# Appunti lezione – Capitolo 2
# Analisi degli algoritmi

Alberto Montresor

14 Settembre, 2019

## 1 Domanda: Moltiplicare numeri complessi

1. **Quanto costa l'algoritmo "banale" dettato dalla definizione?**
   4.02
2. **Potete fare di meglio?**
   Soluzione di Gauss, datata 1805.
   Input: $a, b, c, d$, Output: $A1 = ac - bd, A2 = ad + bc$

$$\begin{aligned} m1 &= ac \\ m2 &= bd \\ A1 &= m1 - m2 = ac - bd \\ m3 &= (a+b)(c+d) = ac + ad + bc + bd \\ A2 &= m3 - m1 - m2 = ad + bc \end{aligned}$$

   In questo caso, il costo è 3.05 (3 moltiplicazioni e 5 addizioni).
3. **Qual è il ruolo del modello di calcolo?**
   Detto $m$ il costo di una moltiplicazione e $a$ il costo di una disequazione, si ottiene

$$3m + 5a \leq 4m + 2a$$

   disequazione che è vera solo quando $m \geq 3a$.

   Si tenga conto che al tempo, il termine "Computer" significava "Colui che calcola" ed era associato agli esseri umani (fin dal 1640); il termine ha iniziato ad essere associato alle "macchine calcolatrici" nel 1897, per poi venire associato a macchine programmabili solo nel 1945.

   Questo giustifica lo sforzo di Gauss per ridurre il numero di calcoli. Tenete conto che esistevano macchine calcolatrici per l'addizione a partire dal 1645 (la pascalina di Pascal) e per la moltiplicazione a partire dal 1672 (la macchina di Leibniz), ma la prima produzione di massa avvenne solo a partire dal 1820.

## 2 Domanda: Sommare numeri binari

Dimostrazione per assurdo. Supponiamo di avere un algoritmo "misterioso" che non esamina tutti i bit. Si esegua l'algoritmo su una coppia di numeri $x$ e $y$. Deve esistere qualche bit in posizione $i$ che non viene esaminato in uno dei due numeri. Supponiamo che l'algoritmo dia una risposta esatta (altrimenti non sarebbe corretto). Cambiamo il valore del bit nella posizione $i$ e diamo all'algoritmo la nuova coppia di numeri. Risponde con lo stesso valore di prima, perché quel bit non viene esaminato, dando quindi una risposta sbagliata perché la somma è cambiata. Assurdo.
In altre parole, qualunque algoritmo che effettui una somma deve esaminare tutti i bit, e quindi non può sommare in modo sublineare. Il meccanismo di somma della scuola elementare è ottimale.

## 3 Domanda: Dimostrazione della correttezza di Insertion Sort

Quando il ciclo è un ciclo **for**, il punto in cui verifichiamo l'invariante di ciclo è appena dopo la modifica della variable (per inizializzazione o incremento), e appena prima della verifica. Questa è un regola generale, ma non è "scritta nella pietra".
L'invariante di ciclo è la seguente: *all'inizio del ciclo $i$-esimo, il sottovettore $A[1 \ldots i-1]$ è ordinato*.

- **Inizializzazione**: Inizialmente, $j = 2$. Il sottoarray è dato da $A[1 \ldots 1]$, che è naturalmente ordinato.
- **Conservazione**: una dimostrazione informale; una dimostrazione formale richiederebbe la definizione di un invariante di ciclo per il while.
  - Sappiamo che il subarray $A[1 \ldots j-1]$ è ordinato.
  - Il ciclo while sposta tutti gli elementi maggiori di $A[j]$ fino a una posizione $k$ (inclusa) di una posizione verso destra, preservando l'ordine ma duplicando il valore in posizione $k$.
  - Tutti i valori precedenti $k$ sono minori di $A[j]$ e ordinati.
  - Tutti i valori dopo $k$ sono maggiori di $A[j]$ e ordinati.
  - Inserendo $A[j]$ in posizione $k$ dimostriamo l'invariante.
- **Terminazione**: Alla fine del ciclo, quando $j = n + 1$, l'invariante dice che l'intero array $A[1 \ldots j - 1] = A[1 \ldots n]$ è ordinato.

## 4 Domanda: Complessità di Insertion Sort

La complessità è data dal numero di confronti che devono essere effettuati per ogni elemento $i$, con $i \in [2, n]$. Chiamiamo questo numero di confronti $t_i$. Il numero di confronti varia a seconda della disposizione iniziale degli elementi. Il numero totale di confronti è pari a

$$\sum_{i=2}^{n} t_i$$

- **Caso ottimo**: L'array è ordinato, quindi $t_i = 1$ per ogni $i$. Questo perché $A[j-1] = A[i] \leq A[j], \forall j$, quindi il ciclo **while** esce subito. La complessità è quindi $O(n)$.
- **Caso pessimo**: L'array è ordinato nell'ordine inverso, tutti gli elementi in $A[1 \ldots i - i]$ sono superiori a $A[i]$, quindi vengono tutti spostati. Quindi il numero di confronti è pari a $i$. Otteniamo:

  $$\sum_{i=2}^{n} i - 1 = \left(\sum_{i=1}^{n-1} i\right) - 1 = \frac{n(n+1)}{2}$$

  Quindi, senza perderci nei dettagli dei conti, possiamo scrivere che il risultato finale sarà:

  $$T(n) = an^2 + bn + c$$

- **Caso medio**: In media, metà degli elementi in $A[1 \ldots i-1]$ sono più grandi di $A[i]$, metà sono più piccoli. Quindi, quello che si ottiene per i singoli elementi è:

  $$\sum_{i=2}^{n} i/2 = \frac{n(n+1)}{4} - 1$$

  Il costo finale sarà quindi simile al precedente.

  $$T(n) = an^2 + bn + c$$

## 5 Domanda: Complessità di merge$()$

Il costo di merge$(A, p, q, r)$ è dato da $n = r - p = O(n)$, perché tutti gli elementi di entrambi gli array devono essere visitati almeno una volta.

## 6 Domanda: Complessità di mergesort$()$

Svolgendo la ricorsione, otteniamo:

$$
\begin{aligned}
T(n) &= cn + 2T(n/2) \\
&= cn + 2cn/2 + 4T(n/4) \\
&= cn + 2cn/2 + 4cn/4 + 8T(n/8) \\
&= \ldots \\
&= cn + 2cn/2 + 4cn/4 + \ldots + 2^{k-1}n/2^{k-1} + 2^k n/2^k T(1) \\
&= cn + 2cn/2 + 4cn/4 + \ldots + 2^{k-1}cn/2^{k-1} + nc_0
\end{aligned}
$$

ovvero, ogni livello costa $cn$. Quanti livelli ci sono? Se $n = 2^k$, allora ci sono $k = \log n$ livelli. Quindi il costo finale sarà $cn \log n + c_0 n$.