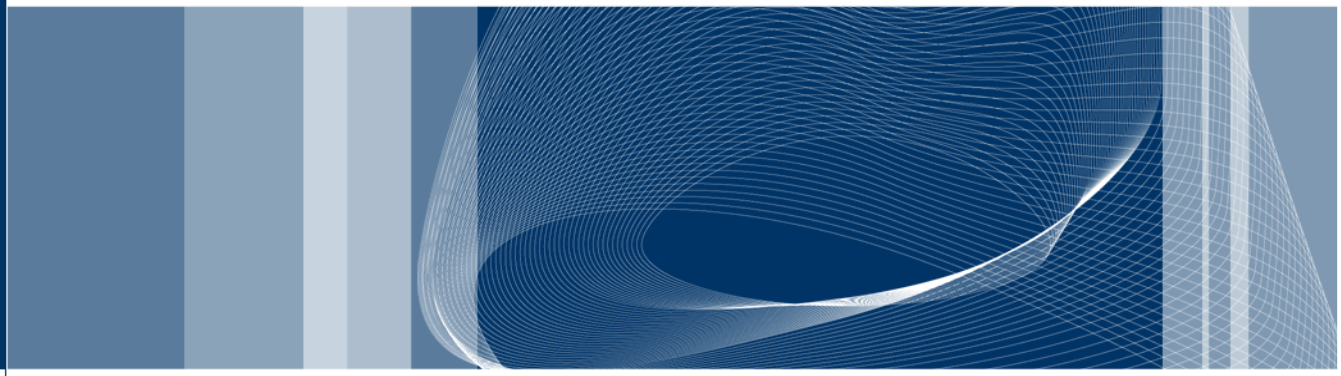
POLITECNICO DI MILANO



Esercizi su analisi della complessità ed algoritmi di ordinamento

Algoritmi, StruttureDatie CalcoloParallelo

Daniele Loiacono

Ripasso Master Theorem

- Si applica a forme ricorsive del tipo,

  $T(n) = aT(n/b) + f(n)$

  dove $a \geq 1$, $b > 1$, ed f è asintoticamente positiva

- Si confronta f(n) con $n^{\log_b a}$

- Tre casi possibili
  - f(n) cresce più lentamente di $n^{\log_b a}$
  - f(n) cresce in maniera simile a $n^{\log_b a}$
  - f(n) cresce più velocemente di $n^{\log_b a}$

Ripasso Master Theorem

- Caso 1: f(n) cresce più lentamente di $n^{\log_b a}$
  - $f(n) = O(n^{\log_b a - \varepsilon})$ per una costante $\varepsilon > 0$
  - Soluzione: $T(n) = \Theta(n^{\log_b a})$
- Caso 2: f(n) cresce in maniera simile a $n^{\log_b a}$
  - $f(n) = Q(n^{\log_b a} \lg^k n)$ per una costante $k \geq 0$
  - Soluzione: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
- Caso 3: f(n) cresce più velocemente di $n^{\log_b a}$
  - $f(n) = \Omega(n^{\log_b a + \varepsilon})$ per una costante $\varepsilon > 0$
  - f(n) cresce più velocemente di $n^{\log_b a}$ (di un fattore n^{ε}), ed f(n) soddisfa la condizione $af(n/b) \leq cf(n)$ per una costante $c < 1$
  - Soluzione: $T(n) = \Theta(f(n))$

Esercizio 1

- Si consideri la seguente relazione di ricorrenza definita per n=2d con d≥0

$$\begin{cases} T(n) = 4T\left(\frac{n}{2}\right) + 1 \\ T(1) = 1 \end{cases}$$

1. Applicare il master theorem del divide et impera per avere una stima asintotica di *T(n)*
2. Risolvere in modo esatto la relazione di ricorrenza usando il metodo iterativo
3. Verificare per induzione la soluzione trovata al punto precedente

Esercizio 1: Soluzione (1)

- Siamo nel primo caso del master theorem
- poichè a = 4, b = 2 e f(n) = 1.

Quindi T(n) è $\Theta(n^{\log_2(4)}) = \Theta(n^2)$

Esercizio 1: Soluzione (2)

- Espandendo la ricorsione:

$$T(n) = 4T\left(n/2\right)+1 = 1+4+16T\left(n/4\right) =$$
$$= 1+4+16+64T(n/8) =$$
$$= \sum_{i=0}^{d-1} 4^i + 4^d T(n/2^d)$$

- La ricorsione termina quando d = $\log_2 n$

$$T(n) = \sum_{i=0}^{\log_2 n} 4^i = \frac{4^{\log_2(n)+1}-1}{3} = \frac{2^{2\log_2(n)+2}-1}{3}$$
$$= \frac{4\cdot 2^{\log_2(n^2)}-1}{3} = \frac{4n^2-1}{3} = \Theta(n^2)$$

Esercizio 1: Soluzione (3)

- Base:

$$T(1) = \frac{4-1}{3} = 1 \quad [\text{OK}]$$

- Induzione:

$$\begin{cases} T(n) = 4T(n/2) + 1 \\ T(n/2) = \dfrac{4(n/2)^2 - 1}{3} = \dfrac{n^2 - 1}{3} \end{cases}$$

$$\Rightarrow T(n) = 4T(n/2) + 1 = \frac{4n^2 - 4}{3} + 1 = \frac{4n^2 - 1}{3} \quad [\text{c. v. d.}]$$

Ripasso regole utili

1. If $f(n)$ is in $O(g(n))$ and $g(n)$ is in $O(h(n))$, then $f(n)$ is in $O(h(n))$. [Transit.]
2. If $f(n)$ is in $O(kg(n))$ for any constant $k > 0$, then $f(n)$ is in $O(g(n))$. [No constant]
3. If $f1(n)$ is in $O(g1(n))$ and $f2(n)$ is in $O(g2(n))$, then $(f1 + f2)(n)$ is in $O(max(g1(n), g2(n)))$. [Drop low order terms]
4. If $f1(n)$ is in $O(g1(n))$ and $f2(n)$ is in $O(g2(n))$ then $f1(n)f2(n)$ is in $O(g1(n)g2(n))$. [Loops]

Queste regole valgono per O, Ω e Θ.

Drop Low Order Terms

- Possono essere eliminati i termini di ordine inferiore.
- Ricordiamo che asintoticamente:

an > nb > lognc con a,b,c cost.

- Esempi:

$$O(a2^n + bn^6) \rightarrow O(a2^n) \rightarrow O(2^n)$$

$$O(an + b\log n) \rightarrow O(an) \rightarrow O(n)$$

Loops

- La proprietà dei loop si può applicare solo se i loop sono indipendenti fra di loro.
- Non sempre è così:

```
for(i=1; i<=n; i++)
      for(j=1; j<=i; j++)
            sum++;
```

- Il ciclo interno viene eseguito un numero di volte che dipende da quello esterno: f2(i) = ci
- Quello esterno esegue n volte il ciclo interno:

$$\Rightarrow T(n) = \sum_{i=1}^{n} ic = c\frac{n(n+1)}{2} = \Theta(n^2)$$

Loops (2)

- Non sempre i cicli hanno incremento unitario:

```
for (k=1; k<=n; k*=2)
    for (j=1; j<=k; j++)
        sum2++;
```

- k vale 2^i con i che varia da 0 a $\log_2 n$

$$T(N) = \sum_{i=0}^{\log_2 n} c \cdot 2^i = c\frac{1-2^{\log_2(n)+1}}{1-2} = c(2 \cdot 2^{\log_2 n} - 1) =$$
$$= c(2n-1) = \Theta(n)$$

Esercizio 2

- Si consideri il seguente frammento di codice in C:

```
int a[n], b[n];
j = 1; i = 2;
while (i <= n) {
b[j] = a[i];
    j++;
    i = i * i;
}
```

- Si determini l'ordine di grandezza Θ della complessità temporale di tale frammento in funzione di n.

Esercizio 2: Soluzione

- Si considerino i valori assunti da i nelle prime iterazioni:

$$2, 4, 16, 256, \ldots$$

- A prima vista non è immediato ricondurli ad una potenza di un qualche numero k. Proviamo però a riscriverli così:

$$2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{2^3}, \ldots$$

- A questo punto risulta evidente che i cresce come 2^{2^k}

Esercizio 2: Soluzione (2)

- k varia tra 0 (quando i=2) e log(logn) (quando i=n), la complessità sarà dunque:

$$\Rightarrow T(n) = c(\log(\log n) + 1) =$$
$$= \Theta(\log(\log n))$$

Esercizio 3

- Si considerino le funzioni F1, F2, F3, ed F4 riportate sotto:

```
void F1(Elem[] array) {
  for (int i=0; i<array.length-1; i++)
  for (int k=0; k<array.length-2; k++)
  if (array[k]>array[k+1])
        scambia(array, k, k+1);
}
```

```
void F3(Elem[] array) {
  swap=true;
  while (swap) {
    swap=false;
    for (int k=0; k<array.length-1; k++)
       if (array[k]>array[k+1]) {
          scambia(k, k+1);
          swap=true;}
}
```

```
void F2(Elem[] array) {
  for (int i=0; i<array.length-1; i++)
  for (int k=array.length-2; k>=i; k--)
  if (array[k]<array[k+1])
        scambia(array, k, k+1);
}
```

```
void F4(Elem[] array) {
  for (int i=array.length-1; i>=0 i--)
  for (int k=array.length-2; k>=0; k--)
      if (array[k]>array[k+1])
        scambia(k, k+1)
}
```

Esercizio 3

- Riempire la seguente tabella con la complessità delle funzioni:

Caso	F1	F2	F3	F4
Ottimo				
Pessimo				
Medio				

Esercizio 3: Soluzione

- Per determinare la complessità devo valutare quante volte viene eseguito il ciclo.
- Fatta eccezione per f3, non dipende dalla disposizione iniziale degli elementi
- Caso medio, pessimo e ottimo coincidono:
  - f1: (n-1)(n-2)
  - f2: $\sum_{i=0}^{n-1} i = n(n-1)/2$
  - f4: n(n-1)

Esercizio 3: Soluzione

- f3 il ciclo viene eseguito un numero di volte che dipende dal numero di inversioni presenti nell'array.
- Nel caso pessimo sarà circa n^2
- Nel caso medio sarà circa $n^2/2$
- Nel caso ottimo (vettore completamente ordinato) il ciclo sarà eseguito solo n-1 volte (è una sola esecuzione completa del ciclo per verificare che l'array è già ordinato)

Esercizio 3: Soluzione

- In sintesi:

	bubsort1	bubsort2	bubsort3	bubsort4
Ottimo	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Pessimo	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Medio	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

Esercizio 4

- Si ipotizzi che esista un algoritmo SPLITk che può dividere una lista L di n elementi in k sottoliste, ognuna delle quali contiene uno o più elementi, tale che tutti gli elementi della sottolista i sono minori di tutti gli elementi della sottolista j, per i<j<=k. Se n<k, allora k-n delle sottoliste prodotte da SPLITk sono vuote. Si assuma che SPLITk abbia complessità Θ(n), n essendo la lunghezza della lista che viene divisa in k sottoliste.
- Si assuma inoltre che le k liste possano essere concatenate in un tempo costante.

Esercizio 4

- Si consideri ora il seguente algoritmo.

```
List SORTk (List L) {
      List sub[k]; // contiene le sottoliste
      if (L.length()>1) {
            SPLITk(L, sub); // SPLITk pone le
                            //sottoliste in sub
            for (int i = 0; i <k; i++)
                  sub[i] = SORTk(sub[i]);

            L = concatenazione delle k sottoliste
                contenute in sub
      }
      return L;
}
```

Esercizio 4

- Rispondere ai seguenti quesiti:

1. Qual è la complessità asintotica di SORTk nel caso ottimo? (giustificare opportunamente la risposta)

2. Qual è la complessità asintotica di SORTk nel caso pessimo? (giustificare opportunamente la risposta)

Esercizio 4: Soluzione

- Il caso ottimo si ha nel caso in cui le k sottoliste sono della stessa dimensione

$$\begin{cases} T(n) = kT\left(\frac{n}{k}\right) + cn + d \\ T(1) = e \end{cases}$$

$$T(n) = kT(n/k) + cn + d = cn + k(kT(n/k^2) + cn/k + d) + d =$$

$$= 2cn + k^2(kT(n/k^3) + cn/k^2 + d) + d(1+k) = 3cn + k^3T(n/k^3) +$$

$$+ d(1+k+k^2) = \ldots = hcn + k^hT(n/k^h) + d\sum_{i=0}^{h-1} k^i$$

La ricorsione termina quando $n/k^h = 1 \Rightarrow h = \log n$ e sia $d\sum_{i=0}^{h-1} k^i = \mathrm{D}$

$$T(n) = cn\log n + k^{\log n}T(1) + D = cn\log n + ne + D \Rightarrow T(N) = \Theta(n\log n)$$

Esercizio 4: Soluzione

- Il caso pessimo si ha nel caso in cui le k sottoliste sono completamente sbilanciate

$$\begin{cases} T(n) = T\left(n-k+1\right) + (k-1)T(1) + cn + d \\ T(1) = e \end{cases} \Rightarrow T(n) = T\left(n-k+1\right) + cn + D$$

$$T(n) = cn + D + (cn + D + T(n-2k+2)) = 3(cn+D) + T(n-3k+3) =$$

$$= \ldots = h(cn+D) + T(n-hk+h)$$

La ricorsione termina quando $n - hk + h = 1 \Rightarrow h = \frac{n-1}{k-1}$

$$T(n) = \frac{n-1}{k-1}(cn+D) + T(1) = c\frac{n(n-1)}{k-1} + D\frac{n-1}{k-1} + e$$

$$\Rightarrow T(N) = \Theta(n^2)$$

Esercizio 5

- Dare dei bound asintotici per le seguenti ricorrenze, giustificandoli

  - $T(n) = 7\,T(n/3) + n^2$
  - $T(n) = 2\,T(n/4) + n \log n$
  - $T(n) = T(n-3) + \log n$

Esercizio 5: Soluzione

- $T(n) = 7\,T(n/3) + n^2$
  - $f(n) = n^2$ cresce più velocemente di $n^{\log_3 7}$
  - Soluzione: $T(n) = \Theta(n^2)$
- $T(n) = 2\,T(n/4) + n \log n$
  - $f(n) = n \lg n$ cresce più velocemente di $n^{\log_4 2}$
  - Soluzione: $T(n) = \Theta(n \log n)$
- $T(n) = T(n-3) + \log n$
  - Upper Bound: $S(n) = T(n-1) + \log n = \log 1 + \log 2 + \ldots + \log n = \log(n!) \rightarrow \Theta(n \log n)$
  - Lower Bound: $T(n) = \log n + \log (n-3) + \log(n-6) + \ldots + \log 1 = \sum \log 3i \;\; (i=1\ldots n/3) >= \sum \log i \;\; (i=1\ldots n/3) = \log((n/3)!) \rightarrow \Theta(n \log n)$
  - Soluzione: $\Theta(n \log n)$

Esercizio 6

- Calcolare la complessità asintotica di T(n) in ognuna di queste ricorrenze assumendo che T(n) sia costante per n sufficientemente piccolo (giustificare opportunamente le risposte).
  - $T(n) = 3T(n/2) + n\log n$.
  - $T(n) = T(n - 1) + 1/n$.
  - $T(n) = 2T(n/2) + n^3$.
  - $T(n) = 16T(n/4) + n^2$.
  - $T(n) = 7T(n/2) + n^2$
  - $T(n) = T(n - 1) + n$

Esercizio 6: Soluzione

- $T(n) = 3T(n/2) + n\log n$
  - nlogn più lento di n^$\log_2 3$ → Θ(n^ $\log_2 3$)
- $T(n) = T(n - 1) + 1/n$
  - T(n) = Σ 1/i (i=1..n) → T(n) = Θ(1)
- $T(n) = 2T(n/2) + n^3$
  - n^3 più veloce di n^$\log_2 2$ → T(n) = Θ(n^3)
- $T(n) = 16T(n/4) + n^2$
  - n^2 come n^$\log_4 16$ → T(n) = Θ(n^2logn)
- $T(n) = 7T(n/2) + n^2$
  - n^2 più veloce di n^$\log_2 7$ → T(n) = Θ(n^2)
- $T(n) = T(n - 1) + n$
  - T(n) = 1+2+ … + n = n(n+1)/2 → T(n) = Θ(n^2)