Complessità computazionale

Teoria della complessità computazionale:

studia le risorse minime necessarie (principalmente

tempo di calcolo e memoria) per la risoluzione di un problema.

Per misurare l'efficienza di un algoritmo in maniera univoca, bisogna definire una metrica che sia indipendente dalle tecnologie utilizzate, altrimenti uno stesso algoritmo potrebbe avere efficienza diversa a seconda della tecnologia sulla quale viene eseguito.

Consideriamo il tempo di esecuzione proporzionale al numero di operazioni che vengono effettuate dall'algoritmo

I problemi vengono classificati in differenti classi di complessità, in base all'efficienza del migliore algoritmo noto che sia in grado di risolvere quello specifico problema

Complessità intrinseca di un problema.

Una particolare soluzione offre una limitazione dall'alto riguardo la complessità del problema.

Una distinzione informale:

- problemi *facili*, di cui si conoscono algoritmi di risoluzione efficienti

- problemi *difficili*, di cui gli unici algoritmi noti non sono efficienti

Esempi di dimensioni:

per un problema sugli interi il numero delle cifre dell'intero
(per un problema sugli interi il valore n dell'intero)
per un problema sugli array la lunghezza dell'array
per un problema sulle liste la lunghezza della lista
per un problema sugli alberi il numero di elementi dell'albero

Una misurazione dettagliata può essere difficilmente applicabile; per questo si introducono delle approssimazioni.

In particolare si ricorre alla notazione O(f(n)). Formalmente:

$g(n) \in O(f(n))$ se $\exists (n_0, c), c \geq 0, n_0 \geq 0$ tali che $\forall n > n_0, g(n) \leq cf(n)$

$3x^2+5x+6 \in O(x^2)$

Se prendo c = 4,

$3n^2+5n+6 < 4n^2$ se n è abbastanza grande

Esempi di ordini:

costante

log (n)

n (lineare)

n * log n

n^k   polinomiali

2^n

n!

Le prestazioni di un algoritmo possono dipendere, oltre che dalla dimensione del problema, anche da come si presentano i dati in input:

si considerano il *caso migliore*, il *caso peggiore* e il *caso medio*.

- Il caso migliore è il caso in cui i dati sono i migliori dati possibili per l'algoritmo, cioè quelli che richiedono meno elaborazioni per essere trattati.

- Il caso peggiore invece prevede i dati che richiedono il massimo numero di passi per l'algoritmo.

- Il caso medio è il caso più utile da analizzare perché fornisce un reale indicatore della complessità dell'algoritmo ma tendenzialmente è anche quello più complesso dato che spesso è difficile determinare quali sono i dati medi.
  A volte per risolvere il problema del caso medio si preferisce eseguire molte simulazioni dell'algoritmo e poi dai tempi ottenuti con le simulazione estrarre una formula che approssimi adeguatamente l'andamento medio.

Sono utili anche altre due misure, complementari della notazione *O grande*:

- $g(n) = \Omega(f(n))$ se $\exists (n_0, c)$ tali che $c \geq 0$, $n_0 \geq 0$, $\forall n > n_0 \, g(n) \geq cf(n)$, cioè $g(n)$ cresce non più lentamente di $f(n)$;

  dà una limitazione inferiore alla funzione g(n)

- $g(n) = \Theta(f(n))$ se $g(n) \in O(f(n))$ e $g(n) \in \Omega(f(n))$

dà una stima esatta della funzione g(n)

Scambio (bubblesort)

```
public void ordina(int vettore[]) {
        int temp;
        for (int i = 0; i < vettore.length - 1; i++) {
            for (int j= vettore.length-1 ; j > i; j--) {
                if (vettore[j] < vettore[j-1]) {
                    temp = vettore[j];
                    vettore[j] = vettore[j - 1];
                    vettore[j - 1] = temp;
}}}}
```

Ci sono due cicli for l'uno dentro l'altro.
Il ciclo for più esterno viene eseguito N-1 volte,
quello più interno N-I volte.

Contiamo ora il numero di operazioni. Per ogni iterazione del ciclo for più esterno vengono eseguiti N-I confronti quindi in totale ne vengono eseguiti:

$$(N-1)+(N-2)+...+1 = N(N-1)/2.$$

Per quanto riguarda gli assegnamenti, nel caso migliore gli assegnamenti che sono situati nel ciclo for più interno, non vengono mai eseguiti (perché A[J]<A[J-1] è sempre falso).

Nel caso peggiore invece, per ogni iterazione del ciclo for interno vengono eseguiti 3 assegnamenti, quindi in tutto ne vengono eseguiti

$$3(N-1)+3(N-2)+...+3 = 3N(N-1)/2$$

Riassumendo:

Condizioni iniziali	Numero confronti	Numero assegnamenti
Già ordinato	$\frac{N(N-1)}{2}$	0
Inv. ordinato	$\frac{N(N-1)}{2}$	$\frac{3N(N-1)}{2}$

Ricerca in vettore non ordinato (lunghezza N)

```
 public int ricercaConWhile(int vettore[], int elem) {
        int i = 0;
        boolean nonTrovato = true;
        while (nonTrovato && i < vettore.length) {
           if (elem == vettore[i]) {
               nonTrovato = false;
            }
           else {i++;}
        }
        if (nonTrovato) {return -1; }
        else{return i; }
    }
```

Analisi complessità media
Per ogni iterazione facciamo un numero costante di operazioni. Il numero di iterazioni dipende dalla posizione dell'elemento cercato. Calcoliamo la media di tutti i casi possibili:

1
2
3
…
N
—

N*(N+1)/2

Sono N casi

La media viene (N+1)/2