Laboratorio Algoritmi e Strutture

Dati *Relazione esercizio 1*

Descrizione dell'esercizio:

L'obiettivo dell'esercizio è implementare una libreria per l'ordinamento generico utilizzando gli algoritmi Merge Sort e Quick Sort.

Tramite Unit Testing si vuole creare dei test automatici per verificare il corretto funzionamento dei due algoritmi.

Si vuole ordinare il file "records.csv" contenente 20 milioni di record in base a tre campi:

1. field1 per i record di tipo stringa
2. field2 per i record di tipo intero
3. field3 per i record di tipo floating point

Durante l'ordinamento di questo file si devono misurare le prestazioni degli algoritmi e fornire una relazione con analisi numerica sui tempi di esecuzione per ogni campo.

Analisi dei risultati:

Qui in seguito vengono mostrate le foto e le rispettive analisi dei tempi di terminazione degli ordinamenti dei 20 milioni di record tramite Merge Sort e Quick Sort per ciascun tipo di campo.

Tempi e analisi Field1 - Char:

- Tempi Merge Sort:

```
dimu@dimu-ubuntu-boot:~/Scrivania/Progetto-ASD/ex1$ make runMain
time bin/main_ex1 src/records.csv src/sorted.csv 1 1
Reading records into memory...
Sorting 20000000 records...
Sorting completed in 39.211634 seconds.
Writing sorted records to the output file...
Writing completed in 6.464002 seconds.
Total time: 45.675636 seconds.
31.57user 14.44system 0:46.44elapsed 99%CPU (0avgtext+0avgdata 14404476maxreside
nt)k
1420976inputs+1416592outputs (0major+16193164minor)pagefaults 0swaps
```

- Tempi Quick Sort:

```
dimu@dimu-ubuntu-boot:~/Scrivania/Progetto-ASD/ex1$ make runMain
time bin/main_ex1 src/records.csv src/sorted.csv 1 2
Reading records into memory...
Sorting 20000000 records...
Sorting completed in 18.036999 seconds.
Writing sorted records to the output file...
Writing completed in 6.557575 seconds.
Total time: 24.594574 seconds.
22.50user 2.28system 0:24.84elapsed 99%CPU (0avgtext+0avgdata 7189120maxresident
)k
1420928inputs+1416592outputs (0major+1796954minor)pagefaults 0swaps
```

- Analisi:

  i tempi di ordinamento delle stringhe tramite algoritmo Quick Sort sono molto più brevi rispetto a quelli tramite Merge Sort. Inizialmente, facendo uso di una funzione di partizionamento basata unicamente sul confronto tra pivot e altro elemento in analisi, avevo notato che l'algoritmo Quick Sort non terminasse mai entro i 10 minuti (circa 14 minuti il "miglior" tempo), dunque dopo svariati tentativi e analisi dell'algoritmo ho deciso di adottare la funzione del partizionamento a tre vie, ossia una funzione che divide in tre parti la stringa in analisi: parte minore del pivot in uso, parte uguale al pivot in uso e parte maggiore del pivot in uso. Così facendo, la problematica del tempo di esecuzione è stata risolta. La causa di ciò è la presenza di tanti elementi stringa di grandi dimensioni uguali all'interno del file da 20 milioni di record, i quali non permettevano di effettuare l'ordinamento in tempi inferiori ai 10 minuti.

Tempi e analisi Field2 - Int:

- Tempi Merge Sort:

```
dimu@dimu-ubuntu-boot:~/Scrivania/Progetto-ASD/ex1$ make runMain
time bin/main_ex1 src/records.csv src/sorted.csv 2 1
Reading records into memory...
Sorting 20000000 records...
Sorting completed in 39.082635 seconds.
Writing sorted records to the output file...
Writing completed in 6.619746 seconds.
Total time: 45.702381 seconds.
31.41user 14.64system 0:46.38elapsed 99%CPU (0avgtext+0avgdata 14404680maxreside
nt)k
1420960inputs+1416592outputs (0major+16193163minor)pagefaults 0swaps
```

- Tempi Quick Sort:

```
dimu@dimu-ubuntu-boot:~/Scrivania/Progetto-ASD/ex1$ make runMain
time bin/main_ex1 src/records.csv src/sorted.csv 2 2
Reading records into memory...
Sorting 20000000 records...
Sorting completed in 28.374972 seconds.
Writing sorted records to the output file...
Writing completed in 6.590423 seconds.
Total time: 34.965395 seconds.
32.86user 2.31system 0:35.19elapsed 99%CPU (0avgtext+0avgdata 7189120maxresident
)k
1420928inputs+1416592outputs (0major+1796955minor)pagefaults 0swaps
```

- Analisi:

  i tempi di ordinamento dei valori interi sono simili per entrambi gli algoritmi, nonostante il Quick Sort prevalga sul Merge Sort di una decina di secondi. Con il campo dei record di tipo intero non ho mai riscontrato problemi riguardanti la terminazione di entrambi gli ordinamenti in tempi brevi.

Tempi e analisi Field3 - Double (Floating Point):

- Tempi Merge Sort:

```
dimu@dimu-ubuntu-boot:~/Scrivania/Progetto-ASD/ex1$ make runMain
time bin/main_ex1 src/records.csv src/sorted.csv 3 1
Reading records into memory...
Sorting 20000000 records...
Sorting completed in 39.537202 seconds.
Writing sorted records to the output file...
Writing completed in 6.429787 seconds.
Total time: 45.966989 seconds.
31.25user 14.93system 0:46.52elapsed 99%CPU (0avgtext+0avgdata 14404412maxreside
nt)k
107768inputs+1416592outputs (1691major+16221431minor)pagefaults 0swaps
```

- Tempi Quick Sort:

```
dimu@dimu-ubuntu-boot:~/Scrivania/Progetto-ASD/ex1$ make runMain
time bin/main_ex1 src/records.csv src/sorted.csv 3 2
Reading records into memory...
Sorting 20000000 records...
Sorting completed in 29.597879 seconds.
Writing sorted records to the output file...
Writing completed in 6.448587 seconds.
Total time: 36.046466 seconds.
33.97user 2.28system 0:36.26elapsed 99%CPU (0avgtext+0avgdata 7189120maxresident
)k
1420936inputs+1416592outputs (0major+1796953minor)pagefaults 0swaps
```

- Analisi:

  l'analisi è la medesima dei valori interi, unica nota differente è che il tempo terminazione di entrambi gli algoritmi per i record di tipo floating point è leggermente più lento rispetto a quello per i record di tipo intero. Anche in questo caso non ho riscontrato problemi per la terminazione degli ordinamenti.

Foto di dimostrazione del corretto funzionamento dei test automatici:

```
src/mergesort_quicksort_test.c:155:mergesort_char_test_zero_elements:PASS
src/mergesort_quicksort_test.c:156:mergesort_char_test_one_element:PASS
src/mergesort_quicksort_test.c:157:mergesort_char_test_four_elements:PASS
src/mergesort_quicksort_test.c:160:mergesort_int_test_zero_elements:PASS
src/mergesort_quicksort_test.c:161:mergesort_int_test_one_element:PASS
src/mergesort_quicksort_test.c:162:mergesort_int_test_four_elements:PASS
src/mergesort_quicksort_test.c:165:mergesort_fp_test_zero_elements:PASS
src/mergesort_quicksort_test.c:166:mergesort_fp_test_one_element:PASS
src/mergesort_quicksort_test.c:167:mergesort_fp_test_four_elements:PASS
src/mergesort_quicksort_test.c:172:quicksort_char_test_zero_elements:PASS
src/mergesort_quicksort_test.c:173:quicksort_char_test_one_element:PASS
src/mergesort_quicksort_test.c:174:quicksort_char_test_four_elements:PASS
src/mergesort_quicksort_test.c:177:quicksort_int_test_zero_elements:PASS
src/mergesort_quicksort_test.c:178:quicksort_int_test_one_element:PASS
src/mergesort_quicksort_test.c:179:quicksort_int_test_four_elements:PASS
src/mergesort_quicksort_test.c:182:quicksort_fp_test_zero_elements:PASS
src/mergesort_quicksort_test.c:183:quicksort_fp_test_one_element:PASS
src/mergesort_quicksort_test.c:184:quicksort_fp_test_four_elements:PASS

-----------------------
18 Tests 0 Failures 0 Ignored
OK
0.00user 0.00system 0:00.00elapsed 74%CPU (0avgtext+0avgdata 1280maxresident)k
0inputs+0outputs (0major+69minor)pagefaults 0swaps
```

In conclusione, l'algoritmo di Quick Sort è in grado di ordinare i 20 milioni di

record in tempi più efficienti rispetto all'algoritmo di Merge Sort. Inoltre, l'algoritmo di Quick Sort fa un uso di gran lunga più efficiente delle risorse del sistema, visto che esegue operazioni nel kernel in tempi di media 7 volte inferiori rispetto all'algoritmo di Merge Sort (si possono leggere i dati nelle foto di analisi dove compare un determinato tempo seguito dalla parola "system").