ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

RICORSIONE

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

RICORSIONE

Una FUNZIONE RICORSIVA è una funzione che chiama se stessa

Ci DEVONO essere di CASI BASE in cui la funzione restituisce un risultato senza chiamare se stessa

Ci sono dei CASI NON BASE in cui la funzione restituisce un risultato chiamando se stessa con parametri più vicini ai casi base (SOTTOPROBLEMA)

Permette di risolvere problemi in modo elegante e con codici compatti

RICORSIONE

ESEMPIO: fattoriale

$$f(n) = \begin{cases} 1 & \text{se } n = 1 \\ n \cdot f(n-1) & \text{altrimenti} \end{cases}$$

Caso Base

Caso non Base

Dimensione del problema originario

Dimensione del sottoproblema su cui viene invocata la ricorsione

RICORSIONE

ESEMPIO: fattoriale

$$f(n) = \begin{cases} 1 & \text{se } n = 1 \\ n \cdot f(n-1) & \text{altrimenti} \end{cases}$$

Caso Base

Caso non Base

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Chiamata ricorsiva

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Prima chiamata `fatt(5)`

$$5 \neq 1$$

`return 5*…`

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Prima chiamata `fatt(5)`

$5 \neq 1$

`return 5*…`

`fatt(4)` Seconda chiamata

$4 \neq 1$

`return 4*…`

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Prima chiamata `fatt(5)`

$5 \neq 1$

`return 5*…`

`fatt(4)` Seconda chiamata

$4 \neq 1$

`return 4*…`

`fatt(3)` Terza chiamata

$3 \neq 1$

`return 3*…`

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Prima chiamata `fatt(5)`

$5 \neq 1$

`return 5*…`

`fatt(4)` Seconda chiamata

$4 \neq 1$

`return 4*…`

`fatt(3)` Terza chiamata

$3 \neq 1$

`return 3*…`

`fatt(2)` Quarta chiamata

`return 2*…`

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Prima chiamata `fatt(5)`

$5 \neq 1$

`return 5*…`

`fatt(4)` Seconda chiamata

$4 \neq 1$

`return 4*…`

`fatt(3)` Terza chiamata

$3 \neq 1$

`return 3*…`

`fatt(2)` Quarta chiamata

`return 2*…`

`fatt(1)` Quinta chiamata

`return 1`

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
   then return 1
   else return n*fatt(n-1)
```

Prima chiamata `fatt(5)`

$5 \neq 1$

5*4*3*2*1

`return 5*…`

`fatt(4)` Seconda chiamata

$4 \neq 1$

4*3*2*1

`return 4*…`

`fatt(3)` Terza chiamata

$3 \neq 1$

3*2*1

`return 3*…`

`fatt(2)` Quarta chiamata

2*1

`return 2*…`

`fatt(1)` Quinta chiamata

1

`return 1`

ESEMPIO: fattoriale

Costo computazionale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Possiamo scrivere $T(n)$ sfruttando la natura ricorsiva della funzione `fatt(n)`

- Se $n = 1$, allora la funzione esegue l'istruzione nel ramo **`then`**, e termina eseguendo solo una operazione logica
- Altrimenti, esegue una operazione aritmetica e successivamente tutte quelle eseguite da **`fatt(n-1)`**

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 1 + T(n-1) & \text{altrimenti} \end{cases}$$

EQUAZIONE di RICORRENZA

ESEMPIO: fattoriale

Costo computazionale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 1 + T(n-1) & \text{altrimenti} \end{cases}$$

Come determinare una forma chiusa per $T(n)$?

$$T(n) = 1 + T(n-1)$$

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Costo computazionale

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 1 + T(n-1) & \text{altrimenti} \end{cases}$$

Come determinare una forma chiusa per $T(n)$?

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + 1 + T(n-2) = 2 + T(n-2) \\ &= 2 + 1 + T(n-3) = 3 + T(n-3) \\ &= \dots. \\ &= i + T(n-i) \\ &= \dots. \\ &= (n-1) + T(n-(n-1)) = (n-1) + 1 = n \end{aligned}$$

al passo generico i → $= i + T(n-i)$

ultimo passo per $i = n - 1$ → $= (n-1) + T(n-(n-1)) = (n-1) + 1 = n$

ESEMPIO: fattoriale

```
fatt(n)
 if n=1
  then return 1
  else return n*fatt(n-1)
```

Costo computazionale

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 1 + T(n-1) & \text{altrimenti} \end{cases}$$

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

E' LINEARE?

NO, E' ESPONENZIALE!!

Dimensione dell'input —> $\log n$ bit

RICORSIONE

ESEMPIO: numeri di Fibonacci

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ f(n-1) + f(n-2) & \text{altrimenti} \end{cases}$$

Casi Base

Caso non Base

RICORSIONE

ESEMPIO: numeri di Fibonacci

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ f(n-1) + f(n-2) & \text{altrimenti} \end{cases}$$

Casi Base

Caso non Base

Dimensione del problema originario

Dimensione di UN sottoproblema su cui viene invocata la ricorsione

Dimensione di UN ALTRO sottoproblema su cui viene invocata la ricorsione

RICORSIONE

ESEMPIO: numeri di Fibonacci

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ f(n-1) + f(n-2) & \text{altrimenti} \end{cases}$$

Casi Base

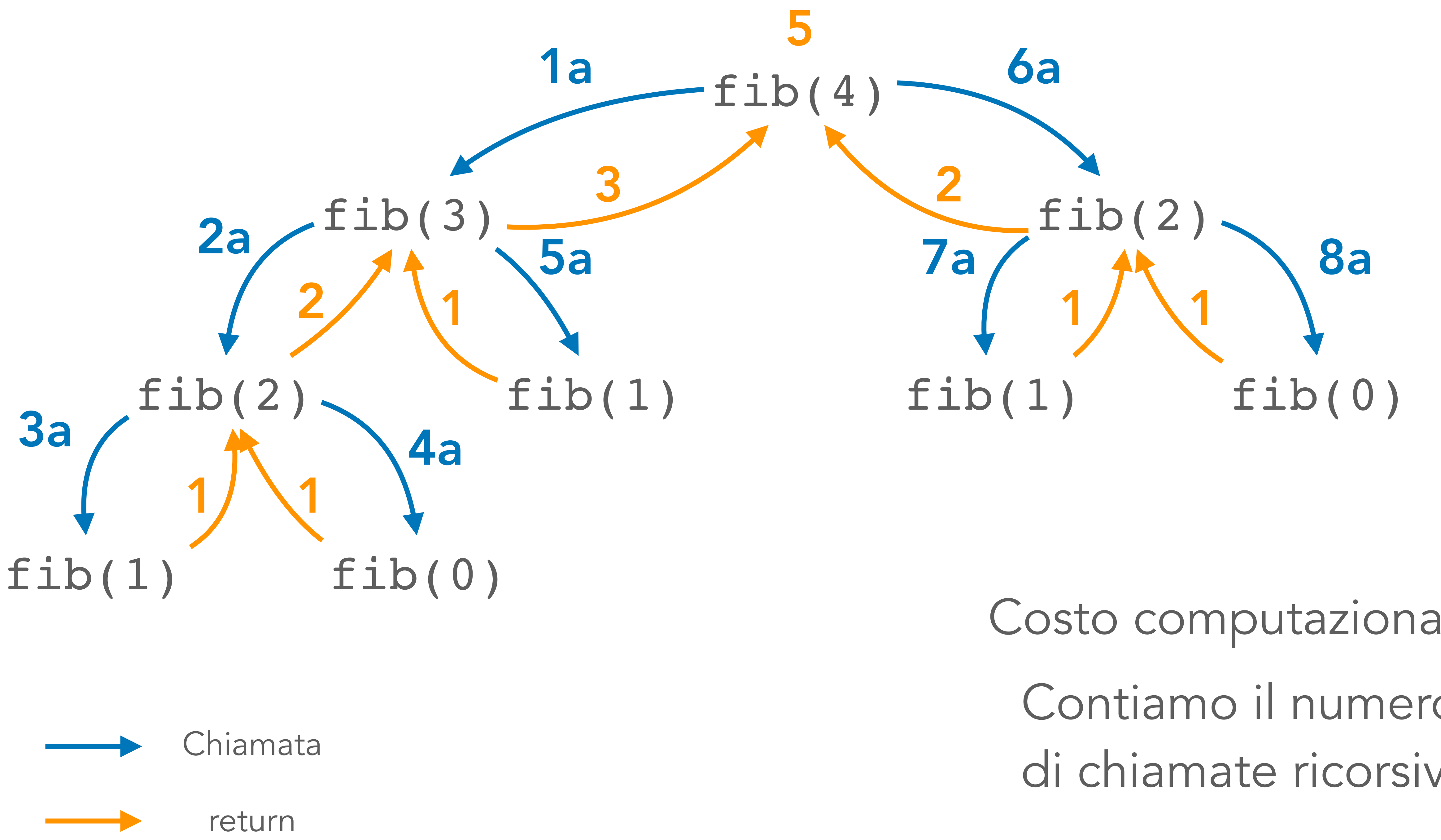
Caso non Base

```
fib(n)
 if n=0 or n=1
  then return 1
  else return fib(n-1)+fib(n-2)
```

Chiamate ricorsive

ESEMPIO: numeri di Fibonacci

```
fib(n)
 if n=0 or n=1
  then return 1
  else return fib(n-1)+fib(n-2)
```



Costo computazionale?

Contiamo il numero di chiamate ricorsive

UNIMORE

ESEMPIO: numeri di Fibonacci

```
fib(n)
 if n=0 or n=1
  then return 1
  else return fib(n-1)+fib(n-2)
```

$T(n)$ = numero di chiamate ricorsive necessarie per calcolare **fib(n)**

$$T(n) = \begin{cases} 0 & \text{se } n = 0,1 \\ 2 & \text{se } n = 2 \\ 2 + T(n-1) + T(n-2) & \text{altrimenti} \end{cases}$$

Le chiamate sui sottoproblemi di dimensione n-1 e n-2

Il numero di chiamate per risolvere il sottoproblema di dimensione n-1

Il numero di chiamate per risolvere il sottoproblema di dimensione n-2

ESEMPIO: numeri di Fibonacci

$$T(n) = \begin{cases} 0 & \text{se } n = 0,1 \\ 2 & \text{se } n = 2 \\ 2 + T(n-1) + T(n-2) & \text{altrimenti} \end{cases}$$

$T(n)$ = numero di chiamate ricorsive necessarie per calcolare **`fib(n)`**

$$\boxed{T(n)} = 2 + T(n-1) + T(n-2) \boxed{\leq 2 \cdot T(n-1)} \leq$$

$$\leq 2 \cdot 2 \cdot T(n-2) \leq 2 \cdot 2 \cdot 2T(n-3) \leq \ldots \leq 2^i T(n-i) \leq \ldots$$

$$\leq 2^{n-2} T(n-(n-2)) = 2^{n-2} T(2) = 2^{n-2} \cdot 2 = 2^{n-1}$$

$2 + T(n-2) \leq 2 + T(n-2) + T(n-3) = T(n-1)\ \forall n \geq 3$

quindi

$2 + T(n-1) + T(n-2) \leq T(n-1) + T(n-1) = 2T(n-1)$

per $n \geq 3$

$\Rightarrow \quad T(n) \in O(2^n)$

ESEMPIO: numeri di Fibonacci

$$T(n) = \begin{cases} 0 & \text{se } n = 0,1 \\ 2 & \text{se } n = 2 \\ 2 + T(n-1) + T(n-2) & \text{altrimenti} \end{cases}$$

$T(n)$ = numero di chiamate ricorsive necessarie per calcolare **`fib(n)`**

$$\boxed{T(n)} = 2 + T(n-1) + T(n-2) \geq \boxed{2 \cdot T(n-2) \geq}$$

$$\geq 2 \cdot 2 \cdot T(n-4) \geq 2 \cdot 2 \cdot 2 \cdot T(n-6) \geq \ldots \geq 2^i T(n-2i) \geq \ldots$$

$$\geq 2^{n/2-1} T(n - 2(n/2-1)) = 2^{n/2-1} T(2) = 2^{n/2-1} \cdot 2 = 2^{n/2}$$

$2 + T(n-1) = 2 + T(n-2) + T(n-3) \geq T(n-2)\ \forall n \geq 3$

quindi

$2 + T(n-1) + T(n-2) \geq T(n-2) + T(n-2) = 2T(n-2)$

per $n \geq 3$

$$\Rightarrow \quad T(n) \in \Omega(2^{n/2})$$

ESEMPIO: numeri di Fibonacci

$$T(n) = \begin{cases} 0 & \text{se } n = 0,1 \\ 2 & \text{se } n = 2 \\ 2 + T(n-1) + T(n-2) & \text{altrimenti} \end{cases}$$

$T(n)$ = numero di chiamate ricorsive necessarie per calcolare `fib(n)`

$T(n) \in O(2^n)$

$T(n) \in \Omega(2^{n/2})$

$\Rightarrow$ $T(n) \in \Theta(2^n)$

INTUIZIONE
Sono necessarie
e sufficienti un numero
di chiamate ricorsive
esponenziali in n

Possiamo fare di meglio?

Calcolare `fib(n)` iterativamente,
con un approccio bottom-up

ESERCIZIO

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ a \cdot T(\frac{n}{b}) + c_2 \cdot n^d & \text{altrimenti} \end{cases}$$

Costanti

$a \in \mathbb{N}, a \geq 1$

$b > 1$

$c_1 \geq 0,\ c_2 > 0$

$d \geq 0$

OSSERVAZIONE: $\dfrac{n}{b}$ può stare sia per $\lfloor \dfrac{n}{b} \rfloor$, che per $\lceil \dfrac{n}{b} \rceil$

UNIMORE

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ a \cdot T(\frac{n}{b}) + c_2 \cdot n^d & \text{altrimenti} \end{cases}$$

Costanti

$a \in \mathbb{N}, a \geq 1$

$b > 1$

$c_1 \geq 0,\ c_2 > 0$

$d \geq 0$

Dimensione del problema originario

Numero di sottoproblemi

Dimensione de sottoproblema

Lavoro extra oltre le chiamate ricorsive

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ a \cdot T(\frac{n}{b}) + c_2 \cdot n^d & \text{altrimenti} \end{cases}$$

Costanti

$a \in \mathbb{N}, a \geq 1$

$b > 1$

$c_1 \geq 0,\ c_2 > 0$

$d \geq 0$

ESEMPIO: binary search

$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ 1 \cdot T(\frac{n}{2}) + c_2 \cdot n^0 & \text{altrimenti} \end{cases}$$

$$a = 1,\ b = 2,\ d = 0$$

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

SE

$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ a \cdot T(\frac{n}{b}) + c_2 \cdot n^d & \text{altrimenti} \end{cases}$$

Costanti

$a \in \mathbb{N}, a \geq 1$

$b > 1$

$c_1 \geq 0,\ c_2 > 0$

$d \geq 0$

ALLORA

$$T(n) \in \begin{cases} O(n^d) & \text{se } d > \log_b a \\ O(n^{\log_b a} \log n) & \text{se } d = \log_b a \\ O(n^{\log_b a}) & \text{se } d < \log_b a \end{cases}$$

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

$$T(n) \in \begin{cases} O(n^d) & \text{se } d > \log_b a \\ O(n^{\log_b a} \log n) & \text{se } d = \log_b a \\ O(n^{\log_b a}) & \text{se } d < \log_b a \end{cases}$$

ESEMPIO: binary search

$a = 1,\ b = 2,\ d = 0$

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

$$T(n) \in \begin{cases} O(n^d) & \text{se } d > \log_b a \\ O(n^{\log_b a} \log n) & \text{se } d = \log_b a \\ O(n^{\log_b a}) & \text{se } d < \log_b a \end{cases}$$

ESEMPIO: binary search

$a = 1,\ b = 2,\ d = 0$

$\log_b a = \log_2 1 = 0$

$d = 0$

$\log_b a = 0 = d$

$T(n) \in O(n^0 \log n)$

$T(n) \in O(\log n)$

UNIMORE

Equazioni di ricorrenza

TEOREMA dell'ESPERTO

OSSERVAZIONI:

- Il teorema **NON** può essere usato quando:
  - la ricorsione non è invocata su sottoproblemi dimensione una frazione costante della dimensione originaria (es. fattoriale);
  - la ricorsione è invocata su più sottoproblemi di dimensione diversa (es. fibonacci);
- Sul *Cormen et al.* viene presentata una forma più forte del teorema, più complessa da applicare (per questo esame basta la formulazione vista a lezione);
- Sul *Bertossi, Montresor*, può sembrare che la tesi del teorema si diversa, ma non lo è. Infatti, $\log_b a = \log_2 a / \log_2 b$

Algoritmi ricorsivi

La soluzione al problema viene fornita
scrivendo una funzione o una procedura ricorsiva

INPUT del problema

STRUTTURA GENERICA

Algoritmo $\mathcal{A}(I)$
if (I è ‘‘di taglia piccola’’)
    then return output
$\vdots$
$\mathcal{A}(I_1)$
$\vdots$
$\mathcal{A}(I_a)$
$\vdots$

caso base

altre istruzioni

chiamate ricorsive

Algoritmi ricorsivi

```
Algoritmo 𝒜(I)
if ( I è ``di taglia piccola'')
   then return output
⋮
  𝒜(I_1)
⋮
  𝒜(I_a)
⋮
```

Per derivare l'equazione di ricorrenza:

- Determinare la dimensione dell'input n
- Determinare il caso base (per qualche n_0)
- Determinare il costo del caso base $b(n)$
- Determinare il costo del lavoro extra ricorsione $g(n)$
- Determinare i sottoproblemi e la loro taglia m_i

$$T(n) = \begin{cases} b(n) & \text{se } n = n_0 \\ T(m_1) + T(m_2) + \ldots + T(m_a) + g(n) & \text{altrimenti} \end{cases}$$

= taglia del sottoproblema i-esimo