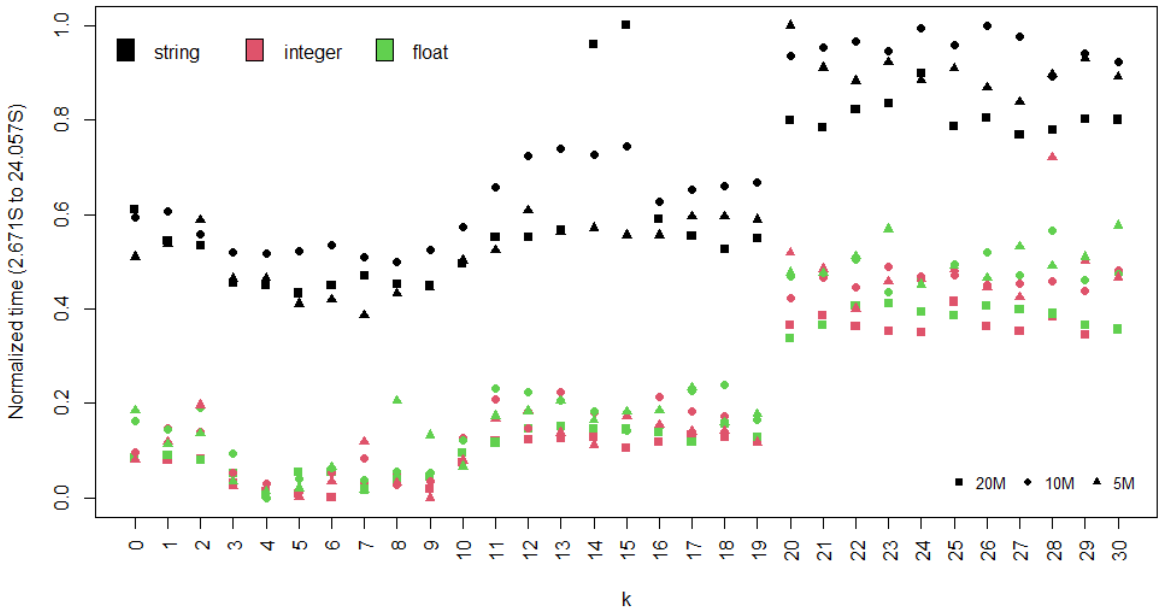
3.1. Dataset: Sono stati generati tre dataset di dimensioni diverse: piccolo, medio e grande. Il dataset piccolo è composto da 5M di elementi, il dataset medio da 10M di elementi e il dataset grande da 20M di elementi. I dati all'interno di ciascun dataset sono stati presi dalla stessa fonte (il file condiviso dal docente, records.csv).

3.2. Valori di "k": L'algoritmo Merge-BinaryInsertion-Sort è stato eseguito su ciascun dataset utilizzando valori di "k" compresi tra 0 e 30 (inclusi). Per ogni valore di "k", l'algoritmo è stato eseguito e sono stati misurati i tempi di risposta con la libreria time.h e la funzione clock().

3.3 Campi chiave di ordinamento: sono stati eseguiti tutti i precedenti test su tutti i campi chiave di ordinamento, interi, floating point e stringhe.
Qui sotto viene mostrato una possibile composizione di un record nei dati:
9, ferro, 2163766, 59111.526206

4 Risultati

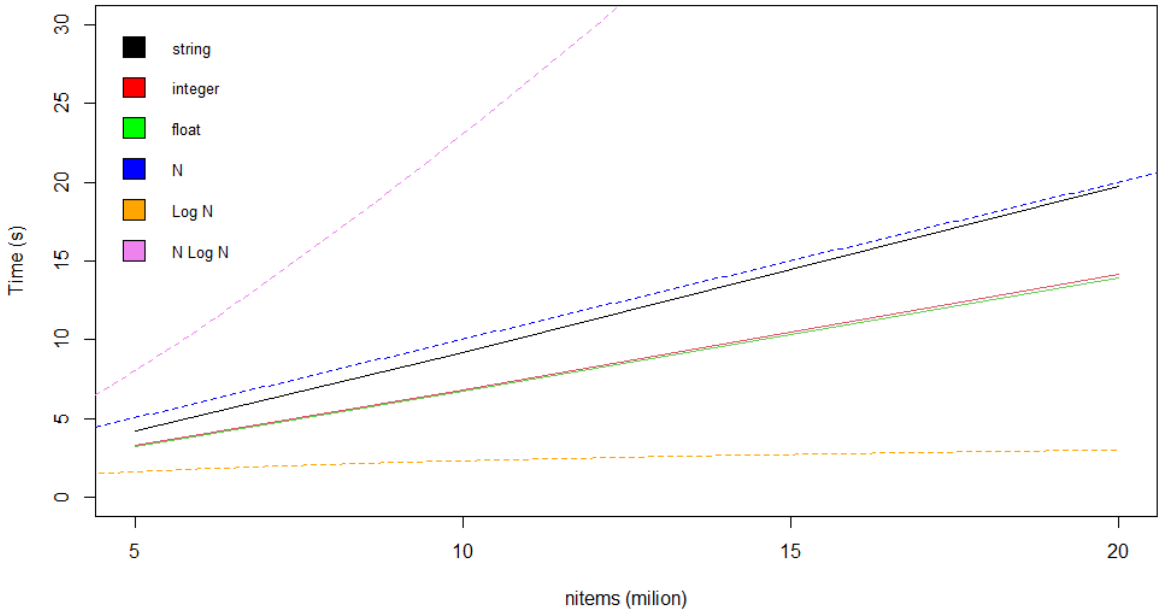
I risultati dell'analisi dei tempi di esecuzione dell'algoritmo Merge-BinaryInsertion-Sort al variare del valore di "k" e dei diversi campi chiave di ordinamento sono stati rappresentati tramite un grafico xy.
Sull'asse delle y, è stato utilizzato il tempo di esecuzione normalizzato (valori normalizzati: da 2.671S a 24.057S, essi sono stati normalizzati basandosi sui campionamenti divisi per quantità di elementi da ordinare), mentre sull'asse delle x sono stati riportati i valori di "k" da 0 a 30 (inclusi).
I dati che hanno generato questo grafico si possono trovare nel file report_tests.csv.



4.1 Tipi di dati: I dati raccolti mostrano che gli interi e i floating point seguono lo stesso andamento nel tempo di esecuzione. Ciò suggerisce che l'ordinamento di questi due tipi di dati con valori numerici non presenta differenze significative in termini di complessità computazionale. Tuttavia, è interessante notare che le stringhe mostrano un andamento simile, ma con tempi di esecuzione complessivamente maggiori rispetto agli altri tipi di dati. In altre parole, l'ordinamento di stringhe richiede un tempo maggiore rispetto all'ordinamento di valori numerici.

4.2 Valori di "k": è interessante notare che per valori di "k" superiori a 20, si osserva un'impennata significativa nel tempo impiegato da parte di tutti e tre i tipi di dati. Questo potrebbe indicare che valori di "k" troppo elevati possono portare a un aumento significativo della complessità computazionale dell'algoritmo, indipendentemente dal tipo di campo chiave utilizzato.

4.3 Quantità di dati: Osservando il grafico sotto, si può notare che il valore tempo di esecuzione rimane sostanzialmente lineare al variare della quantità di dati da ordinare. Questo sembrerebbe indicare che l'effetto della dimensione del dataset sull'efficienza dell'algoritmo è proporzionale e si situa intorno a N.
Questo però non è vero in quanto l’algoritmo dovrebbe seguire una curva N Log N, quindi possiamo presupporre che al momento abbiamo troppi pochi dati per trarre una conclusione.



Infine, dai risultati emersi, si può dedurre che i migliori risultati in termini di tempo di esecuzione normalizzato si ottengono per valori di "k" compresi tra 4 e 9 (inclusi) per tutti e tre i tipi di dati considerati. Questo intervallo di valori di "k" sembra fornire un buon equilibrio tra l'efficienza dell'algoritmo e la complessità computazionale.

5 Conclusione

L'analisi dei dati conferma in parte le aspettative pregresse riguardo alle prestazioni dell'algoritmo Merge-BinaryInsertion-Sort. Dai dati raccolti si evince un tempo linearmente dipendente dalla quantità di dati da ordinare, in linea con la complessità N. Tuttavia esso non coincide con le prestazioni del Merge-BinaryInsertion-Sort che dovrebbero essere nell'ordine di N Log N, però se si andassero a raccogliere più dati con diverse campionature si riuscirebbe probabilmente ad ottenere una curva più vicina al risultato teorico.

Come previsto, le stringhe hanno richiesto un tempo di esecuzione complessivamente maggiore rispetto agli interi e ai floating point. Questo è dovuto alla maggiore complessità delle stringhe e della comparazione tra di esse, rispetto alle operazioni matematiche tra i valori numerici.

Un'altra aspettativa confermata è il fatto che un valore di "k" troppo grande influisce negativamente sulle performance dell'algoritmo. Questo perché il BinaryInsertion-Sort, utilizzato dopo la prima fase del merge sort, ha una complessità temporale di O(N log N). Pertanto, all'aumentare di "k", il tempo di esecuzione dell'algoritmo aumenta in modo significativo, degradando rapidamente le performance complessive.

Sulla base dei risultati ottenuti, si può suggerire che, per ottenere le migliori performance dell'algoritmo, si dovrebbe selezionare un valore di "k" compreso tra 4 e 9 (inclusi). Questo intervallo sembra fornire un buon equilibrio tra efficienza e complessità computazionale, evitando sia un valore troppo piccolo che potrebbe ridurre l'efficienza dell'algoritmo, sia un valore troppo grande che potrebbe compromettere drasticamente le performance a causa del BinaryInsertion-Sort.