Esercizio

Siano $S_1 = S_1[0]S_1[1]\ldots S_1[n_1-1]$ e $S_2 = S_2[0]S_2[1]\ldots S_2[n_2-1]$ due sequenze ordinate di chiavi intere. In ogni sequenza le chiavi sono distinte, ma una stessa chiave può comparire sia in S_1 che in S_2. Progettare una modifica dell'algoritmo `Merge` che restituisca il numero di chiavi in $S_1 \cap S_2$. L'algoritmo deve usare spazio aggiuntivo costante oltre alle due sequenze di input, e avere complessità $O(n_1 + n_2)$.

IDEA: adattare Merge in modo che quando incontra le stesse chiave in S_1 e S_2 incrementa un contatore C senza salvare la chiave in sequenza di uscita. È facile vedere che per ogni $k \in S_1 \cap S_2$ esiste un istante nella esecuzione

a meno, in cui $S_1[i] = S_2[j] = k$

Algoritmo CountIntersection (S_1, S_2)

input: S_1 e S_2 sequenze ordinate di interi distinti (in ciascuna sequenza)

output: $|S_1 \cap S_2|$

```
i ← 0; j ← 0; C ← 0;
while ((i < |S1|) AND (j < |S2|)) do {
  if (S1[i] = S2[j]) then {i++; j++; C++}
  else if (S1[i] < S2[i]) then i++ else j++
```

}
return C

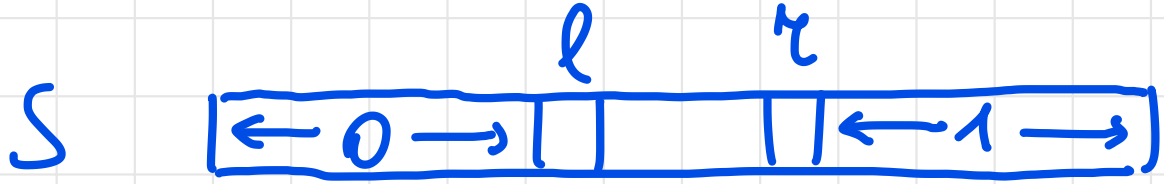
Complessità: $\Theta(n_1+n_2)$

Stesse regole di Regex

Esercizio

Progettare un algoritmo che ordini una sequenza di n bit $S = S[0]S[1]\ldots S[n-1]$ in tempo $\Theta(n)$. L'algoritmo deve essere in-place e utilizzare un solo ciclo.

IDEA: Adattare Partition con un unico ciclo while e 2 variabili l e r inizializzate con' l = 0 r = n-1 mantenendo il seguente invariante



Nella generica iterazione facio le seguenti operazioni in base a S[l] e S[r]

4 POSSIBILI CONFIGURAZIONI

S[l]	S[r]	Operazioni da fare
0	0	l++
0	1	l++, r--
1	0	swap(S[l], S[r])
1	1	r--

Finisco quando l = r+1

Completare (precedenza e anelli) per esercizio

Esercizio C-13.40 [GTG14]

Siano $S_1, S_2, \ldots, S_k$ k sequenze non vuote di entry con chiavi nel range $[0, N-1]$, per un qualche $N \geq 2$. Descrivere un algoritmo che ordini separatamente tutte le sequenze in tempo $O(n+N)$, dove n è la somma delle taglie delle sequenze. In output le sequenze devono rimanere distinte.

Esercizio

Sia S una sequenza di n elementi distinti sui quali è definito un ordine totale. Si ricordi che una *inversione* in S è una coppia di elementi $S[i], S[j]$ tali che $j < i$ ma $S[j] > S[i]$. Descrivere un algoritmo che determina il numero di inversioni in S in tempo $O(n \log n)$.
(**Suggerimento:** adattare MergeSort.)

IDEA: Eseguire MergeSort con le seguenti modifiche

* MergeSort(S) restituisce anche
  - la sequenza ordinata
  - il # di inversioni in S
* Il numero m di inversioni in S è calcolato così

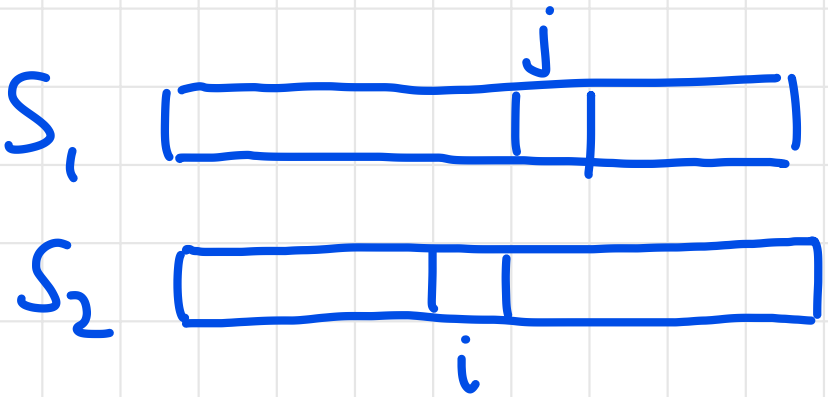
$n = n_1 + n_2 + n_{12}$ dove

n_1 = # inversioni nella prima metà di S
(calcolato ricorsivamente

n_2 = # inversioni nella seconda metà di S
(calcolato ricorsivamente

n_{12} = # inversioni a cavallo tra prima
e seconda metà di S calcolato
nella fase di merge

Merge:



Quando nel Merge sto guardando $S_1[j]$ e $S_2[i]$ e sapo che $S_1[j] < S_2[i]$ a quel punto memorizzo $S_1[j]$ nella sequenza ordinata finale e so che esso precede ESATTAMENTE i numeri con $S_2[0], S_2[1], \ldots S_2[i-1]$ e poi quindi incrementerò M_{12} di i

Completam per esercitio

Esercizio C-13.48 [GTG14]

Sia dato un insieme A di n viti e un insieme B di n dadi. Ogni vite va bene per un solo dado. Progettare e analizzare un algoritmo per trovare tutte le coppie (vite,dado) giuste avendo a disposizione una primitiva che data una vite a e un dado b decide in tempo costante se a è *piccola*, *giusta*, o *grande* per b.