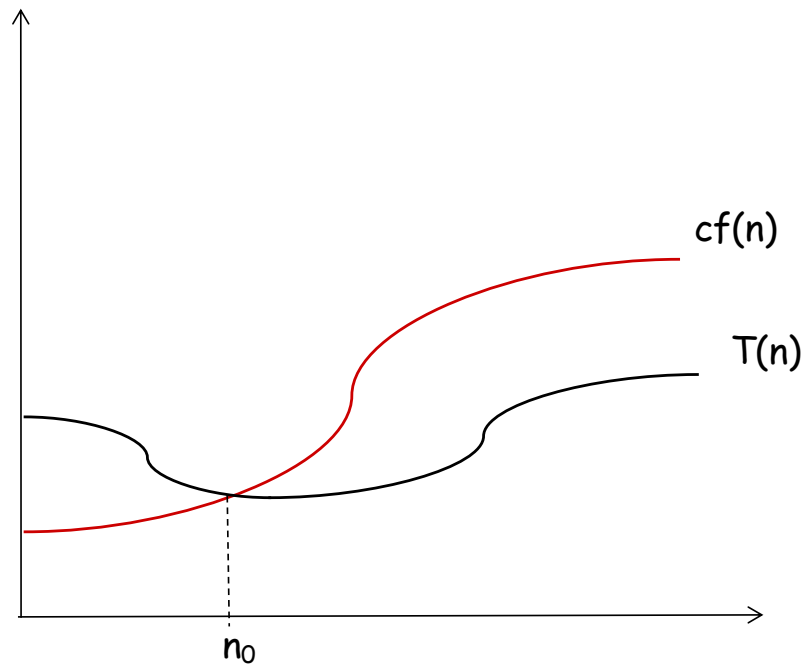


Ordine asintotico di grandezza

Limiti superiori. $0 \leq T(n)$ è $O(f(n))$ se esistono delle costanti $c > 0$ ed $n_0 \geq 0$ tali che per tutti gli $n \geq n_0$ si ha $T(n) \leq c \cdot f(n)$.



Ordine asintotico di grandezza

Limiti superiori. $0 \leq T(n)$ è $O(f(n))$ se esistono delle costanti $c > 0$ ed $n_0 \geq 0$ tali che per tutti gli $n \geq n_0$ si ha $T(n) \leq c \cdot f(n)$.

Esempio: dimostriamo che $3n^2 + 2n$ è $O(n^2)$.

Dobbiamo dimostrare l'esistenza delle costanti $c > 0$ ed $n_0 \geq 0$ tali $3n^2 + 2n \leq cn^2$ per ogni $n \geq n_0$.

RisolviAMO la disequazione $3n^2 + 2n \leq cn^2$:

$$c \geq 3n^2/n^2 + 2n/n^2 = 3 + 2/n$$

Per $n \geq 1$, si ha che $3 + 2/n \leq 3 + 2 = 5$

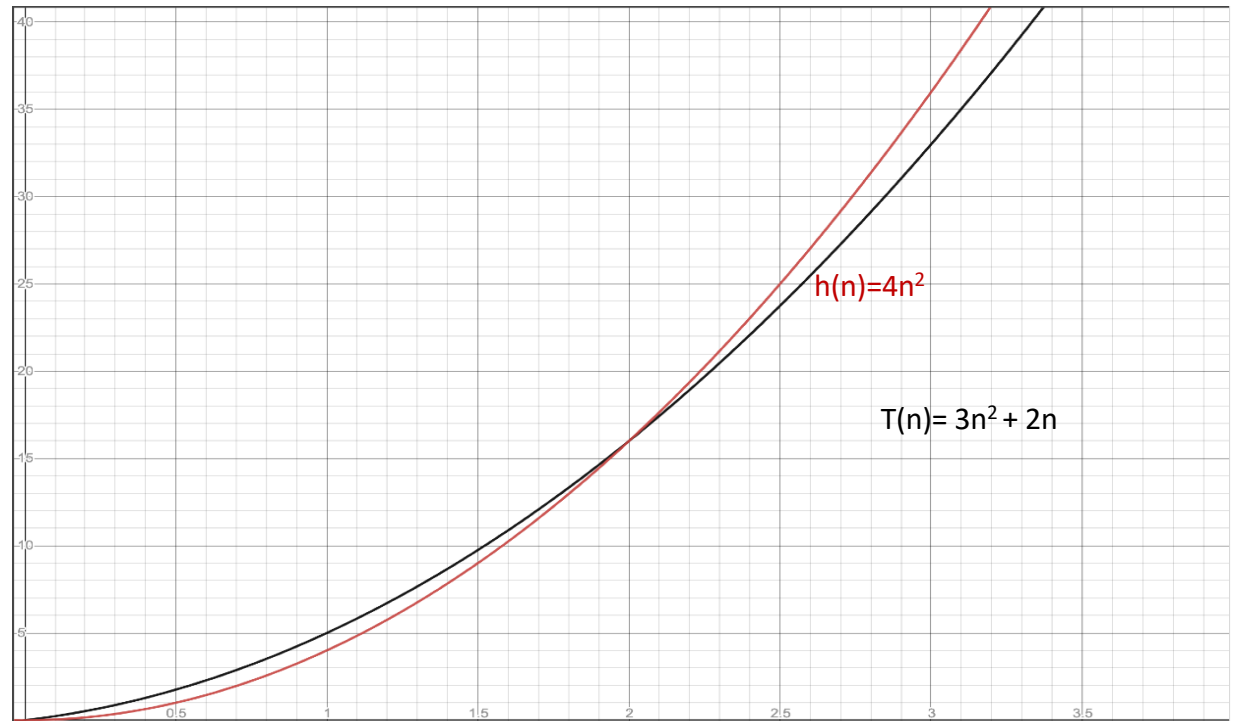
Se poniamo $n_0 = 1$ e $c = 5$, abbiamo che

$$3n^2 + 2n \leq cn^2 \text{ per ogni } n \geq n_0$$

Se partissimo da un valore di n più grande potremmo prendere una costante c più piccola

Se prendessimo $n_0 = 2$, allora potremmo prendere $c = 4$

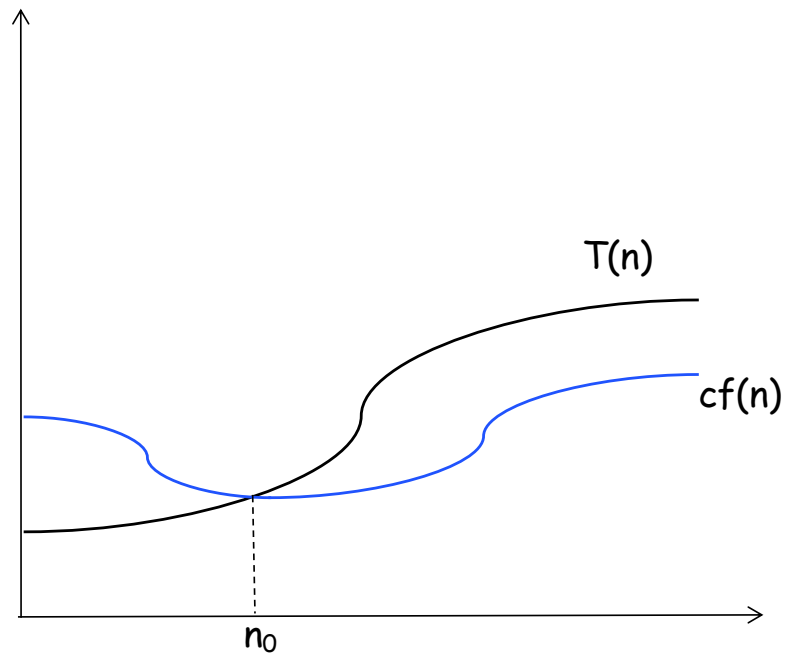
Se prendessimo $c = 3.1$ allora potremmo prendere $n_0 = 20$.



In figura confrontiamo il grafico di $T(n) = 3n^2 + 2n$ con quello di $h(n) = 4n^2$

Ordine asintotico di grandezza

Limiti inferiori. $T(n)$ è $\Omega(f(n))$ se esistono costanti $c > 0$ ed $n_0 \geq 0$ tali che per tutti gli $n \geq n_0$ si ha $T(n) \geq c \cdot f(n) \geq 0$.



Ordine asintotico di grandezza

Limiti inferiori. $T(n)$ è $\Omega(f(n))$ se esistono costanti $c > 0$ ed $n_0 \geq 0$ tali che per tutti gli $n \geq n_0$ si ha $T(n) \geq c \cdot f(n) \geq 0$.

Esempio: dimostriamo che $3n^2 + 2n$ è $\Omega(n^2)$.

Dobbiamo dimostrare l'esistenza delle costanti $c > 0$ ed $n_0 \geq 0$ tali $3n^2 + 2n \geq cn^2$ per ogni $n \geq n_0$.

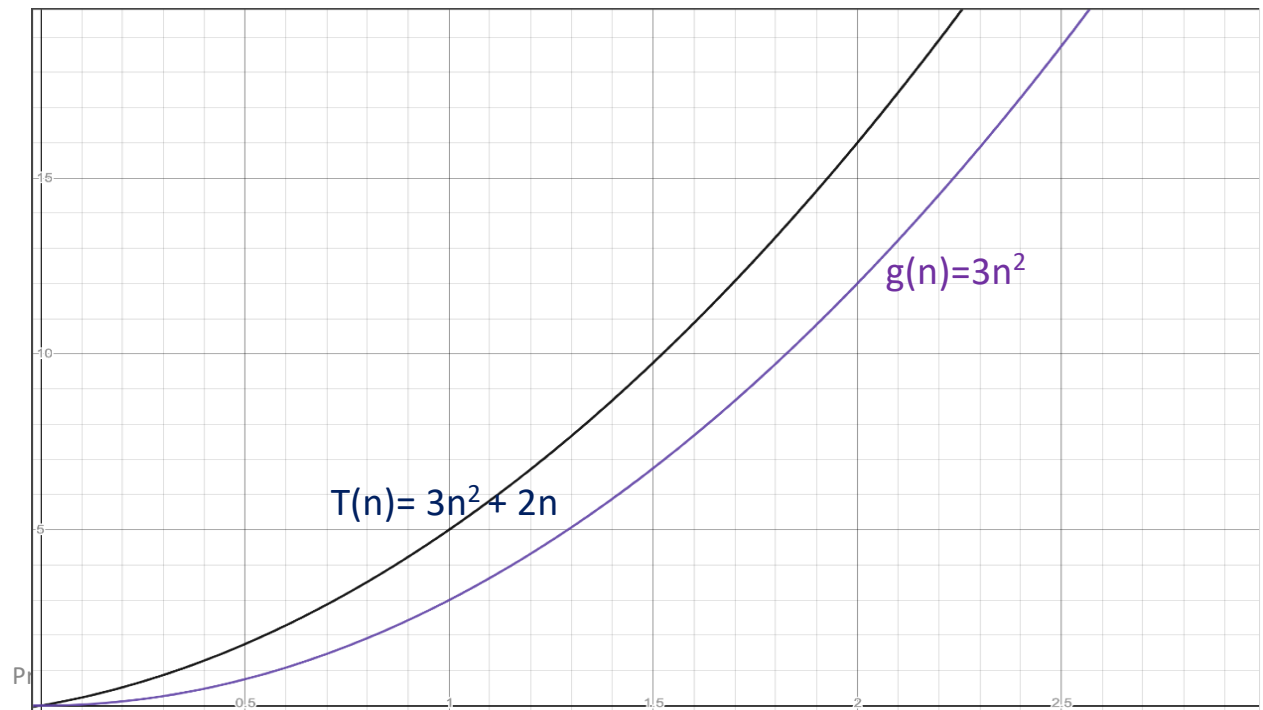
Vera sicuramente per $n=0$ a prescindere dal valore di c . Vediamo quando è vera per altri n .

Risolvi la disequazione $3n^2 + 2n \geq cn^2$. Questa è soddisfatta se e solo se $c \leq 3n^2/n^2 + 2n/n^2 = 3 + 2/n$

Osserviamo che $3 + 2/n \geq 3$ per ogni $n > 0$.

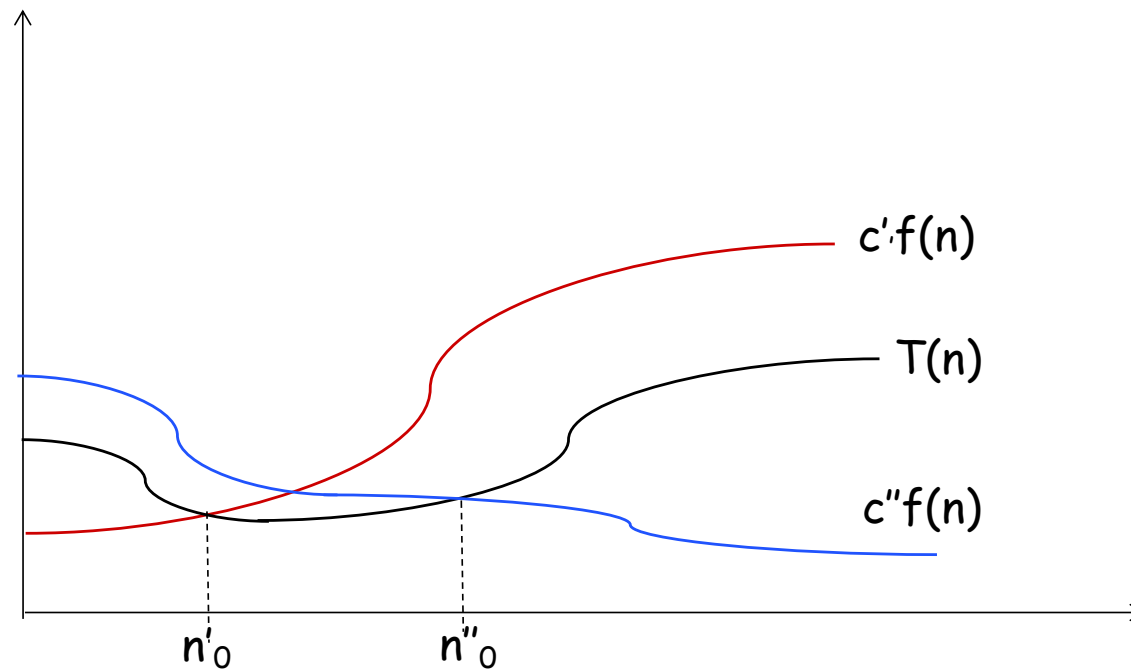
Possiamo quindi prendere $c=3$ ed $n_0 = 0$

Abbiamo quindi trovato le costanti c ed n_0
(con $c=3$ ed $n_0=0$) per cui si ha che $3n^2 + 2n \geq cn^2$ per ogni $n \geq n_0$.



Ordine asintotico di grandezza

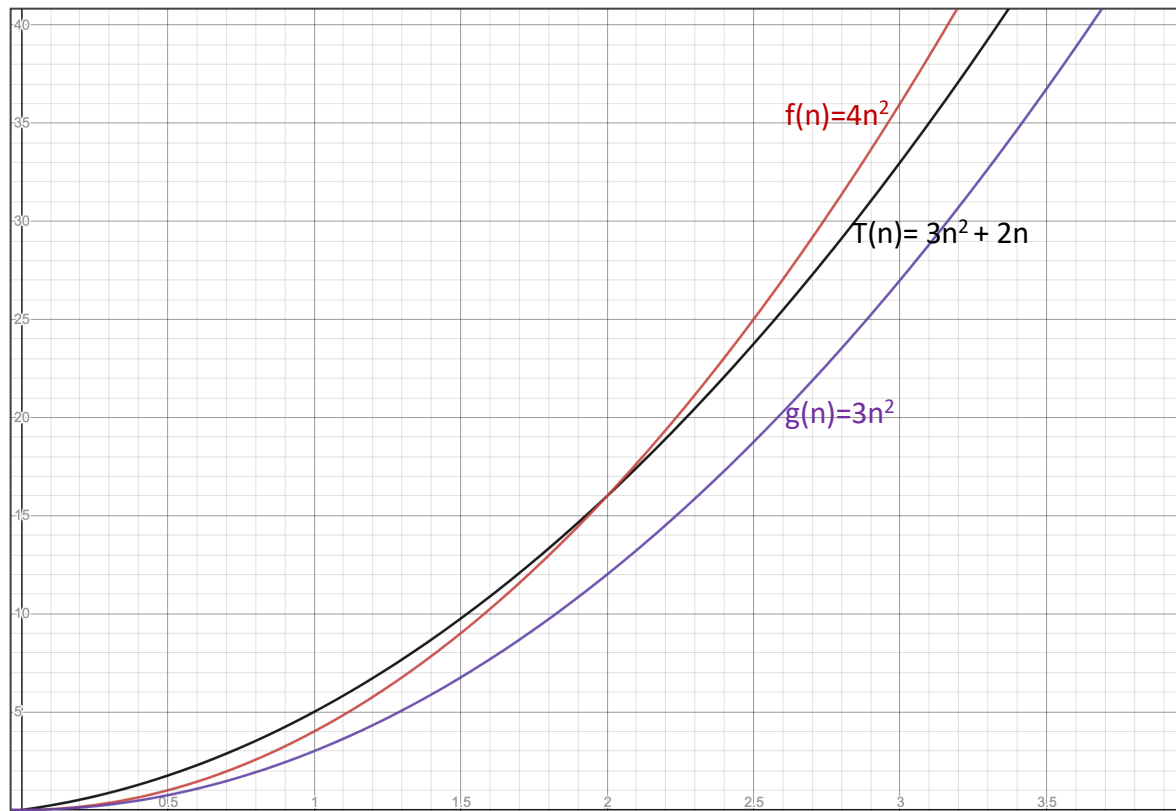
Limiti esatti. $T(n)$ è $\Theta(f(n))$ se $T(n)$ sia $O(f(n))$ che $\Omega(f(n))$.



Ordine asintotico di grandezza

Limiti esatti: $T(n)$ è $\Theta(f(n))$ se $T(n)$ sia $O(f(n))$ che $\Omega(f(n))$.

Esempio: abbiamo dimostrato che $3n^2 + 2n$ è sia $O(n^2)$ che $\Omega(n^2) \rightarrow 3n^2 + 2n$ è $\Theta(n^2)$



Ordine asintotico di grandezza

- Quando analizziamo un algoritmo miriamo a trovare stime asintotiche quanto più "strette" è possibile
- Dire che InsertionSort ha tempo di esecuzione $O(n^3)$ non è errato ma $O(n^3)$ non è un limite "stretto" in quanto si può dimostrare che InsertionSort ha tempo di esecuzione $O(n^2)$
- $O(n^2)$ è un limite stretto?
 - Sì, perché il numero di passi eseguiti da InsertionSort è an^2+bn+c , con $a>0$, che non solo è $O(n^2)$ ma è anche $\Omega(n^2)$.
 - Si può dire quindi che il tempo di esecuzione di InsertionSort è $\Theta(n^2)$

Errore comune

Affermazione priva di senso. Ogni algoritmo basato sui confronti richiede almeno $O(n \log n)$ confronti.

- Per i lower bound si usa Ω

Affermazione corretta. Ogni algoritmo basato sui confronti richiede almeno $\Omega(n \log n)$ confronti.

Proprietà

Transitività .

- Se $f = O(g)$ e $g = O(h)$ allora $f = O(h)$.
- Se $f = \Omega(g)$ e $g = \Omega(h)$ allora $f = \Omega(h)$.
- Se $f = \Theta(g)$ e $g = \Theta(h)$ allora $f = \Theta(h)$.

Additività .

- Se $f = O(h)$ e $g = O(h)$ allora $f + g = O(h)$.
- Se $f = \Omega(h)$ e $g = \Omega(h)$ allora $f + g = \Omega(h)$.
- Se $f = \Theta(h)$ e $g = \Theta(h)$ allora $f + g = \Theta(h)$.

Bound asintotici per alcune funzioni di uso comune

Polinomi. $a_0 + a_1n + \dots + a_d n^d$, con $a_d > 0$, è $\Theta(n^d)$.

Dim. $O(n^d)$: dobbiamo trovare due costanti $c > 0$ e $n_0 \geq 0$ tali che $a_0 + a_1n + \dots + a_d n^d \leq c n^d$ per ogni $n \geq n_0$

$$a_0 + a_1n + a_2 n^2 + \dots + a_d n^d$$

$$\leq |a_0| + |a_1|n + |a_2|n^2 + \dots + |a_d|n^d$$

$$\leq (|a_0| + |a_1| + |a_2| + \dots + |a_d|) n^d, \text{ per ogni } n \geq 1.$$


Basta quindi prendere $n_0=1$ e $c = |a_0| + |a_1| + \dots + |a_d|$ (è una costante)

Bound asintotici per alcune funzioni di uso comune

Dimostriamo come esercizio che $a_0 + a_1n + \dots + a_d n^d$ è anche $\Omega(n^d)$:

- $a_0 + a_1n + \dots + a_d n^d = a_d n^d + \dots + a_1n + a_0 \geq a_d n^d - (|a_0| + |a_1|n + \dots + |a_{d-1}|n^{d-1})$
- Abbiamo appena visto che un polinomio di grado d è $O(n^d)$
 - Ciò implica $|a_0| + |a_1|n + \dots + |a_{d-1}|n^{d-1} = O(n^{d-1})$
e di conseguenza esistono due costanti $n'_0 \geq 0$ e $c' > 0$
tali che $|a_0| + |a_1|n + \dots + |a_{d-1}|n^{d-1} \leq c'n^{d-1}$ per ogni $n \geq n'_0$
 - **In realta' possiamo prendere $n'_0 = 1$ come nella dim. precedente.**
- Quindi $a_d n^d - (|a_0| + |a_1|n + \dots + |a_{d-1}|n^{d-1}) \geq a_d n^d - c'n^{d-1}$ per ogni $n \geq n'_0$
- da cui $a_0 + a_1n + \dots + a_d n^d \geq a_d n^d - c'n^{d-1}$ per ogni $n \geq n'_0$

Bound asintotici per alcune funzioni di uso comune

-  Nella slide precedente abbiamo dimostrato che esistono due costanti $n'_0 \geq 0$ e $c' > 0$ tali che $a_0 + a_1n + \dots + a_d n^d \geq a_d n^d - c' n^{d-1}$ per ogni $n \geq n'_0 = 1$.
- Siccome $n'_0 = 1$ allora stiamo considerando solo valori di $n > 1$ e possiamo scrivere il secondo membro della disequazione come $(a_d - c'/n)n^d$
 - * Per dimostrare $a_0 + a_1n + \dots + a_d n^d = \Omega(n^d)$ dobbiamo trovare le costanti $n_0 \geq 0$ e $c > 0$ tali che $a_0 + a_1n + \dots + a_d n^d \geq cn^d$ per ogni $n \geq n_0$
 - Ovviamente se troviamo due costanti $n_0 \geq n'_0$ e $c > 0$ tali $(a_d - c'/n)n^d \geq cn^d$ per ogni $n \geq n_0$ allora vale anche la disuguaglianza in *
 - Risolviamo $(a_d - c'/n)n^d \geq cn^d$.
 - $(a_d - c'/n)n^d \geq cn^d \iff a_d - c'/n \geq c$ (vale \iff perche' considero gli $n \geq n'_0 = 1$)
 - Mi basta che c sia maggiore di 0 quindi posso semplicemente risolvere la disequazione $a_d - c'/n > 0$ rispetto ad n .
 - $a_d - c'/n > 0 \iff n > c'/a_d$. Ho finito: prendo $n_0 = \max\{1, 2c'/a_d\}$ e $c = a_d/2$ (ottenuto rimpiazzando n con $2c'/a_d$ in $a_d - c'/n$)

Ordine asintotico di grandezza

Esempio:

$$T(n) = 32n^2 + 17n + 32.$$

- $T(n)$ è $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$ e $\Theta(n^2)$.

- $T(n)$ **non** è $O(n)$, $\Omega(n^3)$, $\Theta(n)$ o $\Theta(n^3)$.

Tempo lineare: $O(n)$

Tempo lineare. Il tempo di esecuzione è al più un fattore costante per la dimensione dell'input.

Esempio:

Computazione del massimo. Computa il massimo di n numeri a_1, \dots, a_n .

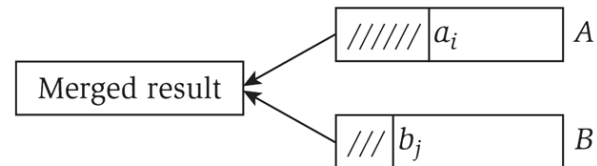
```
max ← a1
for i = 2 to n {
  Se (ai > max)
    max ← ai
}
```

Il problema dell'individuazione del max di n numeri è $\Omega(n)$

Dim. ogni numero diverso dal massimo deve partecipare ad almeno un confronto in cui risulta < dell'altro elemento → almeno un confronto per ciascuno degli $n-1$ elementi diversi dal massimo

Tempo lineare: $O(n)$

Merge. Combinare 2 sequenze ordinate $A = a_1, a_2, \dots, a_n$ with $B = b_1, b_2, \dots, b_m$ in una lista ordinata.



```
i = 1, j = 1
while (i ≤ n and j ≤ m) {
    if (a_i ≤ b_j) aggiungi a_i alla fine della lista output e incrementa i
    else aggiungi b_j alla fine della lista output e incrementa j
}
```

Ciclo per aggiungere alla lista output gli elementi non ancora esaminati di una delle due liste input

Affermazione. Fondere due sequenze ordinate rispettivamente di dimensione n ed m richiede tempo $O(n+m)$.

Dim. Dopo ogni iterazione del while o del ciclo sottostante, la lunghezza dell'output aumenta di 1.

Tempo quadratico: $O(n^2)$

Tempo quadratico. Tipicamente si ha quando un algoritmo esamina tutte le coppie di elementi input

Coppia di punti più vicina. Data una lista di n punti del piano $(x_1, y_1), \dots, (x_n, y_n)$, vogliamo trovare la coppia più vicina.

Soluzione $O(n^2)$. Calcola la distanza tra tutte le coppie di punti.

```
min ← (x1 - x2)2 + (y1 - y2)2
for i = 1 to n {
  for j = i+1 to n {
    d ← (xi - xj)2 + (yi - yj)2
    Se (d < min)
      min ← d
  }
}
```

← Per effettuare i confronti non c'è bisogno di estrarre la radice quadrata

Per esercizio, svolgiamo l'analisi dell'algoritmo nella slide precedente...

Il for esterno mi costa: **tempo lineare in n + il tempo per eseguire tutte le iterazioni del for interno**

Analizziamo il for interno:

Chiamiamo t_i il numero di iterazioni del for interno alla i -esima iterazione del for esterno

Quante volte viene iterato il for interno in totale? Risposta: $t_1+t_2+\dots+t_n$.

Per ogni i si ha $t_i=(n-i)$

Quindi sommando i t_i per tutte le iterazioni del for esterno ho

$$t_1+t_2+\dots+t_n = (n-1)+(n-2)+\dots+1+0 = n(n-1)/2 \text{ iterazioni del for interno } \mathbf{IN\ TOTALE}$$

siccome la singola esecuzione del corpo del for interno richiede tempo pari ad una costante c
→ il tempo richiesto da tutte le iterazioni del for interno è $c(n(n-1)/2)=\Theta(n^2)$

Tempo totale: $\Theta(n)+\Theta(n^2)=\Theta(n^2)$

Tempo cubico: $O(n^3)$

Tempo cubico. Tipicamente si ha quando un algoritmo esamina tutte le triple di elementi.

Esempio:

Disgiunzione di insiemi. Dati n insiemi S_1, \dots, S_n ciascuno dei quali è un sottoinsieme di $\{1, 2, \dots, n\}$, c'è qualche coppia di insiemi che è disgiunta?

Soluzione $O(n^3)$. Per ogni coppia di insiemi, determinare se i due insiemi sono disgiunti. (Supponiamo di poter determinare in tempo costante se un elemento appartiene ad un insieme)

```
for i = 1 to n { //corpo iterato n volte
  flag = true
  for j = i+1 to n { //corpo iterato n-i volte ad ogni iterazione del for esterno
    foreach elemento p di  $S_i$  { //corpo iterato al più n volte ad ogni iteraz. for su j
      if p appartiene anche a  $S_j$  //supponiamo test richiede ogni volta  $O(1)$ 
        flag = false; break;
    }
    If(flag = true) // nessun elemento di  $S_i$  appartiene a  $S_j$ 
      riporta che  $S_i$  e  $S_j$  sono disgiunti
  }
}
```

Un utile richiamo

Alcune utili proprietà dei logaritmi:

1. $\log_a x = (\log_b x) / (\log_b a)$
2. $\log_a(xy) = \log_a x + \log_a y$
3. $\log_a x^k = k \log_a x$

Dalla 1. discende:

4. $\log_a x = 1 / (\log_x a)$

Dalla 3. discende:

5. $\log_a(1/x) = -\log_a x$

Dalla 2. e dalla 5. discende:

6. $\log_a(x/y) = \log_a x - \log_a y$

Regole per la notazione asintotica

$$d(n) = O(f(n)) \Rightarrow ad(n) = O(f(n)), \quad \forall \text{ costante } a > 0$$

$$\text{Es.: } \log n = O(n) \Rightarrow 7 \log n = O(n)$$

$$d(n) = O(f(n)), e(n) = O(g(n)) \Rightarrow d(n) + e(n) = O(f(n) + g(n))$$

$$\text{Es.: } \log n = O(n), \sqrt{n} = O(n) \Rightarrow \log n + \sqrt{n} = O(n)$$

$$d(n) = O(f(n)), e(n) = O(g(n)) \Rightarrow d(n)e(n) = O(f(n)g(n))$$

$$\text{Es.: } \log n = O(\sqrt{n}), \sqrt{n} = O(\sqrt{n}) \Rightarrow \sqrt{n} \log n = O(n)$$

$$d(n) = O(f(n)), f(n) = O(g(n)) \Rightarrow d(n) = O(g(n))$$

$$\text{Es.: } \log n = O(\sqrt{n}), \sqrt{n} = O(n) \Rightarrow \log n = O(n)$$

$$f(n) = a_d n^d + \dots + a_1 n + a_0 \Rightarrow f(n) = O(n^d)$$

$$\text{Es.: } 5n^7 + 6n^4 + 3n^3 + 100 = O(n^7)$$

$$n^x = O(a^n), \quad \forall \text{ costanti } x > 0, a > 1 \quad \text{Es.: } n^{100} = O(2^n)$$

Regole per la notazione asintotica

- Le prime 5 regole nella slide precedente valgono anche se sostituiamo O con Ω o con Θ
- Dimostriamo per esercizio la 1.
- $d(n)=O(f(n)) \rightarrow ad(n)=O(f(n))$, per a costante positiva
- Dim.
- $d(n)=O(f(n)) \rightarrow$ esistono due costanti $c' > 0$ ed $n'_0 \geq 0$ t.c. $d(n) \leq c'f(n)$ per ogni $n \geq n'_0$
- Moltiplicando entrambi i membri della disuguaglianza per a il verso della disuguaglianza rimane invariato perche' $a > 0$.
- Quindi si ha $ad(n) \leq ac'f(n)$ per ogni $n \geq n'_0$.
- Abbiamo quindi trovato le costanti c ed n_0 per cui vale la definizione di $O(f(n))$
 - basta infatti porre $c=ac'$ ed $n_0 = n'_0$
 - NB: ac' e' una costante > 0 perche' sia a che c' sono costanti > 0 .