16 - La complessità degli algoritmi

Sommario

- Le classi di complessità e la notazione “big theta”
- Analisi di complessità della ricerca lineare
- Analisi di complessità dell'Insertion Sort
- Perché la complessità è importante?

Complessità degli algoritmi

- Si misura come numero di istruzioni eseguite per ottenere il risultato. Questo numero dipende dalla dimensione dell'input, chiamata per convenzione ***n***.
  - es: numero di confronti necessari per trovare un elemento in una lista di lunghezza ***n***, o per ordinare una lista di ***n*** elementi
- Si distinguono ***analisi di caso pessimo***, ***medio*** e ***ottimo***.
- Notazione "**big theta**": usata per rappresentare classi di complessità. Esempi:
  - Ricerca lineare è $\Theta(n)$,
  - ***Insertion sort*** è in $\Theta(n^2)$,
  - Ricerca binaria è in $\Theta(\lg n)$,

Analisi dell'algoritmo di ricerca lineare

```
procedure RicLin (X, S)
  N ← lunghezza della sequenza S;
  k ← 1;     //indice del prossimo elemento di S
  while (k <= N) do  // termina quando k = N+1
     if ( X = Sk)      // se ho trovato X
     then  RETURN TRUE; //restituisco TRUE e esco
     else  k ← k+1; // passo al prossimo elemento
  RETURN FALSE; // se arrivo qui, X non c'è in S
```

- Nel caso ottimo, **X** è uguale al primo elemento della sequenza, quindi ho un solo confronto
- Nel caso medio faccio **X/2** confronti
- Nel caso pessimo, confronto **X** con tutti gli elementi della sequenza, quindi ho **N** confronti. **Si dice che RicLin ha complessità $\Theta(n)$**

L'algoritmo di *Insertion sort* in pseudocodice: versione finale

- Assumiamo che la **Lista** sia $S = s_1, s_2, ..., s_L$
- La lunghezza della lista è **L**

```
procedure Sort (Lista)
   N ← 2;
while (N <= L) do
(    pivot ← s_N;  // perno
     k ← N;   // indice della posizione vuota
     while  (k > 0 AND s_{k-1} > pivot) do
     (  s_k ← s_{k-1}; // copio nella posizione vuota
        k ← k-1; // sposto la posizione vuota
      )
      s_k ← pivot;
      N ← N+1;
)
```

Analisi di insertion sort nel caso pessimo

Confronti fatti per ogni pivot

Initial list	1st pivot	2nd pivot	3rd pivot	4th pivot	Sorted list
Elaine David Carol Barbara Alfred	1 Elaine David Carol Barbara Alfred	3 David 2 Elaine Carol Barbara Alfred	6 Carol 5 David 4 Elaine Barbara Alfred	10 Barbara 9 Carol 8 David 7 Elaine Alfred	Alfred Barbara Carol David Elaine

- Vengono considerati ***n-1*** pivot (in posizioni 2, 3, ..., ***n***)
- Il pivot di posizione ***k*** viene confrontato con ***k-1*** valori
- In tutto abbiamo ***1 + 2 + 3 + ... + n-1*** confronti, cioè ***n * (n-1)/2 = 1/2 * (n^2 - n)***
- Quindi **insertion sort** è nella categoria $\Theta(n^2)$.

Perché la complessità è importante?

- Anche se i computer sono sempre più veloci, possono non esserlo abbastanza per manipolare grandi quantità di dati.
- **Esempio**: Cerchiamo uno studente in una lista di 30.000 studenti.
  - Se ogni confronto richiede 10 msec, ci vogliono in media 150 secondi per trovarlo.
  - Se uso la Ricerca Binaria, lo trovo in al massimo $\mathbf{\lg_2(30000) = 15}$ confronti, quindi 0,15 secondi.
- Per ordinare una sequenza, ci sono algoritmi di complessità $\Theta(n \lg n)$ (invece di $\Theta(n^2)$ come Insertion Sort). **Esempio**: per una lista di 1000 elementi ho 10.000 confronti invece di 1.000.000.

Esercizio della volta scorsa

- Scrivere una procedura ricorsiva **`mult(x, y)`** che restituisce il prodotto di due numeri maggiori o uguali di zero, usando la seguente definizione ricorsiva di prodotto:

  `x * 0 = 0`

  `x * (y + 1) = x * y + x`

Una possibile soluzione:

```
procedure mult(x,y)
if (y = 0)
    then return 0;
    else return mult(x, y-1) + x;
```