Esercizio 1: Implementazione della procedura Merge

Implementare una funzione

```
void Merge(int *A, int p, int q, int r)
```

i cui parametri sono un array di interi A, e tre interi $p < q < r$, tali che i due sottoarray $A[p, q-1]$ e $A[q, r-1]$ sono *ordinati in senso crescente*. La funzione genera un nuovo array temporaneo, la cui lunghezza è $r-p$, contenente tutti gli elementi di $A[p, q-1]$ e di $A[q, r-1]$ (con eventuali ripetizioni) in ordine crescente, il nuovo array viene quindi sovrascritto al sottoarray $A[p, r-1]$. Il tempo di esecuzione della funzione dev'essere in $O(r-p)$.

Esercizio 2: MergeSort

Incorporare la funzione `Merge` dell'esercizio precedente all'interno di un programma che ordini ricorsivamente una sequenza di interi utilizzando l'algoritmo *MergeSort*, visto a lezione. A questo scopo, si definisca la funzione ricorsiva

```
void MergeSort(int *A, int p, int q)
```

i cui argomenti sono un array di interi A, e due interi p e q con $q \geq p$, ed il cui effetto è quello di riordinare in modo crescente gli elementi $A[p], A[p+1], \ldots, A[q-1]$.

Esercizio 3: Intersezione in tempo lineare

Sfruttando l'algoritmo della funzione `Merge`, scrivere un programma che accetti in input due sequenze di interi *strettamente crescenti* e stampi gli elementi che occorrono in entrambe in *tempo lineare*, ossia $O(n+m)$ dove n ed m sono le cardinalità delle due sequenze in input. Al solito, si assuma che ogni sequenza sia data specificando la sua lunghezza nella prima lininea di input seguita da tutti i valori che la compongono, ciascuno su una linea distinta.

Esercizio 4 (per casa): Front-Compression

Questo esercizio collegato ad una nota tecnica di compressione, detta *Front-Coding* (o Front-Compression), comunemente impiegata per ridurre lo spazio occupato da dizionari che consistono di una lista di parole lessicograficamente ordinate. L'osservazione di base è che due parole consecutive nell'ordine del dizionario solitamente condividono un certo prefisso di caratteri iniziali. Una parola dunque può essere compressa rimpiazzando i caratteri di tale prefisso con un valore intero che ne esprime la lunghezza, che è eventualmente 0 se la parola e quella che la precede nell'ordine lessicografico iniziano con caratteri diversi. L'esempio della tabella sotto mostra una lista di parole lessicograficamente ordinate e le loro rispettive codifiche mediante Front-Coding :

Originale	**Front-Coded**
`aba`	**0**`aba`
`abba`	**2**`ba`
`bab`	**0**`bab`
`babba`	**3**`ba`
`baco`	**2**`co`

Ad esempio, nella seconda riga della tabella, la parola `abba` condivide con `aba` un prefisso di lunghezza 2 ed è dunque codificata dalla stringa `2ba`, nella quale il prefisso condiviso `ba` è sostituito dalla sua lunghezza rappresentata in decimale. Per convenzione, assumiamo che la prima stringa nell'ordine ha un prefisso condiviso di lunghezza 0.

Scrivete due programmi: un compressore e un decompressore. Il compressore legge da input una sequenza di stringhe in ordine lessicografico, dove ciascuna di esse si trova su una linea distinta, e per ogni stringa letta emette una linea in output contenente la sua codifica tramite `Front-Coding` terminata da $\backslash n$. Nella codifica di ogni stringa, si sostituisca al prefisso condiviso i caratteri della rappresentazione in base 10 della sua lunghezza (come nella tabella sopra).

Il decompressore ha il compito di ricostruire la sequenza di stringhe originale a partire dalle loro codifiche mediante Front-Coding. Quindi il decompressere legge in input la sequenza di stringhe prodotte dal compressore, e per ciascuna di esse stampa la stringa originale in una linea di output terminata da $\backslash n$.

Mediante la redirezione dell'intput si verifichi il funzionamento della coppia compressore/decompressore sul file `input.txt` (scaricabile qui), che consiste di una sequenza di 1000 stringhe lessicograficamente ordinate.

Si utilizzi da shell il comando:

$$(./\texttt{Compressore} < input.txt) > output$$

(dove `Compressore` è il file eseguibile del programma compressore) per generare un file di nome `output` contenente le codifiche di tutte le parole presenti nel file `input.txt`. Successivamente, utilizzando la stessa sintassi per la redirezione, si lanci il decompressore sul file `output` e si verifichi che il file decompresso sia identico all'originale.

Ipotesi:

- Le stringhe in input sono lunghe al piú 100 caratteri
- Ogni stringa puó avere come caratteri soltanto le lettere minuscole dell'alfabeto inglese $a - z$