RELAZIONE ESERCIZIO1

PRIMO USO

Per ordinare un file di 20 milioni di interi abbiamo implementato InsertionSort e MergeSort ottenendo i seguenti risultati :
- InsertionSort impiega un tempo > 10 minuti e riporta un fallimento dell'esecuzione;
- MergeSort impiega un tempo compreso tra i 30 e i 40 secondi;

La grande differenza di prestazione dei due algoritmi è dovuta alla loro complessità.

InsertionSort è un algoritmo intuitivo che risulta efficiente per ordinare un piccolo numero di elementi; inoltre può impiegare quantità diverse di tempo per ordinare sequenze della stessa dimensione, a seconda di quanto queste siano già parzialmente ordinate.
La sua complessità è pari a O(n^2), quindi con un file di input di 20 milioni di interi il tempo di calcolo aumenta in modo esponenziale, rendendo inefficiente l'algoritmo.

MergeSort invece è un algoritmo ricorsivo basato sula tecnica del Divide et Impera, ovvero suddivide il problema in due problemi dello stesso tipo ma di dimensione inferiore.
E' molto più efficiente di InsertionSort perché la sua complessità è pari a O(nlogn) sia nel caso migliore che in quello peggiore.
Pertanto il suo tempo di esecuzione rimane costante e ci permette di ordinare una sequenza di 20 milioni di interi in tempi molto più brevi rispetto ad InsertionSort.

SECONDO USO

Per il secondo uso abbiamo implementato la funzione "searchInt".
Essa prende in input due array :
- searchArray, che contiene gli elementi da sommare
- sum, in cui sono presenti le somme da ricercare.

Il metodo richiama la funzione mergeSort per ordinare searchArray.
Successivamente tramite un ciclo while, searchInt scorre l'array dai suoi estremi muovendosi verso il centro, salvando gli elementi della parte sinistra (indice "left") nella variabile "leftElem" e quelli della parte destra (indice "right") nella variabile "rightElem".
Infine "searchInt" somma gli elementi salvati in leftElem con gli elementi salvati in rightElem e confronta ogni risultato della somma con i numeri presenti nell'array "sum".
Se due elementi contenuti in "searchArray" sommati danno come risultato un elemento contenuto in "sum", la variabile booleana found verrà impostata con il valore "true", altrimenti manterrà il suo valore di default ("false") e il metodo continuerà a scorrere l'array.
Il valore della variabile found viene volta per volta memorizzato nell'ArrayList "foundNumbers" che il metodo stamperà una volta conclusa la sua esecuzione.

Per quanto riguarda le prestazioni, il metodo SearchInt ha complessità O(klogk) in quanto richiama la funzione MergeSort.
Ciò lo rende un algoritmo ottimo perché conserva la proprietà di MergeSOrt di rimanere costante indipendentemente dalla dimensione dell'input: SearchInt è quindi in grado di svolgere la sua funzione in maniera efficiente.