Laboratorio Di Algoritmi e Strutture Dati Anno 2021/2022

Stefano Peddoni Matr 839138

Davide Laguardia Matr 822358

Relazione Esercizio 1

Nell'Esercizio 1 è stata implementata una libreria generica con alcune funzioni:

```c
/* Create new Generic Array */
GenericArray* GenericArray_create(GenericArrayCmp);

/* Returns:
 * - 1 if the array is empty
 * - 0 if the array is not empty
 */
int GenericArray_empty(GenericArray* generic_array);

/* Returns the capacity of the Generic Array */
int GenericArray_capacity(GenericArray* generic_array);

/* Returns the number of elements currently stored in the Generic Array */
unsigned long GenericArray_size(GenericArray* generic_array);

/* Deallocates the Generic Array */
void GenericArray_free(GenericArray* generic_array);

/* It accepts as input a pointer to an Generic Array and an integer "i"
   and it returns the pointer to the i-th element of the Generic Array */
void* GenericArray_get(GenericArray* generic_array, int i);

/* Insert an element inside the Generic Array */
void GenericArray_insert(GenericArray* generic_array, void* element);

/* Method that doubles the size of the Generic Array */
void GenericArray_check_and_realloc(GenericArray* generic_array);

/* Reverses the order of elements in the Generic Array */
void GenericArray_reverse(GenericArray* generic_array);
```

Dopo aver creato la libreria per il generic_array, e testato le funzioni create con alcuni test; abbiamo creato l'applicativo per poter eseguire i due algoritmi richiesti: Insertion Sort (utilizzando la ricerca binaria) e Quick Sort, ordinando i dati che sono presenti all'interno del file records in maniera crescente.

I due algoritmi presi in questione sono caratterizzati in questa maniera:

ALGORITMO	CASO PEGGIORE	CASO MIGLIORE
Quick Sort	O(n^2)	O(nlogn)
InsertionSort	O(n^2)	O(n)

Di seguito abbiamo calcolato per ogni field (int, string e float) i tempi di ordinamento per ciascun algoritmo. Di seguito alcuni esempi di dati che ho ottenuto:

Nell'applicativo sono stati implementati i due algoritmi che abbiamo citato sopra nella tabella. Abbiamo effettuato diverse verifiche con file di dimensioni diverse ed abbiamo ottenuto questi risultati:

RECORDS (STRING)	QUICK SORT	INSERTION SORT
100.000	0.30s	11.34s
500.000	1.69s	104.92s
700.000	2.46s	233.90s
20.0000	87.32s	∞

RECORDS (INT)	QUICK SORT	INSERTION SORT
100.000	0.28s	19.12s
500.000	1.51s	109.47s
700.000	2.19s	220.26s
20.0000	78.55s	∞

RECORDS (FLOAT)	QUICK SORT	INSERTION SORT
100.000	0.30s	17.12s
500.000	1.52s	142.20s
700.000	2.17s	225.60
20.0000	78.77s	∞

Insertion Sort

Possiamo quindi notare che i file contenenti sempre più records, richiedono un tempo maggiore, e nel caso dei 20.000.000 di records il tempo per l'ordinamento risulta essere infinito. Questo dipende anche dalla macchina con cui si lavora, se si utilizza una macchina virtuale i tempi saranno maggiori.

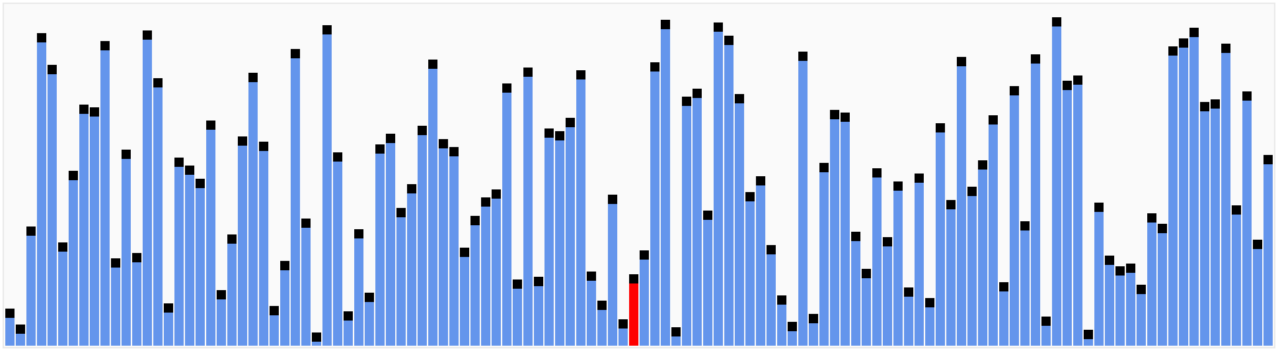
Quick Sort

Possiamo notare che questo algoritmo riesce ad ordinare in maniera crescente i records fino ai 20.000.000. Il field "string" ci mette più tempo rispetto a "int" e "float", ma, nonostante ciò, termina l'ordinamento in maniera positiva.
All'interno dell'algoritmo del Quick Sort viene utilizzato il pivot per la scelta dell'elemento.

Scelta del Pivot

La scelta del pivot nel Quick Sort può avvenire in diversi casi:

- Primo elemento: in questo caso viene preso il valore più basso (low), quindi il primo elemento della lista.
- Ultimo elemento: in questo caso viene preso il valore più alto (high), quindi l'ultimo elemento della lista.
- Elemento centrale: in questo caso è la soluzione migliore, poiché con records elevati riesce ad essere più performante a livello di tempistica.



Di seguito troviamo un grafico che rappresenta i tempi con il quale l'algoritmo riesce ad ordinare un numero di elementi. Possiamo subito notare quello che è accaduto a noi durante lo svolgimento dell'esercizio; l'algoritmo Insertion Sort ci mette molto più tempo rispetto al Quick Sort.