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 $\Theta(n^3)$.

35

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 $a_1, \ldots, a_n$.

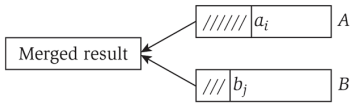
```
max ← a1
for i = 2 to n {
   Se (ai > max)
      max ← ai
}
```

Il problema dell'individuazione del max di n numeri e` Ω(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

36

Tempo lineare: $O(n)$

Merge. Combinare 2 sequenze ordinate $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_m$ 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.

37

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 $(x_1, y_1), \ldots, (x_n, y_n)$, vogliamo trovare la coppia più vicina.

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

```
min ← (x_1 - x_2)^2 + (y_1 - y_2)^2
for i = 1 to n {
   for j = i+1 to n {
      d ← (x_i - x_j)^2 + (y_i - y_j)^2
      Se (d < min)
         min ← d
   }
}
```

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

38

A lezione, per esercizio, abbiamo svolto 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: $t_1+t_2+…+t_n$.

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

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

$t_1+t_2+…+t_n = (n-1)+(n-2)+…+1+0 =n(n-1)/2$ iterazioni del for interno **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: $\Theta(n)+ \Theta(n^2)= \Theta(n^2)$

39

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 $S_1, …, S_n$ ciascuno dei quali è un sottoinsieme di {1, 2, …, 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)

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


40

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

41

Regole per la notazione asintotica

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

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

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

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

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

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

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

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

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

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

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

42

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.

Dimostriamo proprieta` transitiva

- $d(n)=O(f(n))$ e $f(n)=O(g(n)) \rightarrow d(n)=O(g(n))$
- Dim.

1. $d(n)=O(f(n)) \rightarrow$ esistono due costanti $c'>0$ ed $n'_0 \geq 0$ t.c. $d(n) \leq c'f(n)$ per ogni $n \geq n'_0$
2. $f(n)=O(g(n)) \rightarrow$ esistono due costanti $c''>0$ ed $n''_0 \geq 0$ t.c. $f(n) \leq c''g(n)$ per ogni $n \geq n''_0$

- la 1 $\rightarrow d(n) \leq c'f(n)$ per ogni $n \geq n'_0$, la 2 $\rightarrow f(n) \leq c''g(n)$ per ogni $n \geq n''_0$
- e di conseguenza, $d(n) \leq c'f(n) \leq c'(c''g(n))=c'c''g(n)$ per ogni n maggiore di n'_0 e n''_0
- Ponendo $c=c'c''$ ed $n_0=\max\{n'_0, n''_0\}$, possiamo quindi affermare che
- $d(n) \leq cg(n)$ per ogni $n \geq n_0$ e cio` implica $d(n)=O(g(n))$

Dimostriamo l'additivita`

- d(n)=O(f(n)) ed e(n)=O(g(n)) → d(n)+e(n)=O(f(n)+g(n))
- Dim.

1. d(n)=O(f(n)) →esistono due costanti c'>0 ed $n'_0 \geq 0$ t.c. d(n)≤c'f(n) per ogni $n \geq n'_0$
2. e(n)=O(g(n)) →esistono due costanti c''>0 ed $n''_0 \geq 0$ t.c. e(n)≤c''g(n) per ogni $n \geq n''_0$

- la 1 → d(n) ≤c'f(n) per ogni $n \geq n'_0$, la 2 → e(n) ≤c''g(n) per ogni $n \geq n''_0$
- e di conseguenza, d(n)+e(n)≤c'f(n) +c''g(n) ≤ max{c',c''}f(n) + max{c',c''}g(n) = max{c',c''} (f(n)+g(n)) per ogni n maggiore di n'_0 e n''_0
- Ponendo c= max{c',c''} ed n_0=max{n'_0, n''_0}, possiamo quindi affermare che
- d(n)+e(n) ≤c(f(n)+g(n)) per ogni $n \geq n_0$ e cio` implica d(n)+e(n)=O(f(n)+g(n))

Bound asintotici per alcune funzioni di uso comune

Logaritmi

- $O(\log_a n) = O(\log_b n)$, $\Omega(\log_a n) = \Omega(\log_b n)$, $\Theta(\log_a n) = \Theta(\log_b n)$, per ogni costante a, b > 0.

Dim. per O (per le altre notazioni asintotiche le dimostrazioni sono simili)

dalla proprieta` 1 dei logaritmi si ha, $\log_a n = \log_b n/(\log_b a)$ (*)

- siccome banalmente $\log_b n = O(\log_b n)$ allora, per la regola 1 della notazione asintotica (slide 39), si ha $\log_b n/(\log_b a)= O(\log_b n)$
- siccome dalla (*) $\log_a n = \log_b n/(\log_b a)$ allora $\log_a n = O(\log_b n)$

analogamente possiamo dimostrare che $\log_b n = O(\log_a n)$