# 4 L'importanza di essere ordinati: ideologia di Merge & Ricerca Binaria

## 4.1 Vettori simili (II esonero, 21/11/22)

Diciamo che due vettori $u$ e $v$ di lunghezza rispettivamente $m$ ed $n$ sono *simili* se contengono gli stessi elementi, eventualmente con un numero di occorrenze diverse. Ad esempio, sono simili i vettori di interi {1,2,1,1,4,1} e {1,2,4,2} (**attenzione** che la proprietà è simmetrica).

Scrivere lo pseudo-codice di un algoritmo che verifica se due vettori sono simili in ciascuno dei seguenti casi:

1. per due vettori qualsiasi, senza modificarli e senza allocare memoria proporzionale ad $m$ e/o $n$;

2. per due vettori ascendenti. Discutere la complessità e dire se, potendo modificare i vettori, al punto **1** sarebbe conveniente prima ordinarli e poi applicare questo algoritmo invece che usare quello che non li modifica;

3. per due vettori contenenti valori in un intervallo noto $[min, max]$ di interi sufficientemente "piccolo" ($max - min \in \Theta(m + n)$).

## 4.2 Numero di Inversioni♣ (II esonero, 22/4/22)

Il numero di *inversioni* di un vettore $v$ è il numero delle coppie di indici $i < j \in dom(v)$ tali che $v[i] > v[j]$ (assumere per semplicità che il vettore contenga tutti valori distinti).

1. dire quale sia il numero minimo e massimo di inversioni in un vettore $v$ di lunghezza $n$;

2. alla luce del punto precedente, stabilire la complessità minima di qualsiasi algoritmo che conta le inversioni una ad una e scrivere un tale algoritmo;

3. ★progettare un algoritmo *divide-et-impera* che conta il numero delle inversioni in tempo pessimo $\Theta(n \log n)$. [SUGG: modificare opportunamente l'algoritmo *mergeSort* modificando *merge* in modo che calcoli un opportuno valore...]

## 4.3 Punti coincidenti (esame 27/1/22)

Dati due vettori di interi distinti $u$ e $v$ di lunghezza rispettivamente $m$ ed $n$ chiamiamo *punto coincidente* tra $u$ e $v$ una coppia di indici $i, j$ ($1 \leq i \leq m, 1 \leq j \leq n$) tali che $u[i] = v[j]$.

1. Scrivere lo pseudocodice di una funzione *quantiCoincidenti(u, m, v, n)* che calcola il numero di punti coincidenti tra $u$ e $v$ e fornire l'analisi del costo computazionale (questa funzione *non può modificare* i vettori in ingresso).

2. Scrivere lo pseudocodice di una funzione *quantiCoincidentiOrd(u, m, v, n)* che calcola il numero di punti coincidenti tra $u$ e $v$ sotto la precondizione $\mathbf{Cr}(u)$ & $\mathbf{Cr}(u)$, cioè $u$ e $v$ siano *strettamente* crescenti e fornire l'analisi del costo computazionale.

3. Generalizzare al caso in cui $u$ e $v$ siano ascendenti, cioè possano esserci elementi ripetuti. Osservare che, in questo caso, ci possono essere fino a $m \cdot n$ punti coincidenti, quindi contandoli uno a uno si ottiene necessariamente una funzione quadratica. Trovare il modo di scrivere una funzione di complessità $\Theta(m + n)$.

4. Avendo due vettori non ordinati e potendoli modificare, dal punto di vista della complessità computazionale, è conveniente usare la funzione *quantiCoincidenti* oppure prima ordinare i vettori e poi usare la funzione *quantiCoincidentiOrd*? Motivare la risposta (per semplificare i conti, potete assumere $m = n$).

### 4.4 Conta somme di valore $s$ (I esonero, 12/4/19)

Dato un vettore $v$ di interi si vuole trovare il numero di somme di coppie di elementi distinti che restituiscono un certo valore $s$. Formalmente, vogliamo conoscere la cardinalità dell'insieme $\{(i, j) \mid i < j \in dom(v), v[i] + v[j] = s\}$.

Esempio Se il vettore fosse $v = \{1, 7, 4, 5, 2, 3\}$ ed $m = 8$, ci sono 2 coppie che danno come somma 8, cioè 1+7 e 5+3. Osservate che non va considerato il fatto che 4+4=8 (non si tratta di una somma di elementi distinti).

1. Scrivere una funzione `n = countSums(v, s)` che torna il numero delle coppie *distinte* di $v$ di somma $s$.

2. Sotto la precondizione che il vettore $v$ sia strettamente crescente, scrivere una funzione lineare in $n$ [Sugg: per $0 \leq i < j < n$, cosa posso dedurre da $v[i]+v[j] < s$, da $v[i] + v[j] > s$ o da $v[i] + v[j] = s$?].

3. Se al punto precedente è stata data una funzione iterativa, scrivere una funzione ricorsiva, o viceversa.

4. Argomentare la correttezza delle funzioni di complessità lineare (anche senza una vera e propria dimostrazione), possibilmente aiutandosi con asserzioni logiche (precondizioni, invarianti, etc.).

### 4.5 Successione di Ulam (homework 1, 2014)

Consideriamo la *successione di Ulam*, definita come segue: $u_0 = 1, u_1 = 2$ e $u_n$ ($n > 1$) è il *minimo* numero naturale che si può scrivere in *modo unico* come somma di due precedenti numeri della successione. La successione sopra definita comincia con $1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, \ldots$.

Per capire meglio la definizione, osservate che $u_4$ non può essere 5, in quanto $5 = 2 + 3 = u_1 + u_2$, ma anche $5 = 1 + 4 = u_0 + u_3$. Viceversa solo $u_1 + u_3 = 2 + 4$ danno come somma 6.

Dare un algoritmo che calcola l'$n$–esimo numero di Ulam $u_n$.

SUGG: si può ottenere un algoritmo di complessità $\Theta(n^2)$ che usa memoria $\Theta(n)$ allocando un vettore di dimensione $n$ in cui memorizzare i numeri della successione via via trovati, e usare la funzione soluzione del problema 4.4 per calcolare il nuovo numero (tale funzione può anche essere ottimizzata per lo specifico uso di determinare se un certo candidato è effettivamente un numero di Ulam).

### 4.6 Elemento doppio (esame 22/4/22)

1. Scrivere un algoritmo che, dato un vettore $v$ di interi, determina (se esistono) una coppia di indici $i$ e $j$ tali che $v[i] = 2 * v[j]$;

2. risolvere lo stesso problema nel caso in cui il vettore $v$ sia ordinato in senso non decrescente;

3. valutare la complessità degli algoritmi forniti nei due casi e valutare se sia conveniente o meno prima ordinare l'array $v$ per risolvere questo problema usando la tecnica del punto 2.

### 4.7 Punto Fisso (esame 1/9/22)

In un vettore $v$ di interi lungo $n$, un *punto fisso* è un indice $i \in [0, n)$ tale che $v[i] = i$.

1. Scrivere una funzione `f, i = fixedPoint(v)` che carica TRUE in `f` se esiste un punto fisso nel vettore $v$ e FALSE altrimenti. Nel caso in cui carica TRUE in `f`, carica anche nella variabile `i` l'indice di un punto fisso.

2. Sotto la precondizione che il vettore $v$ sia ordinato in modo strettamente crescente (gli interi sono quindi tutti distinti), determinare le precondizioni per cui *possa* esistere un punto fisso in un segmento $v[inf, sup]$ di $v$.

3. Usare le condizioni del punto precedente per scrivere una funzione `f, i = fixedPointSorted(v)` che assumento $v$ strettamente crescente, risolva il problema del punto fisso in tempo $\mathcal{O}(\log n)$. Argomentare brevemente la correttezza.

### 4.8 Elementi di un vettore in $[m, M]$

1. Dare un algoritmo che, dato un vettore di numeri $v$ lungo $n$ e due numeri $m \leq M$, calcola quanti elementi di $v$ siano compresi nell'intervallo $[m, M]$. Formalmente, l'algoritmo deve calcolare $\#\{j \in [0, n) \mid m \leq v[j] \leq M\}$.

2. Assumendo che $v$ sia ordinato crescente, dare un algoritmo che migliora la complessità asitontica di quello dato al punto precedente.

3. Assumendo che $v$ contenga interi in un intervallo noto $[min, max]$ sufficientemente piccolo, cioè $max - min \in \Theta(n)$, progettare una struttura dati in modo da poter rispondere alla domanda $\#\{j \in [0, n) \mid m \leq v[j] \leq M\}$ in tempo $\Theta(1)$. Dare un algoritmo per costruire tale struttura dati.

### 4.9 Distanza tra due vettori

Definiamo *distanza tra due vettori* $u$ e $v$ il valore:

$$\min\{|u[i] - v[j]| \mid i \in dom(u), j \in dom(v)\}.$$

Dare un algoritmo per calcolare la distanza tra vettori, in ciascuno dei seguenti casi:

1. per *due vettori qualsiasi*, senza modificarli e senza allocare memoria proporzionale ad $m$ e/o $n$;

2. per *due vettori ascendenti*. Discutere la complessità e dire se, potendo modificare i vettori, al punto 1. sarebbe conveniente prima ordinarli e poi applicare questo algoritmo invece che usare quello che non li modifica;

3. per due vettori contenenti valori in un intervallo noto $[a, b]$ di interi sufficientemente "piccolo" $(b - a \in \Theta(m + n))$.