Algoritmi e Strutture Dati

Introduzione

Informazioni utili

- T.H. Cormen, C.E. Leiserson, R.L Rivest "Introduzione agli algoritmi". Jackson Libri
- Inizio dei laboratori: prima settimana di novembre.
- Iscrizione al laboratorio obbligatoria:
  http://www.docenti.unina.it/roberto.prevete
- Sito web con le slides dei corsi:
  http://people.na.infn.it/~bene/ASD/

Gli argomenti di oggi

- Analisi della bontà degli algoritmi
- Modello Computazionale
- Tempo di esecuzione degli algoritmi
- Notazione asintotica
- Analisi del Caso Migliore, Caso Peggiore e del Caso Medio

Cosa si analizza/valuta di un algoritmo

- Correttezza
  - Dimostrazione formale (matematica)
  - Ispezione informale
- Utilizzo delle risorse
  - Tempo di esecuzione
  - Utilizzo della memoria
  - Altre risorse: banda di comunicazione
- Semplicità
  - Facile da capire e da manutenere

Tempo di esecuzione

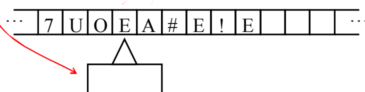
- Il *tempo di esecuzione* di un programma dipende da:
  - Hardware
  - Compilatore
  - Input
  - Altri fattori: casualità, …

Modello computazionale

- Modello RAM (Random-Access Memory)
  - Memoria principale infinita
    - Ogni cella di memoria può contenere una quantità di dati finita.
    - Impiega lo stesso tempo per accedere ad ogni cella di memoria.
  - Singolo processore + programma
    - In 1 unità di tempo: operazioni di *lettura*, *esecuzione* di una *computazione*, *scrittura*;
    - Addizione, moltiplicazione, assegnamento, confronto, accesso a puntatore
- Il modello RAM costituisce un'astrazione dei moderni calcolatori.

Un altro modello computazionale

- Il modello della Macchina di Turing
  - Nastro di lunghezza infinita
    - In ogni *cella* può essere contenuta una quantità di informazione finita
  - Una testina + un processore + programma
    - In 1 unità di tempo
      - Legge o scrive la cella di nastro corrente e
      - Si muove di 1 cella a sinistra, oppure di 1 cella a destra, oppure resta ferma



Un problema di conteggio

- **Input**
  - Un intero N dove $N \geq 1$.
- **Output**
  - Il numero di coppie ordinate (i , j) tali che i e j sono interi e $1 \leq i \leq j \leq N$.
- Esempio: N=4
  - (1,1), (1,2), (1,3), (1,4), (2,2), (2,3), (2,4), (3,4), (4,4)
  - Output = 10

Algoritmo 1

```
int Count_1(int N)
1     sum = 0                    — 1
2     for i = 1 to N             — 3(N+1)
3       for j = i to N           — 3∑_{i=1}^N [(N+1-i)+1]
4         sum = sum + 1          — 2∑_{i=1}^N (N+1-i)
5     return sum                 — 1
```

Tempo di esecuzione $2+3(N+1)+3\sum_{i=1}^{N}[(N+1-i)+1]+2\sum_{i=1}^{N}(N+1-i)$ → $\frac{5}{2}N^2+\frac{17}{2}N+5$

Algoritmo 2

```
int Count_2(int N)
1     sum = 0                    — 1
2     for i = 1 to N             — 3(N+1)
3       sum = sum + (N+1-i)      — 4N
4     return sum                 — 1
```

Il tempo di esecuzione è $7N+5$

Ma osserva-mo che: $\sum_{i=1}^{N}(N+1-i)=\sum_{i=1}^{N}i=N(N+1)/2$

Algoritmo 3

$$\sum_{i=1}^{N}(N+1-i)=\sum_{i=1}^{N}i=N(N+1)/2$$

```
int Count_3(int N)
1     sum = N(N+1)/2             — 4
2     return sum                 — 1
```

Il tempo di esecuzione è 5 unità di tempo

Riassunto dei tempi di esecuzione

Algoritmo	Tempo di Esecuzione
Algoritmo 1	$\frac{5}{2}N^2+\frac{17}{2}N+5$
Algoritmo 2	$7N+5$
Algoritmo 3	5

Ordine dei tempi di esecuzione

Supponiamo che 1 operazone atomica impieghi 1 μs.

	1000	10000	100000	1000000	10000000
N	1 μs	10 μs	100 μs	1 ms	10 ms
20N	20 μs	200 μs	2 ms	20 ms	200 ms
N $\log_2$ N	9.96	132 μs	1.66 ms	19.9 ms	232 ms
20N $\log_2$ N	199 μs	2.7 ms	33 ms	398 ms	4.6 sec
N^2	1 ms	100 ms	10 sec	17 min	1.2 giorni
$20N^2$	20 ms	2 sec	3.3 min	5.6 ore	23 giorni
N^3	1 sec	17 min	12 gior.	32 anni	32 millenni

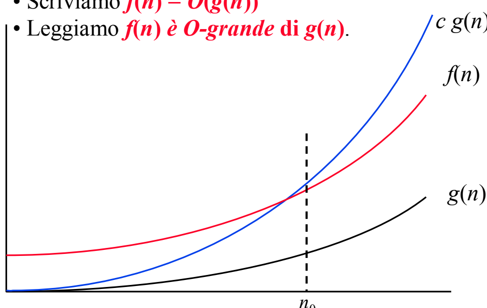
Riassunto dei tempi di esecuzione

Algoritmo	Tempo di Esecuzione	Ordine del Tempo di Esecuzione
Algoritmo 1	$\frac{5}{2}N^2 + \frac{17}{2}N + 5$	N^2
Algoritmo 2	$7N+5$	N
Algoritmo 3	5	Costante

Limite superiore asintotico

esiste $c>0, n_0>0$ t.c. $f(n) \leq c\, g(n)$ per tutti gli $n \geq n_0$

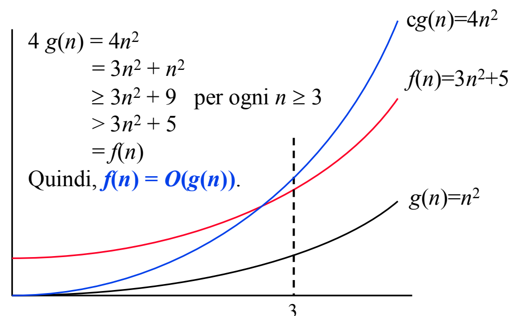
- $g(n)$ è detto un **limite superiore asintotico** di $f(n)$.
- Scriviamo $f(n) = O(g(n))$
- Leggiamo $f(n)$ è ***O-grande*** di $g(n)$.



Esempio di limite superiore asintotico

$$\begin{aligned} 4\, g(n) &= 4n^2 \\ &= 3n^2 + n^2 \\ &\geq 3n^2 + 9 \quad \text{per ogni } n \geq 3 \\ &> 3n^2 + 5 \\ &= f(n) \end{aligned}$$

Quindi, $f(n) = O(g(n))$.



Esercizio sulla notazione O

- **Mostrare che $3n^2+2n+5 = O(n^2)$**

$$\begin{aligned} 10n^2 &= 3n^2 + 2n^2 + 5n^2 \\ &\geq 3n^2 + 2n + 5 \text{ per } n \geq 1 \end{aligned}$$

$c = 10$, $n_0 = 1$

Utilizzo della notazione O

- **In genere quando impieghiamo la notazione *O*, utilizziamo l'espressione più "*semplice*" all'interno della notazione:**
- **Scriveremo quindi**
  - $3n^2+2n+5 = O(n^2)$
- **Anche se seguenti sono tutte corrette ma in genere *non useremo*:**
  - $3n^2+2n+5 = O(3n^2+2n+5)$
  - $3n^2+2n+5 = O(n^2+n)$
  - $3n^2+2n+5 = O(3n^2)$

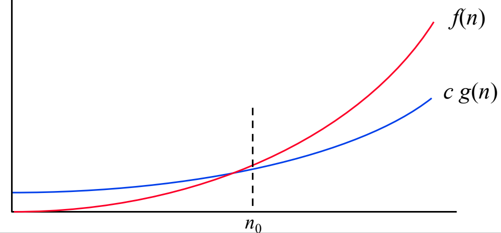
Esercizi sulla notazione O

- $f_1(n) = 10\, n + 25\, n^2$ • $O(n^2)$
- $f_2(n) = 20\, n \log n + 5\, n$ • $O(n \log n)$
- $f_3(n) = 12\, n \log n + 0.05\, n^2$ • $O(n^2)$
- $f_4(n) = n^{1/2} + 3\, n \log n$ • $O(n \log n)$

Limite inferiore asintotico

esiste $c>0, n_0>0$ t.c. $f(n) \geq c\, g(n)$ per tutti gli $n \geq n_0$

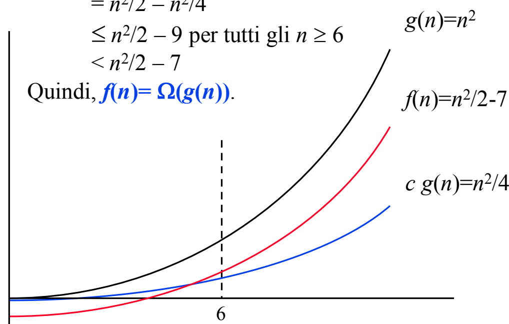
- $g(n)$ è detto un **limite inferiore asintotico** di $f(n)$.
- Scriviamo $f(n) = \Omega(g(n))$
- Leggiamo ***f(n) è* Omega-grande di *g(n)***.



Esempio di limite inferiore asintotico

$g(n)/4 = n^2/4$
$= n^2/2 - n^2/4$
$\leq n^2/2 - 9$ per tutti gli $n \geq 6$
$< n^2/2 - 7$

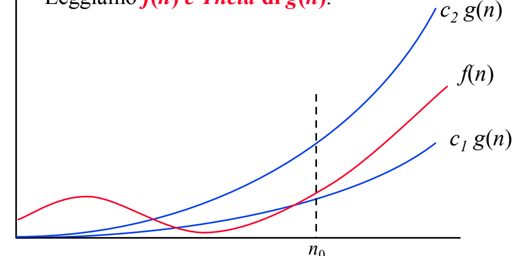
Quindi, $f(n) = \Omega(g(n))$.



Limite asintotico stretto

$f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

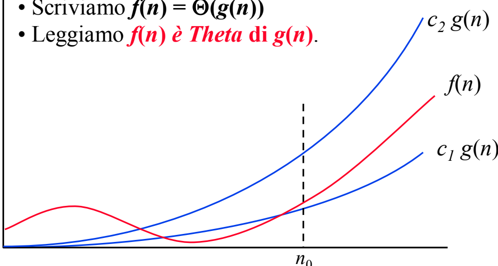
- $g(n)$ è detto un **limite asintotico stretto** di $f(n)$.
- Scriviamo $f(n) = \Theta(g(n))$
- Leggiamo ***f(n) è Theta* di *g(n)***.



Limite asintotico stretto

esistono $c_1>0$, $c_2>0$ e $n_0>0$ tali che
$c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ per tutti gli $n \geq n_0$

- $g(n)$ è detto un **limite asintotico stretto** di $f(n)$.
- Scriviamo $f(n) = \Theta(g(n))$
- Leggiamo ***f(n) è Theta* di *g(n)***.



Riassunto della notazione asintotica

- **O: *O-grande*: limite superiore asintotico**
- **Ω: *Omega-grande*: limite inferiore asintotico**
- **Θ: *Theta*: limite asintotico stretto**

- **Usiamo la *notazione asintotica* per dare un limite ad una funzione ($f(n)$), a meno di un fattore costante (c).**

Teoremi sulla notazione asintotica

Teoremi:

- $f(n) = O(g(n))$ se e solo se $g(n) = \Omega(f(n))$.
- Se $f_1(n)=O(f_2(n))$ e $f_2(n)=O(f_3(n))$, allora $f_1(n)=O(f_3(n))$
- Se $f_1(n)=\Omega(f_2(n))$ e $f_2(n)=\Omega(f_3(n))$, allora $f_1(n)=\Omega(f_3(n))$
- Se $f_1(n)=\Theta(f_2(n))$ e $f_2(n)=\Theta(f_3(n))$, allora $f_1(n)=\Theta(f_3(n))$
- Se $f_1(n) = O(g_1(n))$ e $f_2(n) = O(g_2(n))$, allora

  $$O(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$

- Se $f(n)$ è un *polinomio* di grado d, allora $f(n) = \Theta(n^d)$

Algoritmo 4

```
int Count_4( int N)
1    sum = 0                  O(1)
2    for i =1 to N            O(N)
3        for j =1 to N        O(N^2)
4          if i <= j then     O(N^2)
5             sum = sum+1     O(N^2)
6    return sum               O(1)
```

Il tempo di esecuzione è $O(N^2)$

Tempi di esecuzione asintotici

Algoritmo	Tempo di Esecuzione	Limite asintotico
Algoritmo 1	$\frac{5}{2}N^2+\frac{17}{2}N+5$	$O(N^2)$
Algoritmo 2	$7N+5$	$O(N)$
Algoritmo 3	5	$O(1)$
Algoritmo 4	$6N^2+6N+4$	$O(N^2)$

Somma Massima della Sottosequenza

- **Input**
  - Un intero N dove $N \geq 1$.
  - Una lista $(a_1, a_2, \ldots, a_N)$ di N interi.
- **Output**
  - Un intero S tale che $S = \sum_{k=i}^{j} a_k$ dove $1 \leq i,j \leq N$ e S è il più grande possibile.

- Esempio:
  - N=9, (2,-4,8,3,-5,4,6,-7,2)
  - Output = 8+3-5+4+6 = 16

Algoritmo 1

```
int Max_seq_sum_1(int N, array a[])                 O(1)
  maxsum = 0
  for i=1 to N                                      O(N)
    for j=i to N                                    O(N^2)
      sum = 0
      for k=i to j                                  O(N^3)
        sum = sum + a[k]
      maxsum = max(maxsum,sum)
  return maxsum
```

Tempo di esecuzione $\sum_{i=1}^{N}\sum_{j=i}^{N}\sum_{k=i}^{j} 3 = O(N^3)$

Algoritmo 2

```
int Max_seq_sum_2( int N, array a[])
  maxsum = 0                                        O(1)
  for i=1 to N                                      O(N)
    sum = 0
    for j=i to N                                    O(N^2)
      sum = sum + a[i]
      maxsum = max(maxsum,sum)
  return maxsum
```

Tempo di esecuzione $\sum_{i=1}^{N}\sum_{j=i}^{N} 3 = O(N^2)$

Ordinamento di una sequenza

- **Input :** una sequenza di numeri.
- **Output :** una permutazione (riordinamento) tale che tra ogni 2 elementi adiacenti nella sequenza valga "qualche" relazione di ordinamento (ad es. ≤).
- *Insert Sort*
  - È efficiente solo per sequenze piccole;
  - Algoritmo di ordinamento sul posto.

1) La sequenza viene scandita a partire dal secondo elemento tramite j;
2) i viene assegnato all'elemento precedente j (si considera la porzione di array da 1 fino ad i già ordinata e l'elemento j è quindi il primo della sequenza non-ordinata);
3) Si cerca la posizione corretta per l'elemento j nella porzione ordinata (da 1 ad i).
4) Si torna al passo 1 se la sequenza non è terminata;

Insert Sort

<u>**Algoritmo**</u> **:**

- $A[1..n]$: sequenza numeri di input
- ***Key*** : numero corrente da mettere in ordine

```
1  for j = 2 to Length(A) do
2     Key = A[j]
      // Scelta del j-esimo elemento da ordinare
3     i = j-1
4     while i > 0 and A[i] > Key do
5        A[i+1] = A[i]
6        i=i-1
7     A[i+1] = Key
```

Analisi di Insert Sort

Istruzione	Volte	Costo
1 `for j = 2 to Length(A) do`	n	c_1
2 `Key = A[j]`	$n-1$	c_2
`// Commento`	$n-1$	0
3 `i = j-1`	$n-1$	c_3
4 `while i>0 and A[i] > Key do`	$\sum_{j=2}^{n} t_j$	c_4
5 `A[i+1] = A[i]`	$\sum_{j=2}^{n} (t_j - 1)$	c_5
6 `i=i-1`	$\sum_{j=2}^{n} (t_j - 1)$	c_6
7 `A[i+1] = Key`	$n-1$	c_7

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{j=2}^{n}t_j+c_5\sum_{j=2}^{n}(t_j-1)+c_6\sum_{j=2}^{n}(t_j-1)+c_7(n-1)$$

Analisi di Insert Sort: Caso migliore

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{j=2}^{n}t_j+c_5\sum_{j=2}^{n}(t_j-1)+c_6\sum_{j=2}^{n}(t_j-1)+c_7(n-1)$$

Il *caso migliore* si ha quando l'array è già ordinato:

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{j=2}^{n}t_j+c_7(n-1)$$

Inoltre, in questo caso t_j è **1**, quindi:

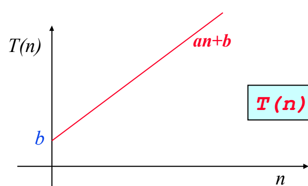
$$T(n)=(c_1+c_2+c_3+c_4+c_7)n-(c_2+c_3+c_4+c_7)$$

`T(n) = an+b`

Analisi di Insert Sort: Caso migliore

$$T(n)=(c_1+c_2+c_3+c_4+c_7)n-(c_2+c_3+c_4+c_7)$$

`T(n) = an+b`



$T(n) = \Theta(n)$

Analisi di Insert Sort: Caso peggiore

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{j=2}^{n}t_j+c_5\sum_{j=2}^{n}(t_j-1)+c_6\sum_{j=2}^{n}(t_j-1)+c_7(n-1)$$

Il *caso peggiore* si ha quando l'array è in ordine inverso.
In questo caso t_j è j (perché?)

$$\sum_{j=2}^{n}t_j=\sum_{j=1}^{n}t_j-1=\frac{n(n+1)}{2}-1$$

$$\sum_{j=2}^{n}(t_j-1)=\sum_{j=2}^{n}t_j-\sum_{j=2}^{n}1=\frac{n(n+1)}{2}-1-(n-1)=\frac{n(n-1)}{2}$$

Quindi:

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\left(\frac{n(n+1)}{2}-1\right)+c_5\left(\frac{n(n-1)}{2}\right)+c_6\left(\frac{n(n-1)}{2}\right)+c_7(n-1)$$

Analisi di Insert Sort: Caso peggiore

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

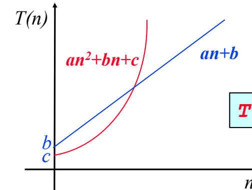
$$T(n) = \left(\frac{c_4 + c_5 + c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$$

`T(n) = an² + bn + c`

Analisi di Insert Sort: Caso peggiore

$$T(n) = \left(\frac{c_4 + c_5 + c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$$

`T(n) = an² + bn + c`



`T(n) = O(n²)`

Analisi di Insert Sort: Caso medio

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

Il ***caso medio*** è il valore medio del tempo di esecuzione.
Supponiamo di scegliere una ***sequenza casuale*** e che tutte le sequenze abbiano uguale probabilità di essere scelte.
In media, ***metà degli elementi*** ordinati saranno ***maggiori*** dell'elemento che dobbiamo sistemare.
In media quindi dovremo ***controllare*** ad ogni ciclo **`while`** ***metà del sottoarray***.
Quindi t_j sarà $j/2$.

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} \frac{j}{2} = \frac{1}{2}\left(\sum_{j=1}^{n} j - 1\right) = \frac{n^2 + n - 2}{4}$$

$$\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} \left(\frac{j}{2} - 1\right) = \frac{n^2 - 3n + 2}{4}$$

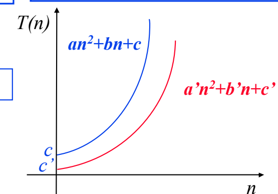
Analisi di Insert Sort: Caso medio

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} \frac{j}{2} = \frac{1}{2}\left(\sum_{j=1}^{n} j - 1\right) = \frac{n^2 + n - 2}{4}$$

$$\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} \left(\frac{j}{2} - 1\right) = \frac{n^2 - 3n + 2}{4}$$

`T(n) = a'n² + b'n + c'`



Analisi del Caso Migliore e Caso Peggiore

- **Analisi del Caso Migliore**
  - **Ω-grande, limite inferiore, del tempo di esecuzione per un qualunque *input di dimensione N*.**
- **Analisi del Caso Peggiore**
  - ***O*-grande, limite superiore, del tempo di esecuzione per un qualunque *input di dimensione N*.**

Analisi del Caso Medio

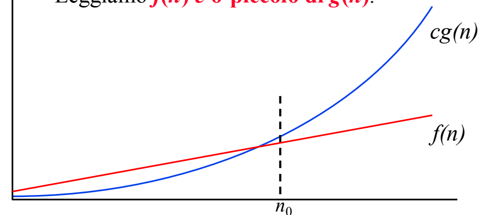
- **Analisi del Caso Medio**
  - **Alcuni algoritmi sono efficienti in pratica.**
  - **L'analisi è in genere molto più difficile.**
  - **Bisogna generalmente assumere che tutti gli input siano ugualmente probabili.**
  - **A volte non è ovvio quale sia il valore medio.**

Ulteriori notazioni asintotiche

- ***O*, *Ω* ci forniscono un modo per parlare di limiti che possono essere asintoticamente *stretti*:**
  - **ad esempio $2n = O(n)$, e anche $2n = \Omega(n)$**
  - **$2n = O(n^2)$, ma $2n \neq \Omega(n^2)$**
- **Esistono notazioni asintotiche per parlare di limiti *non* asintoticamente stretti!**
  - **$O(n) \Rightarrow o(n)$**
  - **$\Omega(n) \Rightarrow \omega(n)$**

Limite asintotico superiore non stretto

- **per ogni $c>0$ esiste un n_0** tale che **$f(n) < c\, g(n)$** per ogni $n \geq n_0$
- $g(n)$ è detto un **limite superiore asintotico non stretto** di $f(n)$.
- Scriviamo **$f(n) = o(g(n))$**
- Leggiamo ***$f(n)$* è o-piccolo di *$g(n)$***.



Limite asintotico superiore non stretto

- **$f(n) = o(g(n))$** significa che per ogni $c>0$ esiste un n_0 tale che **$f(n) < c\, g(n)$** per ogni $n \geq n_0$
- Una definizione alternativa ***ma equivalente*** è che:

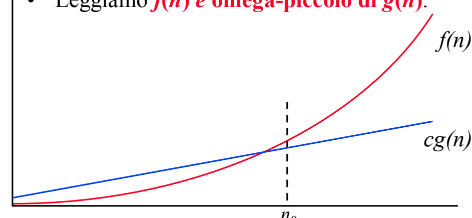
$$f(n) = o(g(n)) \text{ sse } \lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$$

o equivalentemente che:

$$f(n) = o(g(n)) \text{ sse } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Limite asintotico inferiore non stretto

- **per ogni $c>0$ esiste un n_0** tale che **$f(n) > c\, g(n)$** per ogni $n \geq n_0$
- $g(n)$ è detto un **limite inferiore asintotico non stretto** di $f(n)$.
- Scriviamo **$f(n) = \omega(g(n))$**
- Leggiamo ***$f(n)$* è omega-piccolo di *$g(n)$***.



Limite asintotico inferiore non stretto

- **$f(n) = \omega(g(n))$** significa che per ogni $c>0$ esiste un n_0 tale che **$f(n) > c\, g(n)$** per ogni $n \geq n_0$
- Una definizione alternativa ***ma equivalente*** è che:

$$f(n) = \omega(g(n)) \text{ sse } \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

o equivalentemente che:

$$f(n) = \omega(g(n)) \text{ sse } \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Tecniche di sviluppo di algoritmi

- Agli esempi di algoritmi visti fin'ora seguono l'***approccio incrementale*: la soluzione viene costruita passo dopo passo.**
- ***Insert sort*: dopo aver ordinato una sottoparte dell'array, si *inserisce al sposto giusto* un altro elemento, *ottenendo un sottoarray ordinato più grande*.**
- **Esistono *altre tecniche di sviluppo di algoritmi* con filosofie differenti:**
  - **Divide-et-Impera**

Divide-et-Impera

- Il problema viene scomposto in *sottoproblemi simili*, che vengono *risolti separatamente*. Le *soluzioni dei sottoproblemi* vengono infine *fuse* insieme per ottenere la soluzione al problema più complesso.
- Consiste di 3 passi:
  - *Divide* il problema in vari sottoproblemi, tutti simili al problemi originario ma più semplici.
  - *Impera* (conquista) i sottoproblemi risolvendoli ricorsivamente. Quando un sottoproblema diviene banale, risolverlo direttamente.
  - *Fonde* le soluzioni dei sottoproblemi per ottenere la soluzione del (sotto)problema che li ha originati.

Divide-et-Impera e ordinamento

- **Input :** una sequenza di numeri.
- **Output :** una permutazione (riordinamento) tale che tra ogni 2 elementi adiacenti nella sequenza valga "qualche" relazione di ordinamento (ad es. ◙).
- *Merge Sort* (divide-et-impera)
  - *Divide* la sequenza di *n* elementi in 2 *sottosequenze* di *n*/2 elementi ciascuna.
  - *Impera* (conquista) i sottoproblemi ordinando le sottosequenze richiamando *ricorsivamente Merge Sort* stesso. Quando una sequenza è unitaria il sottoproblema è banalmente risolto.
  - *Fonde* insieme le soluzioni dei sottoproblemi per ottenere la sequenza ordinata del (sotto)problema originario.

Merge Sort

Algoritmo :

- *A*[1..*n*]: sequenza dei numeri in input
- *p,r*: indici degli estremi della sottosequenza da ordinare

```
Merge_Sort(array A, int p,r)
1  if p < r  then
2        q = ♏(p+q)/2◆
3        Merge_Sort(A,p,q)
4        Merge_Sort(A,q+1,r)
5        Merge(A,p,q,r)
```

Divide (riga 2) — Impera (righe 3-4) — Fusione (riga 5)

Merge Sort: analisi

```
Merge_Sort(array A, int p,r)
1  if p < r  then
2        q = ♏(p+q)/2◆
3        Merge_Sort(A,p,q)
4        Merge_Sort(A,q+1,r)
5        Merge(A,p,q,r)
```

T(n) = ✈*(1)* se *n*=1

T(n) = 2 *T(n/2)* + $T_{merge}(n)$

$T_{merge}(n)$ = ✈*(n)*

Equazione di Ricorrenza

$$T(n)=\begin{cases}\Theta(1) & se\ n=1\\ 2T(n/2)+\Theta(n) & se\ n>1\end{cases}$$

Merge Sort: analisi

```
Merge_Sort(array A, int p,r)
1  if p < r  then
2        q = ♏(p+q)/2◆
3        Merge_Sort(A,p,q)
4        Merge_Sort(A,q+1,r)
5        Merge(A,p,q,r)
```

$$T(n)=\begin{cases}\Theta(1) & se\ n=1\\ 2T(n/2)+\Theta(n) & se\ n>1\end{cases}$$

Soluzione: *T(n)* = ✈*(n log n)*

Divide-et-Impera: Equazioni di ricorrenza

- *Divide*: *D(n)* tempo per dividere il problema
- *Impera*: se si divide il problema in *a* sottoproblemi, ciascuno di dimensione *n/b*, il tempo per conquistare i sottoproblemi sarà *aT(n/b)*.

  Quando un sottoproblema diviene banale (input minore o uguale ad una costante *c*), usiamo ✈*(1)*.
- *Fondi*: *D(n)* tempo per fondere le soluzioni dei sottoproblemi.

$$T(n)=\begin{cases}\Theta(1) & se\ n\le c\\ aT(n/b)+D(n)+C(n) & se\ n>c\end{cases}$$

Gli argomenti fino ad oggi

- **Analisi della bontà di un algoritmo**
  - **Correttezza, utilizzo delle risorse, semplicità**
- **Modello computazionali: modello RAM**
- **Tempo di esecuzione degli algoritmi**
- **Notazione asintotica: *O*-grande, Ω-grande, Θ**
- **Analisi del Caso Migliore, Caso Peggiore e del Caso Migliore**
- **Notazione asintotica: *o*-piccolo, ω-piccolo**
- **Analisi di algoritmi *Divide et Impera*: Merge Sort**