

Algoritmi e Strutture Dati

Introduzione

Informazioni utili

- T.H. Cormen, C.E. Leiserson, R.L. Rivest
“**Introduzione agli algoritmi**”. Jackson Libri

- Sito web con le **slides** del corso:

<http://people.na.infn.it/~bene/ASD/>

Complessità degli algoritmi

- **Analisi delle prestazioni degli algoritmi**
- **Servirà un Modello Computazionale di riferimento.**
- **Tempo di esecuzione degli algoritmi**
- **Notazione asintotica**
- **Analisi del Caso Migliore, Caso Peggior e del Caso Medio**
- **Esempi di analisi di algoritmi di ordinamento.**

Analisi di un algoritmo

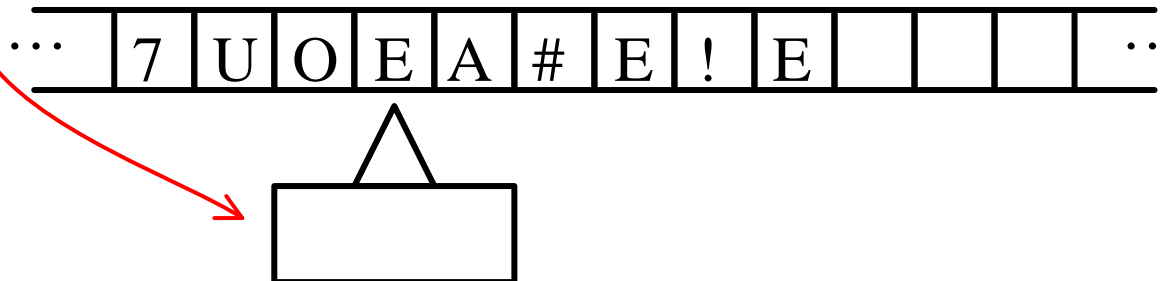
- **Correttezza**
 - **Dimostrazione formale (matematica)**
 - **Ispezione informale**
- **Utilizzo delle risorse**
 - **Tempo di esecuzione**
 - **Utilizzo della memoria**
 - **Altre risorse: banda di comunicazione**
- **Semplicità**
 - **Facile da capire, modificare e mantenere**

Tempo di esecuzione

- Il *tempo di esecuzione* di un programma dipende da:
 - **Hardware**
 - **Compilatore/Interprete**
 - *Tipo e dimensione dell'input*
 - **Altri fattori: casualità, ...**

Un noto modello computazionale

- Il modello della **Macchina di Turing**
 - **Nastro di lunghezza infinita**
 - In ogni *cella* può essere contenuta una quantità di informazione finita (un simbolo)
 - **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



Il modello computazionale RAM

- **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 è una semplificazione dei moderni computer.**

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 siano interi e $1 \leq i \leq j \leq N$.
- Esempio: $N=4$
 - $(1,1), (1,2), (1,3), (1,4), (2,2), (2,3), (2,4), (3,3), (3,4), (4,4)$
 - Output = 10

Algoritmo 1

```
int Count_1(int N)
```

```
1    sum = 0
```

```
2    for i = 1 to N
```

```
3        for j = i to N
```

```
4            sum = sum + 1
```

```
5    return sum
```

1

$2N + 2$

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

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

1

Il tempo di esecuzione è $2 + 2N + 2 + 2 \sum_{i=1}^N (N + 2 - i) + 2 \sum_{i=1}^N (N + 1 - i) = 2N^2 + 6N + 4$

Algoritmo 2

<code>int Count_2(int N)</code>	
<code>1 sum = 0</code>	1
<code>2 for i = 1 to N</code>	$2N + 2$
<code>3 sum = sum + (N+1-i)</code>	$4N$
<code>4 return sum</code>	1

Il tempo di esecuzione è $6N + 4$

**Ma notate
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
```

```
2    return sum
```

Il tempo di esecuzione è **5** unità di tempo

Riassunto dei tempi di esecuzione

Algoritmo	Tempo di Esecuzione
Algoritmo 1	$2N^2 + 6N + 4$
Algoritmo 2	$6N+4$
Algoritmo 3	5

Ordine dei tempi di esecuzione

Supponiamo che **1 operazione atomica**
impieghi **1 $\eta s = 10^{-9}$ s**

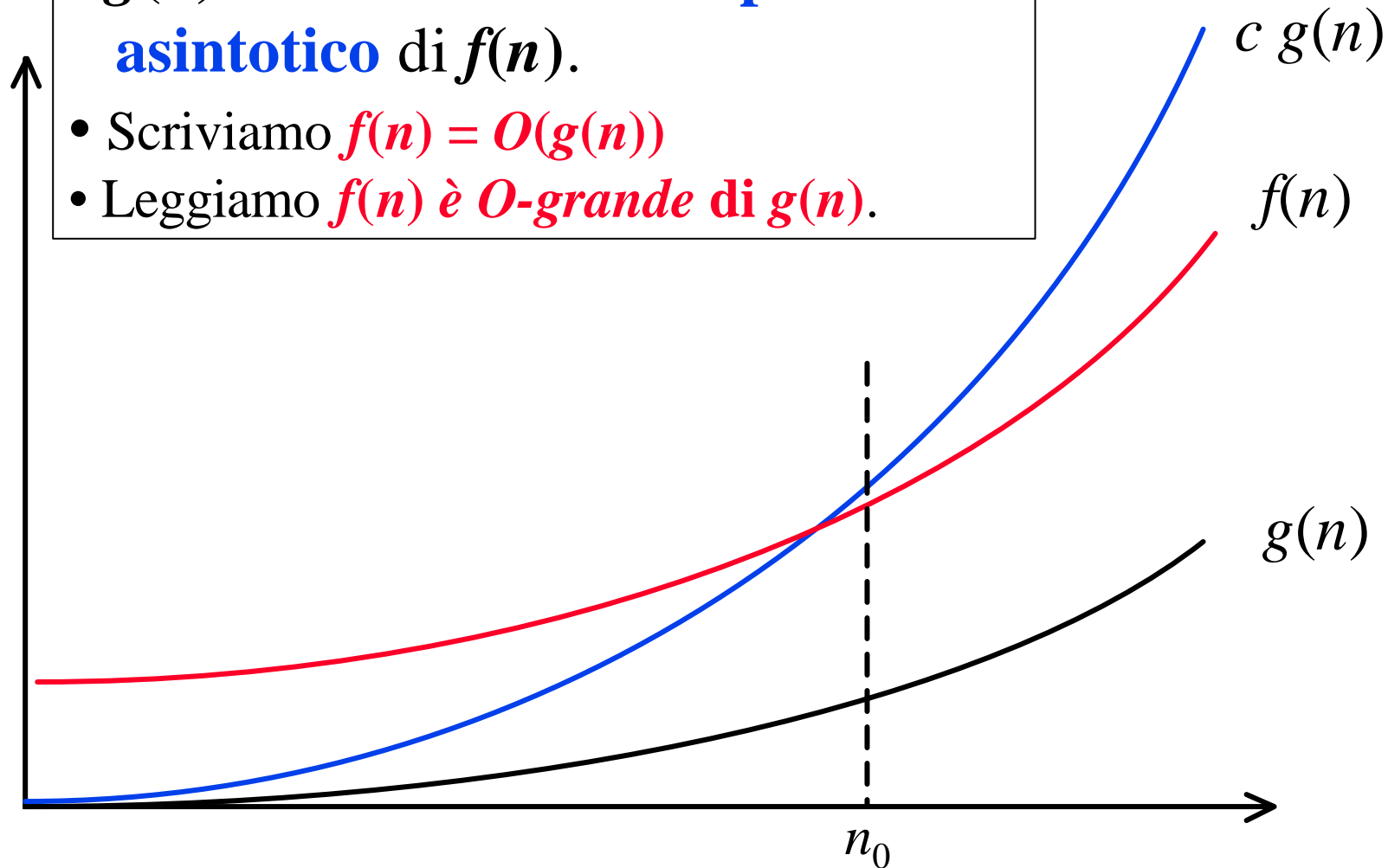
	1.000	10.000	100.000	1.000.000	10.000.000
N	1 μs	10 μs	100 μs	1 ms	10 ms
20N	20 μs	200 μs	2 ms	20 ms	200 ms
N Log N	9.96 μs	132 μs	1.66 ms	19.9 ms	232 ms
20N Log N	199 μs	2.7 ms	33 ms	398 ms	4.6 sec
N ²	1 ms	100 ms	10 sec	17 min	1.2 giorni
20N ²	20 ms	2 sec	3.3 min	5.6 ore	23 giorni
N ³	1 sec	17 min	12 gior.	32 anni	32 millenni

Riassunto dei tempi di esecuzione

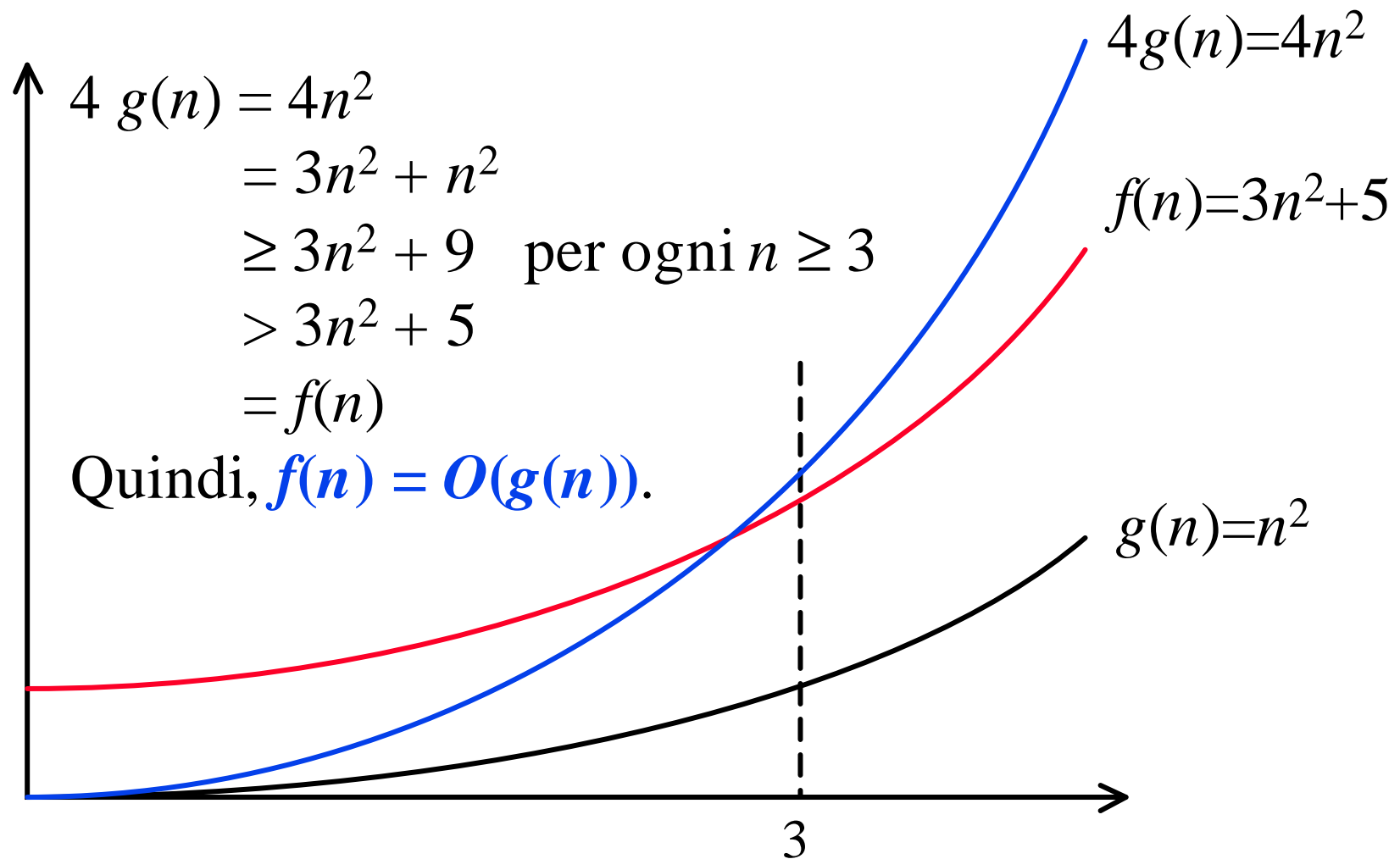
Algoritmo	Tempo di Esecuzione	Ordine del Tempo di Esecuzione
Algoritmo 1	$2N^2 + 6N + 4$	N^2
Algoritmo 2	$6N+4$	N
Algoritmo 3	5	Costante

Limite superiore asintotico

- $\exists c > 0, n_0 > 0$ " $n \geq n_0$. $f(n) \leq c g(n)$
- $g(n)$ 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



Esercizio sulla notazione O

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

$$10n^2 = 3n^2 + 2n^2 + 5n^2$$

$$\geq 3n^2 + 2n + 5 \text{ per } n \geq 1$$

$$c = 10, n_0 = 1$$

Utilizzo della notazione O

- In genere quando impieghiamo la notazione O , utilizziamo la formula più “*semplice*”.

- Scriviamo:

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

- Le seguenti sono tutte corrette ma in genere non le si userà:

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

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

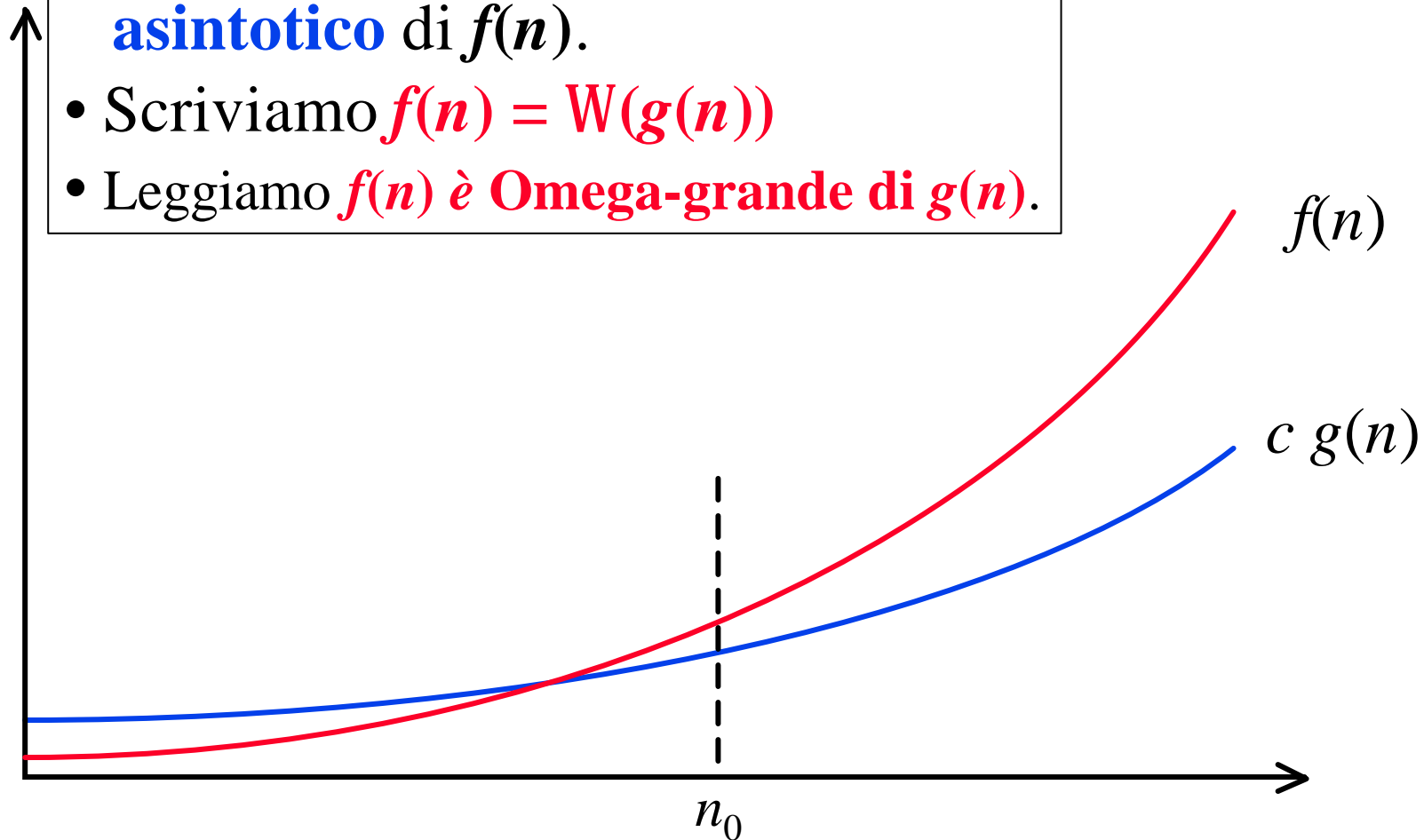
- $3n^2+2n+5 = O(3n^2)$

Esercizi sulla notazione O

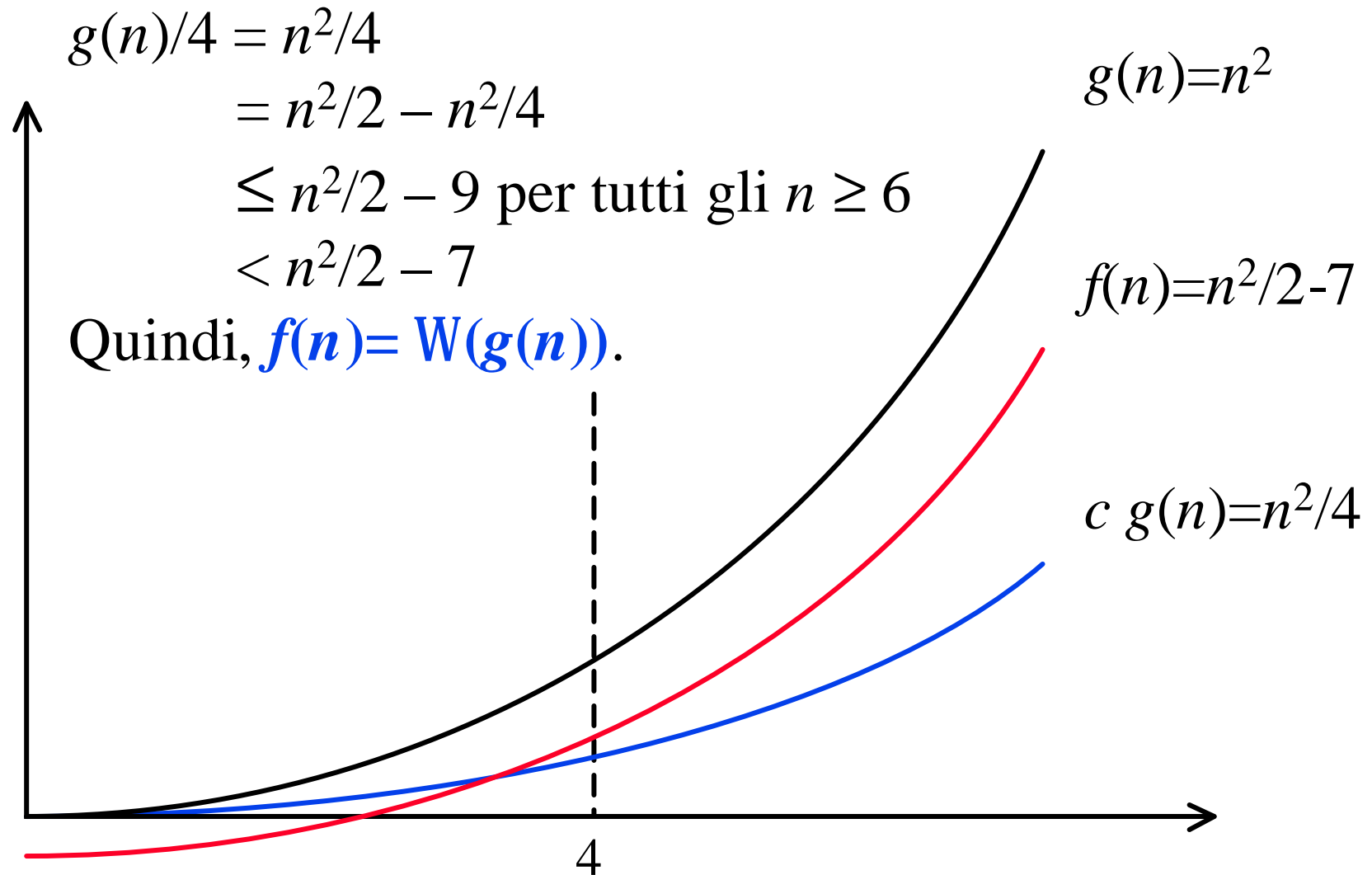
- $f_1(n) = 10n + 25n^2$
 - $f_2(n) = 20n \log n + 5n$
 - $f_3(n) = 12n \log n + 0.05n^2$
 - $f_4(n) = n^{1/2} + 3n \log n$
- $O(n^2)$
 - $O(n \log n)$
 - $O(n^2)$
 - $O(n \log n)$

Limite inferiore asintotico

- $c > 0, n_0 > 0 \mid n \geq n_0. f(n) \geq c g(n)$
- $g(n)$ è detto un **limite inferiore asintotico** di $f(n)$.
- Scriviamo $f(n) = \Omega(g(n))$
- Leggiamo $f(n)$ è Omega-grande di $g(n)$.

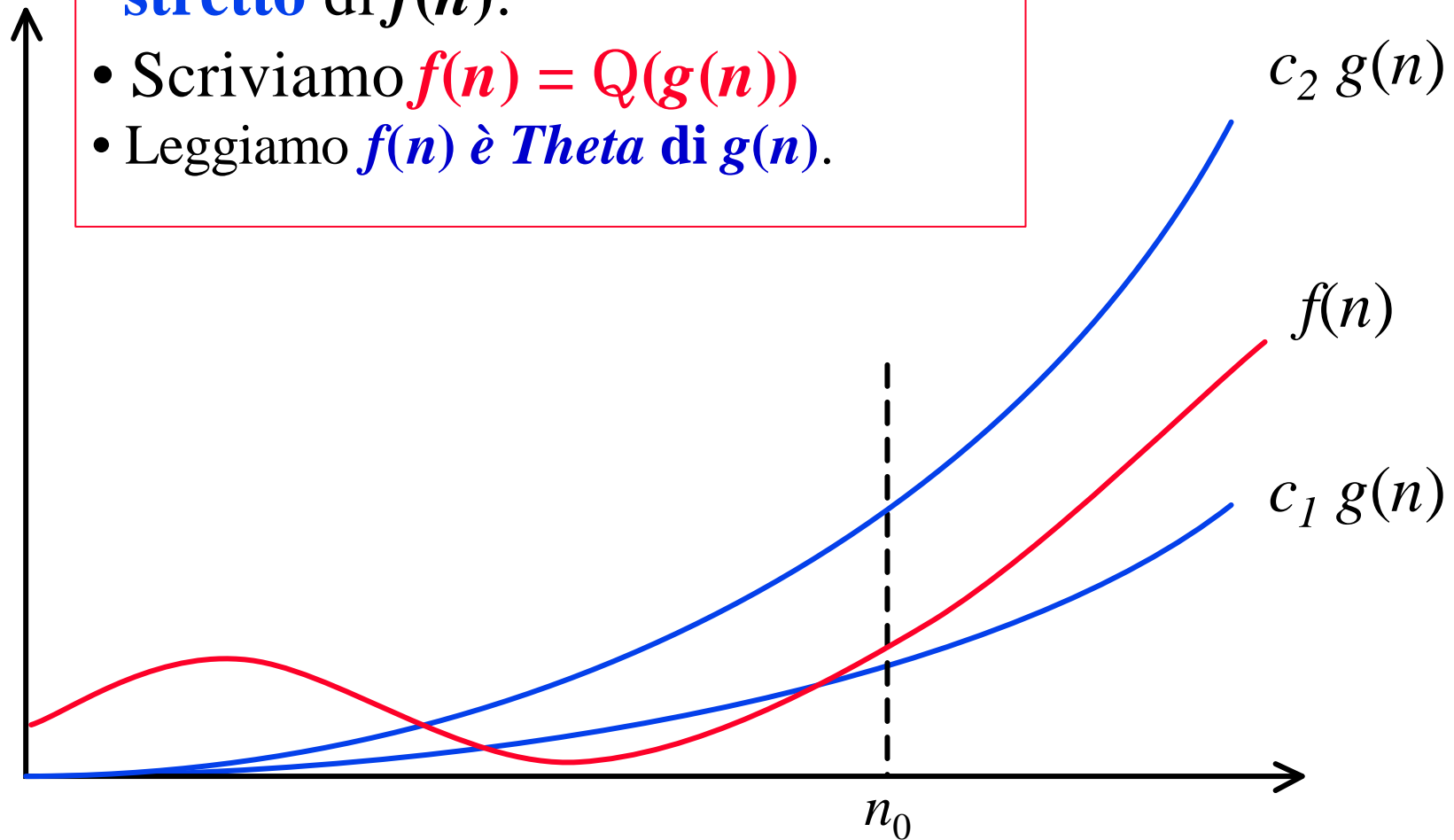


Esempio di limite inferiore asintotico



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 $g(n)$.



Riassunto della notazione asintotica

- ***O***: ***O-grande***: limite superiore asintotico
- ***W***: ***Omega-grande***: limite inferiore asintotico
- ***Q***: ***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:

1. $f(n) = O(g(n))$ se e solo se $g(n) = W(f(n))$.
2. Se $f_1(n) = O(f_2(n))$ e $f_2(n) = O(f_3(n))$, allora $f_1(n) = O(f_3(n))$
3. Se $f_1(n) = W(f_2(n))$ e $f_2(n) = W(f_3(n))$, allora $f_1(n) = W(f_3(n))$
4. Se $f_1(n) = Q(f_2(n))$ e $f_2(n) = Q(f_3(n))$, allora $f_1(n) = Q(f_3(n))$
5. Se $f_1(n) = O(g_1(n))$ e $f_2(n) = O(g_2(n))$, allora
$$O(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$
6. Se $f(n)$ è un *polinomio* di grado d , allora $f(n) = Q(n^d)$

Teoremi sulla notazione asintotica

Proprietà:

Se $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$ allora $f(n) = O(g(n))$

Se $\lim_{n \rightarrow \infty} f(n)/g(n) = k > 0$ allora $f(n) = O(g(n))$
e $f(n) = \Theta(g(n))$
quindi $f(n) = \Theta(g(n))$

Se $\lim_{n \rightarrow \infty} f(n)/g(n) \rightarrow \infty$ allora $f(n) = \Omega(g(n))$

Algoritmo 4: analisi asintotica

<code>int Count_4(int N)</code>	
<code>1 sum = 0</code>	$O(1)$
<code>2 for i =1 to N</code>	$O(N)$
<code>3 for j =1 to N</code>	$O(N^2)$
<code>4 if i <= j then</code>	$O(N^2)$
<code>5 sum = sum+1</code>	$O\left(\frac{N(N+1)}{2}\right) = O(N^2)$
<code>6 return sum</code>	$O(1)$

Il tempo di esecuzione è $O(N^2)$

Tempi di esecuzione asintotici

Algoritmo	Tempo di Esecuzione	Limite asintotico
Algoritmo 1	$2N^2 + 6N + 4$	$O(N^2)$
Algoritmo 2	$6N+4$	$O(N)$
Algoritmo 3	5	$O(1)$
Algoritmo 4	$4N^2 + 5N + 4$	$O(N^2)$

Somma Massima di una sottosequenza contigua

- **Input**

- Un intero N dove $N \geq 1$.
- Una sequenza (a_1, a_2, \dots, 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.
- (tutti gli elementi nella sommatoria sono contigui).

- **Esempio:**

- $N=9, (2, -4, 8, 3, -5, 4, 6, -7, 2)$
- $\text{Output} = 8 + 3 - 5 + 4 + 6 = 16$

Algoritmo 1

```
int Max_seq_sum_1(int N, array a[])  
    maxsum = 0  $O(1)$   
    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 $O(N^3)$

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[j]
```

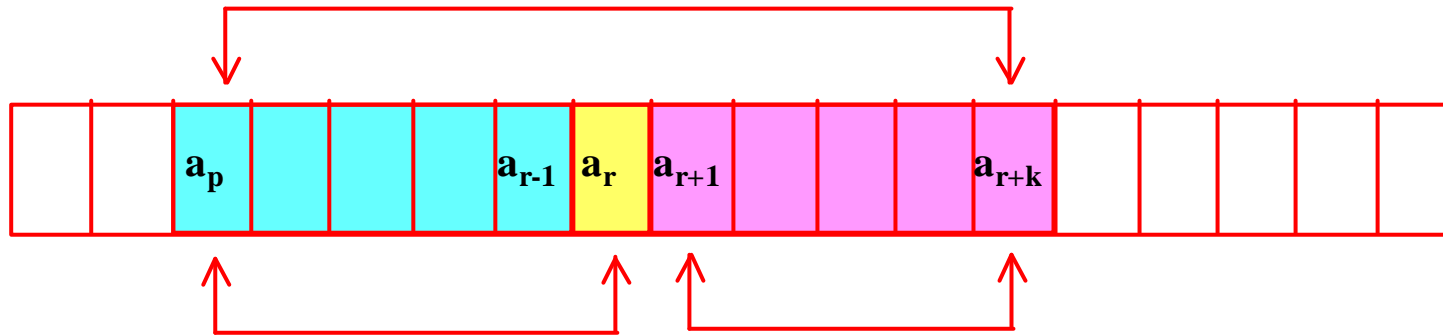
```
            maxsum = max(maxsum, sum)
```

```
    return maxsum
```

Tempo di esecuzione $O(N^2)$

Esiste un algoritmo che risolve il problema in tempo $O(N)$

Algoritmo 3: intuizione



1. Se $a_p + \dots + a_r > 0$ allora

$$a_p + \dots + a_{r+k} > a_{r+1} + \dots + a_{r+k} \quad " \quad k^3 1$$

2. Se $a_p + \dots + a_{r-1} > 0$ ma $a_p + \dots + a_r < 0$ allora

$$a_p + \dots + a_{r+k} < a_{r+1} + \dots + a_{r+k} \quad " \quad k^3 1$$

Algoritmo 3

```
int Max_seq_sum_3(int N, array a[])
```

```
    maxsum = 0
```

$O(1)$

```
    sum = 0
```

```
    for i=1 to N
```

$O(N)$

```
        if (sum + a[i]>0) then
```

```
            sum = sum + a[i]
```

```
        else
```

```
            sum = 0
```

```
            maxsum = max(maxsum, sum)
```

```
    return maxsum
```

Tempo di esecuzione $O(N)$

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. \leq).
- **Insert Sort**
 - È efficiente solo per piccole sequenze di numeri;
 - Algoritmo di ordinamento sul posto.

- 1) La sequenza viene scandita dal primo elemento; l'indice *i*, **inizialmente** assegnato al primo elemento, indica l'elemento corrente;
- 2) Si considera la parte a sinistra di *i* (compreso) già ordinata;
- 3) Si seleziona il primo elemento successivo ad *i* nella sottosequenza non-ordinata assegnando $j = i+1$;
- 4) Si cerca il posto giusto per l'elemento *j* nella sottosequenza ordinata.
- 5) Si incrementa *i*, si torna al passo 3) se la sequenza non è terminata;

Insert Sort

Algoritmo :

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

```
1  for j = 2 to Lenght(A)
2      do Key = A[j]
      /* Scelta del j-esimo elemento da ordinare */
3      i = j-1      /* [1...i] è la porzione ordinata */
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

```
1  for j = 2 to Lenght(A)
2      do Key = A[j]
      /* Commento */
3      i = j-1
4      while i>0 and A[i] > Key
5          do A[i+1] = A[i]
6              i=i-1
7      A[i+1] = Key
```

n c_1

$n-1$ c_2

$n-1$ 0

$n-1$ c_3

$\sum_{j=2}^n t_j$ c_4

$\sum_{j=2}^n t_j - 1$ c_5

$\sum_{j=2}^n t_j - 1$ c_6

$n-1$ c_7

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

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:

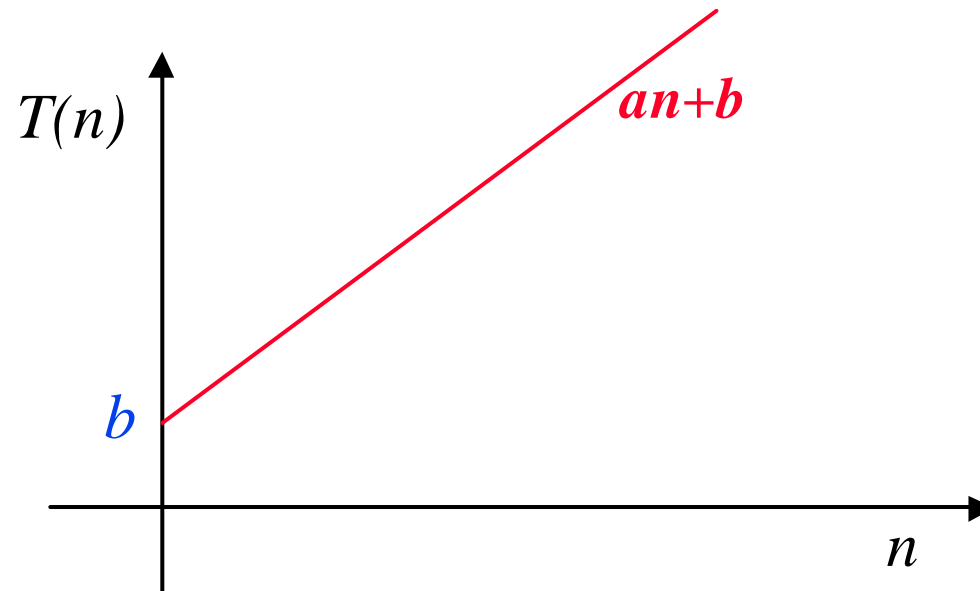
$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

$$T(n) = an + b$$

Analisi di Insert Sort: Caso migliore

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

$$T(n) = an + b$$



Analisi di Insert Sort: Caso peggiore

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

Il caso 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:

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

Analisi di Insert Sort: Caso peggiore

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

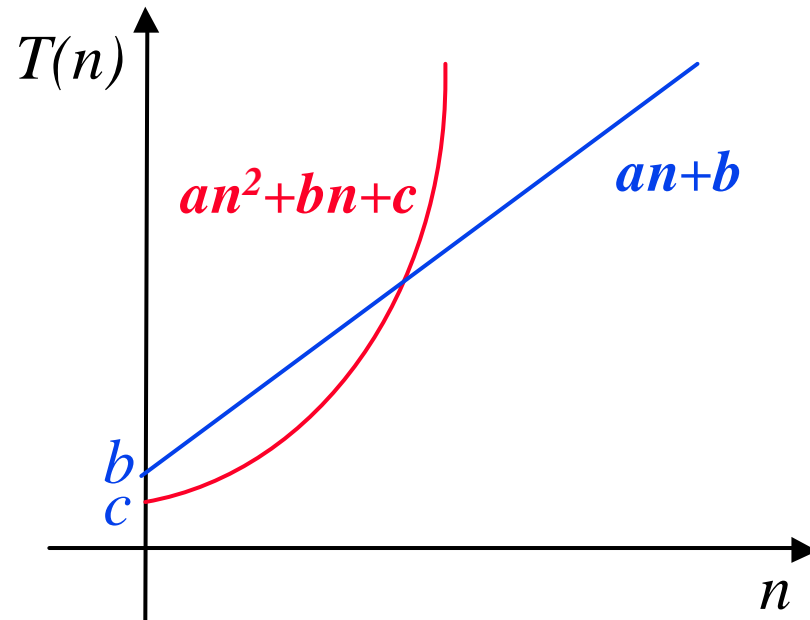
$$T(n) = \left(\frac{c_4 + c_5 + c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

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

Analisi di Insert Sort: Caso peggiore

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

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



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 **controlliamo metà del sottoarray** ad ogni ciclo **while**.

Quindi t_j è $j/2$.

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

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

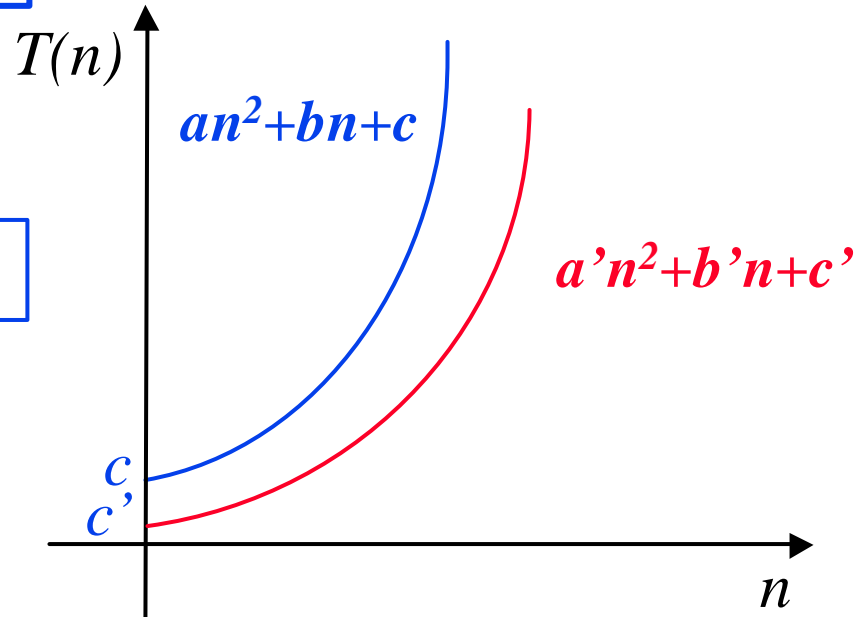
Analisi di Insert Sort: Caso medio

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

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

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

$$T(n) = a'n^2 + b'n + c'$$



Analisi del Caso Migliore e Caso Peggior

- **Analisi del Caso Migliore**
 - Ω -grande, limite inferiore, del tempo di esecuzione per un qualunque *input di dimensione* N .
- **Analisi del Caso Peggior**
 - 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 la media.**

Tecniche di sviluppo di algoritmi

- Agli esempi visti fino ad ora seguono l'***approccio incrementale***: la soluzione viene costruita passo dopo passo.
- ***Insert sort*** avendo ordinato una sottoparte dell'array, inserisce al posto 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 suddiviso in sottoproblemi analoghi, che vengono risolti separatamente. Le soluzioni dei sottoproblemi vengono infine fuse insieme per ottenere la soluzione dei problemi più complessi.
- Consiste di **3 passi**:
 - *Divide* il problema in vari sottoproblemi, tutti *simili* (tra loro e) al *problema originario* ma più semplici.
 - *Impera* (conquista) i sottoproblemi risolvendoli ricorsivamente. Quando un sottoproblema diviene banale, risolverlo direttamente.
 - *Fondi* 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. \leq).
- *Merge Sort* (divide-et-impera)
 - *Divide:* scompone la sequenza di n elementi in 2 sottosequenze di $n/2$ elementi ciascuna.
 - *Impera:* conquista i sottoproblemi ordinando ricorsivamente le sottosequenze con *Merge Sort* stesso. Quando una sottosequenza è unitaria, il sottoproblema è banale.
 - *Fondi:* compone insieme le soluzioni dei sottoproblemi per ottenere la sequenza ordinata del (sotto-)problema.

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 =  $\lfloor (p+r)/2 \rfloor$ 
```

```
3      Merge_Sort(A, p, q)
```

```
4      Merge_Sort(A, q+1, r)
```

```
5      Merge(A, p, q, r)
```

Divide

Impera

Combina

Esercizio: definire la procedure Merge

Merge Sort: analisi

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

$$T(n) = \Theta(1) \text{ se } n=1$$

$$T(n) = 2T(n/2) + T_{\text{merge}}(n) + \Theta(1)$$

$$T_{\text{merge}}(n) = \Theta(n)$$

Equazione di Ricorrenza

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

Merge Sort: analisi

```
Merge_Sort(array A, int p,r)
```

```
1  if p < r then
```

```
2      q =  $\lfloor (p+r)/2 \rfloor$ 
```

```
3      Merge_Sort(A,p,q)
```

```
4      Merge_Sort(A,q+1,r)
```

```
5      Merge(A,p,q,r)
```

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

Soluzione: $T(n) = \Theta(n \log n)$

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 (*l'input è minore o uguale ad una costante c*), in tempo è $\Theta(1)$.

- **Fondi:** $C(n)$ tempo per comporre le soluzioni dei sottoproblemi nella soluzione più complessa.

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

Gli argomenti trattati

- **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 Peggior e del Caso Medio**