

# Algoritmi e Strutture Dati

## Algoritmi Ricorsivi e Ricorrenze

Maria Rita Di Berardini, Emanuela Merelli<sup>1</sup>

<sup>1</sup>Dipartimento di Matematica e Informatica  
Università di Camerino

15 ottobre 2009

# L'isola dei conigli

Leonardo da Pisa (noto anche come Fibonacci) si interessò di molte cose, tra cui il seguente problema di dinamica delle popolazioni:

Quanto velocemente si espanderebbe una popolazione di conigli sotto appropriate condizioni?

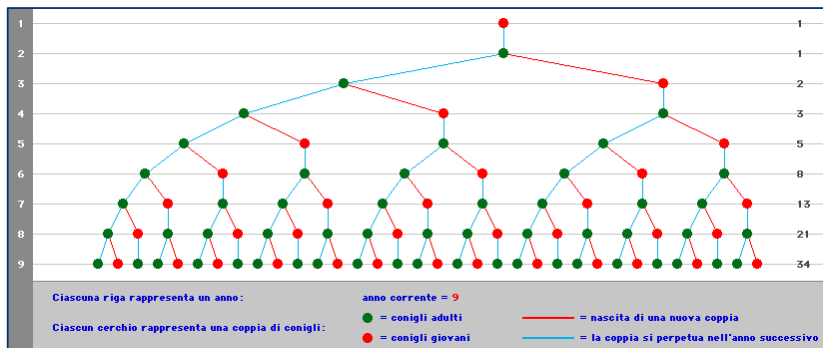
In particolare, partendo da una coppia di conigli in un'isola deserta, quante coppie si avrebbero nell'anno  $n$ ?

# Le regole di riproduzione

- Una coppia di conigli genera due coniglietti ogni anno
- I conigli cominciano a riprodursi soltanto al secondo anno dopo la loro nascita
- I conigli sono immortali

# L'albero dei conigli

La riproduzione dei conigli può essere descritta in un albero come segue:



# La regola di espansione

Nell'anno  $n$ , ci sono tutte le coppie dell'anno precedente, più una nuova coppia di conigli per ogni coppia presente due anni prima

Indicando con  $F_n$  il numero di coppie dell'anno  $n$ , abbiamo la seguente [relazione di ricorrenza](#):

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

# Il problema

Come calcoliamo  $F(n)$ ?

# Un approccio numerico

Possiamo usare una funzione matematica che calcoli direttamente i numeri di Fibonacci

Si può dimostrare che:

$$F(n) = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$$

dove

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

e

$$\hat{\phi} = \frac{1 - \sqrt{5}}{2} \approx -0.618$$

# Algoritmo fibonacci1

**algoritmo** fibonacci1(intero n)  $\rightarrow$  intero

**return**  $\frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$

# L'algoritmo fibonacci1 è corretto?

**Problema:** qual è l'accuratezza su  $\phi$  e  $\hat{\phi}$  per ottenere un risultato corretto?

Ad esempio con tre cifre decimali:  $\phi \approx 1.618$  e  $\hat{\phi} \approx -0.618$

| n  | fibonacci1 | arrotondamento | $F(n)$ |
|----|------------|----------------|--------|
| 3  | 1.99992    | 2              | 2      |
| 16 | 986.698    | 987            | 987    |
| 18 | 2583.1     | 2583           | 2584   |

# L'algoritmo fibonacc12

L'algoritmo `fibonacci1` non è corretto; un possibile approccio alternativo consiste nell'utilizzare un algoritmo ricorsivo:

```
algoritmo fibonacci2(intero n) → intero  
    if ( $n \leq 2$ ) return 1  
    else return fibonacci2( $n - 1$ ) +  
                  fibonacci2( $n - 2$ )
```

Opera solo su numeri interi

# L'algoritmo fibonacc12

Il costo di esecuzione di `fibonacc12` è espresso dalla seguente funzione ricorsiva (equazione di ricorrenza o semplicemente ricorrenza)

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{se } n \geq 3 \\ c & \text{se } n = 1, 2 \end{cases}$$

vi ricorda qualcosa ??

# L'algoritmo fibonacc12

Il costo di esecuzione di `fibonacci2` è espresso dalla seguente funzione ricorsiva (equazione di ricorrenza o semplicemente ricorrenza)

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{se } n \geq 3 \\ c & \text{se } n = 1, 2 \end{cases}$$

vi ricorda qualcosa ??

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

# L'algoritmo fibonacc12

Il costo di esecuzione di `fibonacc12` è espresso dalla seguente funzione ricorsiva (equazione di ricorrenza o semplicemente ricorrenza)

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{se } n \geq 3 \\ c & \text{se } n = 1, 2 \end{cases}$$

vi ricorda qualcosa ??

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

Possiamo dimostrare che

$$T(n) = cF(n) = c \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$$

# L'algoritmo fibonacc12

In realtà fibonacc12 è un algoritmo molto lento:  $T(n) = O(2^n)$

## Dimostrazione.

Dalla definizione di  $O$ , dobbiamo dimostrare che esistono delle costanti positive  $c_0$  ed  $n_0$  tali che  $T(n) \leq c_0 2^n$  per ogni  $n \geq n_0$ . La prova è per induzione su  $n$  □

# Induzione

**Problema:** dimostrare che una affermazione  $Aff(n)$  è vera per ogni  $n \geq 0$

## Theorem (induzione 1)

Se

- 1  $Aff(0)$  è vera;
- 2  $Aff(n - 1)$  vera implica  $Aff(n)$  vera;

allora  $Aff(n)$  è vera per ogni  $n \geq 0$

## Theorem (induzione 2)

Se

- 1  $Aff(0)$  è vera;
- 2  $Aff(m)$  vera per ogni  $m < n$  implica  $Aff(n)$  vera;

allora  $Aff(n)$  è vera per ogni  $n \geq 0$

# Dimostrazione per induzione: un esempio

Proviamo a dimostrare per induzione su  $n$  che

$$\text{Aff}(n) ::= \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

**Caso Base** ( $i = 1$ ):

$$\sum_{i=1}^1 i = 1 = \frac{1(2)}{2} = \frac{1(1+1)}{2} \quad \text{OK}$$

**Passo induttivo:** Assumiamo  $\text{Aff}(n-1)$  vera, cioè

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

e dimostriamo che, allora, anche  $\text{Aff}(n)$  è vera

# Dimostrazione per induzione: un esempio

$$\begin{aligned}\sum_{i=1}^n i &= \sum_{i=1}^{n-1} i + n = \frac{(n-1)n}{2} + n = \frac{(n-1)n + 2n}{2} \\ &= \frac{n(n-1+2)}{2} = \frac{n(n+1)}{2} \quad \text{OK}\end{aligned}$$

# L'algoritmo fibonacc12

$$T(n) = O(2^n).$$

Dalla definizione di  $O$ , dobbiamo dimostrare che esistono delle costanti positive  $c_0$  ed  $n_0$  tali che  $T(n) \leq c_0 2^n$  per ogni  $n \geq n_0$ .

## Caso base

- $n = 1$ :  $T(1) = c \leq c_0 2$  basta scegliere  $c_0 \geq c/2$

**Passo induttivo:** assumiamo  $T(m) \leq c 2^m$  per ogni  $m \leq n$ . Allora:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) \\ &\leq c_0 2^{n-1} + c_0 2^{n-2} \\ &= c_0 2^{n-2} (2 + 1) \\ &\leq c_0 2^{n-2} 4 \\ &= c_0 2^n \end{aligned}$$



# Ricerca in un'insieme non ordinato

**ricercaSequenziale**(array A, elem x)

```
for i ← 1 to lenght[A]
  do if A[i] = x return trovato
return non trovato
```

$$T_{\text{best}}(n) = 1$$

$$T_{\text{worst}}(n) = n$$

$$T_{\text{avarange}}(n) = (n + 1)/2$$

x è in prima posizione

x è in ultima posizione oppure non è in L  
sotto un assunzione di equi-distribuzione  
delle istanze

# Ricerca in un'insieme ordinato

**RicercaBinaria**(array A, elem x)

$n \leftarrow$  Lunghezza di A

**if**  $n=0$  **return** non trovato

$i \leftarrow \lceil n/2 \rceil$

**if**  $A[i] = x$  **return** trovato

**else if**  $A[i] > x$  RicercaBinaria( $A[1; i - 1]$ , x)

**else** RicercaBinaria( $A[i + 1; n]$ , x)

# Ricerca in un'insieme ordinato

**RicercaBinaria**(array A, elem x)

$n \leftarrow$  Lunghezza di A

**if**  $n=0$  **return** non trovato

$i \leftarrow \lceil n/2 \rceil$

**if**  $A[i] = x$  **return** trovato

**else if**  $A[i] > x$  RicercaBinaria( $A[1; i - 1]$ , x)

**else** RicercaBinaria( $A[i + 1; n]$ , x)

Come analizziamo questo algoritmo?

# Tempo di esecuzione di RicercaBinaria

È descritto mediante la seguente **equazione di ricorrenza** :

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

# Cosa è una relazione di ricorrenza

Una **relazione di ricorrenza** - o più semplicemente **ricorrenza** - è una equazione che descrive una funzione in termini del suo valore con input più piccoli

Esistono tre grandi metodi per risolvere le ricorrenze - ovvero per ottenere dei limiti asintotici " $\Theta$ " o " $O$ "

- Il metodo di sostituzione
- Il metodo iterativo – metodo della albero di ricorsione
- Il metodo dell'esperto che consente di calcolare i limiti per ricorrenze della forma  $T(n) = aT(n/b) + f(n)$  dove  $a \geq 1$ ,  $b > 0$  ed  $f(n)$  è una funzione data

# Ricorrenze: un caso semplice

**Metodo iterativo:** consiste nello srotolare la ricorsione fino ad ottenere una sommatoria dipendente da  $n$

**Esempio:**

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

Proviamo ad applicare il metodo iterativo a  $T(n)$

$$\begin{aligned} T(n) &= 1 + T(n/2) \\ &= 1 + 1 + T(n/4) \\ &= 1 + 1 + 1 + T(n/8) \\ &= 1 + 1 + 1 + 1 + T(n/16) \\ &= \dots \end{aligned}$$

# Ricorrenze: un caso semplice

$$\begin{aligned}T(n) &= 1 + T(n/2) \\&= 1 + 1 + T(n/4) \\&= 1 + 1 + 1 + T(n/8) \\&= 1 + 1 + 1 + 1 + T(n/16) \\&= \dots \\&= k + T(n/2^k)\end{aligned}$$

Continuiamo a srotolare la ricorsione fin quando  $n/2^k = 1$ ; ora  $n/2^k = 1$  implica  $2^k = n$  e quindi  $k = \log_2 n$  ( $k$  è il logaritmo in base 2 di  $n$ ). Allora:

$$T(n) = \log_2 n + 1 = \Theta(\log_2 n)$$

# Metodo iterativo: un altro esempio (meno semplice)

Il metodo iterativo può essere applicato a qualsiasi ricorrenza, ma a volte può essere di difficile soluzione

**Esempio:**

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

Proviamo ad applicare il metodo iterativo a  $T(n)$

$$\begin{aligned} T(n) &= n + T(n/2) \\ &= n + n/2 + T(n/4) \\ &= n + n/2 + n/4 + T(n/8) \\ &= n + n/2 + n/4 + n/8 + T(n/16) \\ &= \dots \end{aligned}$$

# Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned}T(n) &= n + T(n/2) \\&= n + n/2 + T(n/4) \\&= n + n/2 + n/4 + T(n/8) \\&= n + n/2 + n/2^2 + n/2^3 + T(n/16) \\&= \dots \\&= n + n/2 + n/2^2 + n/2^3 + \dots + n/2^{k-1} + T(n/2^k) \\&= \sum_{i=0}^{k-1} n/2^i + T(n/2^k)\end{aligned}$$

Di nuovo ci fermiamo  $k = \log_2 n$ . Allora:

$$T(n) = \sum_{i=0}^{\log_2 n - 1} n/2^i + 1 = n \sum_{i=0}^{\log_2 n - 1} (1/2)^i + 1$$

# Metodo iterativo: un altro esempio (meno semplice)

$$T(n) = \sum_{i=0}^{\log_2 n - 1} n/2^i + 1 = n \sum_{i=0}^{\log_2 n - 1} (1/2)^i + 1$$

Fact

$$\sum_{i=0}^m \alpha^i = \frac{1 - \alpha^{m+1}}{1 - \alpha}$$

$$T(n) = n \sum_{i=0}^{\log_2 n - 1} (1/2)^i + 1 = n \frac{1 - (1/2)^{\log_2 n}}{1 - (1/2)} + 1$$

# Metodo iterativo: un altro esempio (meno semplice)

$$T(n) = n \frac{1 - (1/2)^{\log_2 n}}{1 - (1/2)} + 1$$

# Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned} T(n) &= n \frac{1 - (1/2)^{\log_2 n}}{1 - (1/2)} + 1 \\ &= n \frac{1 - (1/2)^{\log_2 n}}{1/2} + 1 \end{aligned}$$

# Metodo iterativo: un altro esempio (meno semplice)

$$T(n) = n \frac{1 - (1/2)^{\log_2 n}}{1 - (1/2)} + 1$$

$$= n \frac{1 - (1/2)^{\log_2 n}}{1/2} + 1$$

$$= 2n (1 - (1/2)^{\log_2 n}) + 1$$

# Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned}T(n) &= n \frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\&= n \frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\&= 2n (1 - (1/2)^{\log_2 n}) + 1 \\&= 2n (1 - (1/2^{\log_2 n})) + 1\end{aligned}$$

# Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned}T(n) &= n \frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\&= n \frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\&= 2n (1 - (1/2)^{\log_2 n}) + 1 \\&= 2n (1 - (1/2^{\log_2 n})) + 1 \\&= 2n (1 - 1/n) + 1\end{aligned}$$

# Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned}T(n) &= n \frac{1-(1/2)^{\log_2 n}}{1-(1/2)} + 1 \\&= n \frac{1-(1/2)^{\log_2 n}}{1/2} + 1 \\&= 2n (1 - (1/2)^{\log_2 n}) + 1 \\&= 2n (1 - (1/2^{\log_2 n})) + 1 \\&= 2n (1 - 1/n) + 1 \\&= 2n ((n-1)/n) + 1\end{aligned}$$

# Metodo iterativo: un altro esempio (meno semplice)

$$\begin{aligned}T(n) &= n \frac{1 - (1/2)^{\log_2 n}}{1 - (1/2)} + 1 \\&= n \frac{1 - (1/2)^{\log_2 n}}{1/2} + 1 \\&= 2n (1 - (1/2)^{\log_2 n}) + 1 \\&= 2n (1 - (1/2^{\log_2 n})) + 1 \\&= 2n (1 - 1/n) + 1 \\&= 2n ((n-1)/n) + 1 \\&= 2(n-1) + 1 = 2n - 1 = \Theta(n)\end{aligned}$$

# Alberi di ricorsione

Vengono usati in alternativa al metodo iterativo

Un albero di ricorsione è un albero in cui ogni nodo rappresenta il costo di un sottoproblema da qualche parte nell'insieme delle chiamate ricorsive

Consideriamo la ricorrenza

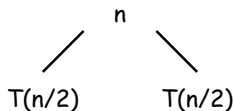
$$T(n) = \begin{cases} n + 2T(n/2) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

e costruiamo la derivazione dell'albero delle ricorrenze per  $T(n)$

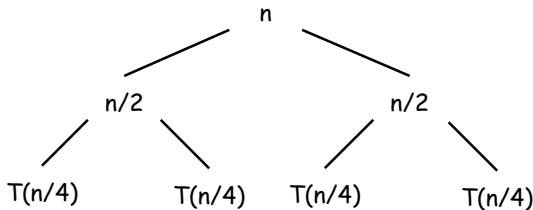
# Alberi di ricorsione: un esempio

$T(n)$

(a)

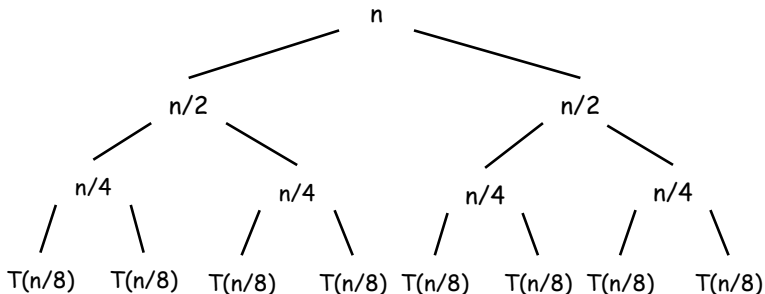


(b)



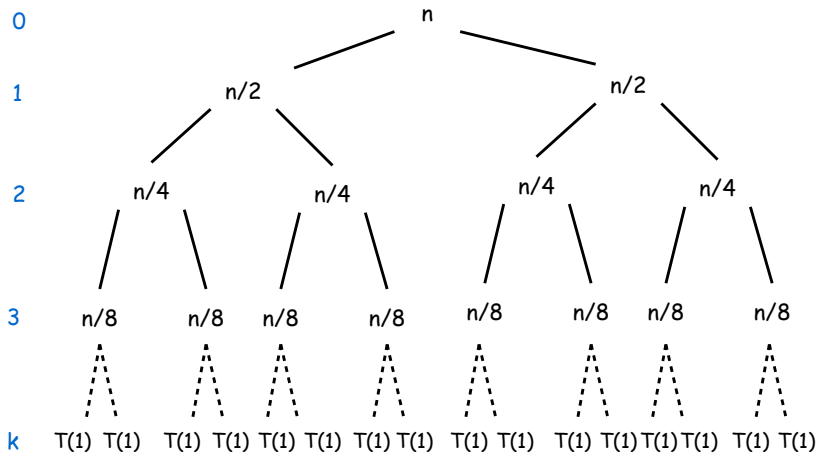
(c)

# Alberi di ricorsione: un esempio



(d)

# Alberi di ricorsione



# Alberi di ricorsione

Sia  $i$  un dato livello dell'albero di ricorsione ( $i = 0, \dots, \#livelli$ ) (al momento, non meglio specificato)

- **Quanti nodi ci sono al livello  $i$ ?** esattamente  $2^i$  nodi (al livello 0 abbiamo  $1 = 2^0$  nodi, ed ogni livello ha un numero di nodi doppio rispetto al livello precedente)
- **Quale è il costo di un nodo al livello  $i$ ?** ogni nodo al livello  $i$  costa  $n/2^i$  nodi (l'unico nodo al livello 0 ha un costo pari ad  $n = n/1 = n/2^0$ , ed ogni volta che scendiamo di livello il costo dei nodi si dimezza)
- il costo complessivo dei nodi al livello  $i$  è

$$\#nodi(i) \times costo\_nodo(i) = 2^i \times n/2^i = n$$

# Alberi di ricorsione

Quale è il costo complessivo della chiamata  $T(n)$ ?

$$T(n) = \sum_{j=0}^{\#livelli} costo\_livello(j) = \sum_{j=0}^{\#livelli} n = n (\#livelli + 1)$$

Ci rimane da determinare questo  $\#livelli$

L'ultimo livello corrisponde ad una serie di chiamate di  $T(1)$  (ci fermiamo quando la dimensione del problema è 1)

$1 = n/2^k$  con  $k$  pari all'altezza dell'albero di ricorsione e quindi al  $\#livelli$ :  $n/2^k = 1$  implica  $2^k = n$  e quindi  $\#livelli = k = \log_2 n$

# Alberi di ricorsione

Ricapitolando:

$$T(n) = n (\log_2 n + 1) = \Theta(n \log_2 n)$$

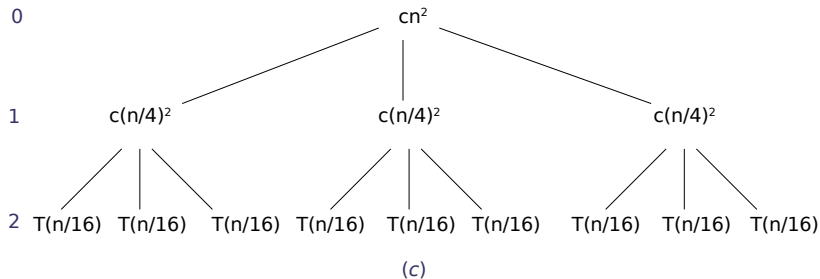
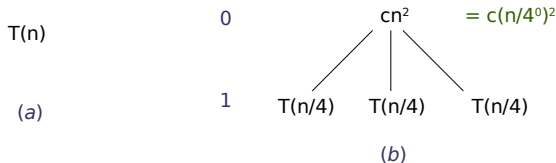
# Alberi di ricorsione: un altro esempio

Consideriamo la ricorrenza

$$T(n) = \begin{cases} 3T(n/4) + cn^2 & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

e costruiamo la derivazione dell'albero delle ricorrenze per  $T(n)$

# Alberi di ricorsione: un altro esempio



## Alberi di ricorsione: un altro esempio

Il numero di nodi al livello  $i$  è pari a  $3^i$ . Inoltre il costo di ciascun nodo al livello  $i$  è  $c(n/4^i)^2 = cn^2/16^i$ . Allora il costo di ciascun livello  $i$  è  $3^i \times cn^2/16^i = c(3/16)^i n^2$

$k + 1$  livelli dove  $k$  è tale che  $n/4^k = 1$  ossia  $k = \log_4 n$

Il costo complessivo è

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n} c(3/16)^i n^2 = cn^2 \sum_{i=0}^{\log_4 n} (3/16)^i = cn^2 \left( \frac{1 - (3/16)^{\log_4 n + 1}}{1 - (3/16)} \right) \\ &= cn^2 \left( \frac{1 - (3/16)^{\log_4 n + 1}}{13/16} \right) = \frac{16}{13} cn^2 \left( 1 - (3/16)^{\log_4 n + 1} \right) \end{aligned}$$

# Alberi di ricorsione: un altro esempio

Ora

$$(3/16)^{\log_4 n+1} = \frac{3}{16} \left( \frac{3}{16} \right)^{\log_4 n} = \frac{3}{16} \left( \frac{3^{\log_4 n}}{16^{\log_4 n}} \right) =$$

$$\frac{3}{16} \left( \frac{n^{\log_4 3}}{n^{\log_4 16}} \right) = \frac{3}{16} \left( \frac{n^{\log_4 3}}{n^2} \right) = \frac{3n^{\log_4 3}}{16n^2}$$

Quindi, il costo complessivo è

$$T(n) = \frac{16}{13} cn^2 \left( 1 - (3/16)^{\log_4 n+1} \right) = \frac{16}{13} cn^2 \left( 1 - \frac{3n^{\log_4 3}}{16n^2} \right) =$$

$$\frac{16}{13} cn^2 \left( \frac{16n^2 - 3n^{\log_4 3}}{16n^2} \right) = \frac{c}{13} \left( 16n^2 - 3n^{\log_4 3} \right) = O(n^2)$$

# Metodo della sostituzione

**Metodo della sostituzione:** “indovinare” una possibile soluzione ed usare l'induzione matematica per dimostrare che la soluzione è corretta

Consideriamo di nuovo la ricorrenza

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

e dimostriamo, applicando il metodo della sostituzione, che

$$T(n) = O(n)$$

# Metodo della sostituzione

Dobbiamo dimostrare che esistono delle costanti positive  $c$  ed  $n_0$  tali che  $0 \leq T(n) \leq cn$  per ogni  $n \geq n_0$

**Caso base**  $n = 1$ :  $T(1) = 1 \leq c1 = c$  per ogni costante  $c \geq 1$  (positiva)

**Passo induttivo**: assumiamo  $T(n') \leq cn'$  per ogni  $n' < n$ . Allora

$$\begin{aligned} T(n) &= n + T(n/2) \\ &\leq n + cn/2 \\ &= n(1 + c/2) \quad \text{se scegliamo } c \geq 2^1 \\ &\leq cn \end{aligned}$$

---

<sup>1</sup>Infatti se  $c \geq 2$ , allora  $1 \leq c/2$  e  $1 + c/2 \leq c/2 + c/2 = c$

# Metodo della sostituzione: un altro esempio

Consideriamo la seguente ricorrenza

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

Dimostriamo che

$$T(n) = O(n \log_2 n)$$

ossia che esistono delle costanti positive  $c$  ed  $n_0$  tali che

$$0 \leq T(n) \leq cn \log_2 n \text{ per ogni } n \geq n_0$$

**Caso base**  $n = 1$  :  $T(1) = 1 \leq c \cdot 0 = 0$  è chiaramente falsa

$$n = 2: T(2) = 2 + 2T(1) = 4 \leq 2c \text{ basta scegliere } c \geq 2$$

# Metodo della sostituzione: un altro esempio

**Passo induttivo:** assumiamo  $T(m) \leq cm \log_2 m$  per ogni  $m < n$ .  
Allora

$$\begin{aligned} T(n) &= n + 2T(n/2) \\ &\leq n + 2c \frac{n}{2} \log_2 \frac{n}{2} \\ &= n + cn \log_2 \frac{n}{2} \\ &= n + cn(\log_2 n - \log_2 2) \\ &= n + cn(\log_2 n - 1) \\ &= cn \log_2 n + n(1 - c) \\ &\leq cn \log_2 n \quad \text{per ogni } c \geq 2 \end{aligned}$$

Quindi, se scegliamo  $c \geq 2$  ed  $n_0 = 2$ ,  $T(n) \leq cn \log_2 n$  per ogni  $n \geq n_0$

# Come formulare una buona ipotesi?

- Non esiste un metodo generale per formulare una “buona” (corretta) ipotesi
- Esistono delle euristiche che possono aiutarci
- È inanzitutto possibile utilizzare gli alberi di ricorsione per farsi un'idea dell'andamento della ricorrenza
- In altri casi si può procedere per similitudine. Ad esempio: se  $T(n) = 2T(n/2) + n = O(n \log n)$ , allora sembra abbastanza probabile che anche  $T(n) = 2(T(n/2) + 17) + n = O(n \log n)$ .

# Finezze

- Si assuma di voler dimostrare che  $T(n) = 2T(n/2) + 1 = O(n)$  ossia che esistono delle costanti positive  $c$  ed  $n_0$  tali che

$$T(n) \leq cn \text{ per ogni } n \geq n_0$$

- Se scegliamo  $c \geq 1$ , il caso base è banalmente vero:  $T(1) = 1 \leq c$
- Ipotesi induttiva:

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\leq 2c\frac{n}{2} + 1 \\ &= cn + 1 \not\leq cn \end{aligned}$$

- Saremmo tentati di dimostrare che  $T(n) = O(n^2)$ , che può funzionare, ma la nostra ipotesi è corretta.

# Finezze

- Possiamo provare a dimostrare un'ipotesi induttiva più forte: esistono delle costanti positive  $c$ ,  $b$  ed  $n_0$  tali che

$$T(n) \leq cn - b \text{ per ogni } n \geq n_0$$

- Se scegliamo  $c \geq b + 1$ , (e quindi  $c - b \geq 1$ ) il caso base è verificato vero:  $T(1) = 1 \leq c - b$
- Ipotesi induttiva:

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\leq 2(c\frac{n}{2} - b) + 1 \\ &= cn - 2b + 1 \\ &\leq cn - 2b + b \quad \text{se scegliamo } 1 \leq b \\ &= cn - b \end{aligned}$$

- Ricapitolando: se scegliamo  $b \geq 1$ ,  $c \geq b + 1 \geq 2$ , allora  $T(n) \leq cn - b$  per ogni  $n \geq n_0 = 1$

# Teorema dell'esperto – Teorema Master

Permette di analizzare algoritmi basati sulla tecnica del **divide et impera**:

- dividi il problema (di dimensione  $n$ ) in  $a$  sotto-problemi di dimensione  $n/b$
- risolvi i sotto-problemi ricorsivamente
- ricombina le soluzioni

Sia  $f(n)$  il tempo per dividere e ricombinare istanze di dimensione  $n$ . La relazione di ricorrenza è data da:

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

# Teorema dell'esperto – Teorema Master

Per calcolare la soluzione della ricorrenza

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

confronta il comportamento asintotico  $f(n)$  con quello di  $n^{\log_b a}$ ; distingue tre possibili casi

- 1 Se  $n^{\log_b a}$  cresce più velocemente di  $f(n)$ , allora  $T(n) = \Theta(n^{\log_b a})$
- 2  $n^{\log_b a}$  cresce esattamente come  $f(n)$ , allora  $T(n) = \Theta(n^{\log_b a} \log_2 n)$
- 3 Se  $n^{\log_b a}$  cresce più lentamente di  $f(n)$ , allora  $T(n) = \Theta(f(n))$

# Teorema dell'esperto – Teorema Master

Per calcolare la soluzione della ricorrenza

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

confronta il comportamento asintotico  $f(n)$  con quello di  $n^{\log_b a}$ ; distingue tre possibili casi

- 1 Se  $f(n) = O(n^{\log_b a - \varepsilon})$  per qualche costante  $\varepsilon > 0$ , allora  $T(n) = \Theta(n^{\log_b a})$
- 2 Se  $f(n) = \Theta(n^{\log_b a})$ , allora  $T(n) = \Theta(n^{\log_b a} \log_2 n)$
- 3 Se (i)  $f(n) = \Omega(n^{\log_b a + \varepsilon})$  per qualche costante  $\varepsilon > 0$  e (ii) per qualche costante  $c < 1$ ,  $af(n/b) \leq cf(n)$ , allora  $T(n) = \Theta(f(n))$

# Applicazioni del teorema dell'esperto: caso 2

$$T(n) = n + 2T(n/2)$$

$$a = b = 2, \quad \log_b a = 1 \quad \text{e} \quad n^{\log_b a} = n$$

$$f(n) = n = \Theta(n^{\log_b a})$$



$$T(n) = \Theta(n^{\log_b a} \log_2 n) = \Theta(n \log_2 n)$$

# Applicazioni del teorema dell'esperto: caso 1

$$T(n) = c + 9T(n/3)$$

$$a = 9, b = 3, \log_b a = \log_3 9 = 2 \quad \text{e} \quad n^{\log_b a} = n^2$$
$$f(n) = c = O(n) = O(n^{\log_b a - \varepsilon}) = O(n^{2-\varepsilon}) \quad \text{con } \varepsilon = 1$$
$$\Downarrow$$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

## Applicazioni del teorema dell'esperto: caso 3

$$T(n) = n + 3T(n/9)$$

$$a = 3, b = 9, \log_b a = \log_9 3 = 1/2$$

$$f(n) = n = \Omega(n) = \Omega(n^{\log_9 3 + \varepsilon}) \text{ con } \varepsilon = 1/2$$

inoltre,

$$af(n/b) = 3f(n/9) = 3(n/9) = 1/3n \leq cf(n), \text{ per } c = 1/3$$



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