Modulo: Tecniche di programmazione ricorsiva [P2_11]

Unità didattica: Valutazione di programmi ricorsivi [2-AT]

Titolo: Analisi di function ricorsive

Argomenti trattati:

- ✓ Profondità di ricorsione
- ✓ Valutazione della complessità di tempo e di spazio di una function ricorsiva

Prerequisiti richiesti: function C, complessità computazionale asintotica

Esempio ...già visto!

```
float recurs_fact(short n)
{
if  (n <= 1) return 1;
else return n*recurs_fact(n-1);
}
```

```
float recurs_fact(short n)
{float nfatt;
if  (n <= 1) nfatt=1;
else nfatt=n*recurs_fact(n-1);
return nfatt;
}
```

La ricorsione semplifica talvolta la leggibilità dell'algoritmo, ma non sempre è conveniente per la complessità di spazio e di tempo.

Infatti ad ogni chiamata ricorsiva vengono allocate delle nuove variabili (temporanee) contenenti tutti i parametri formali e tutte le variabili locali della function.

? = chiamata ricorsiva

(4!) ?
4! = 4 * ?
3! = 3 * ?
2! = 2 * ?
1! = 1

stack	
n=1	nfatt= 1
n=2	nfatt= 2
n=3	nfatt= 6
n=4	nfatt=24

Nell'esempio precedente dell'algoritmo ricorsivo per il fattoriale di **n**, per **n=4** ci sono **4 chiamate**.

DEF

profondità di ricorsione (o livello di ricorsione)

=

massimo numero di chiamate ricorsive eseguite

La **profondità di ricorsione** dipende dal valore particolare dei dati di input.

La **profondità di ricorsione** moltiplicata per il numero di variabili usate dalla procedura (v. interne e parametri formali) quantifica l'occupazione di memoria dello stack di dati temporanei della procedura ricorsiva.

Analisi della **profondità di ricorsione** di un problema di dimensione *n*

Come viene influenzata la complessità computazionale dal numero di chiamate ricorsive effettuate nei vari tipi di algoritmi esaminati ?

ricorsione lineare

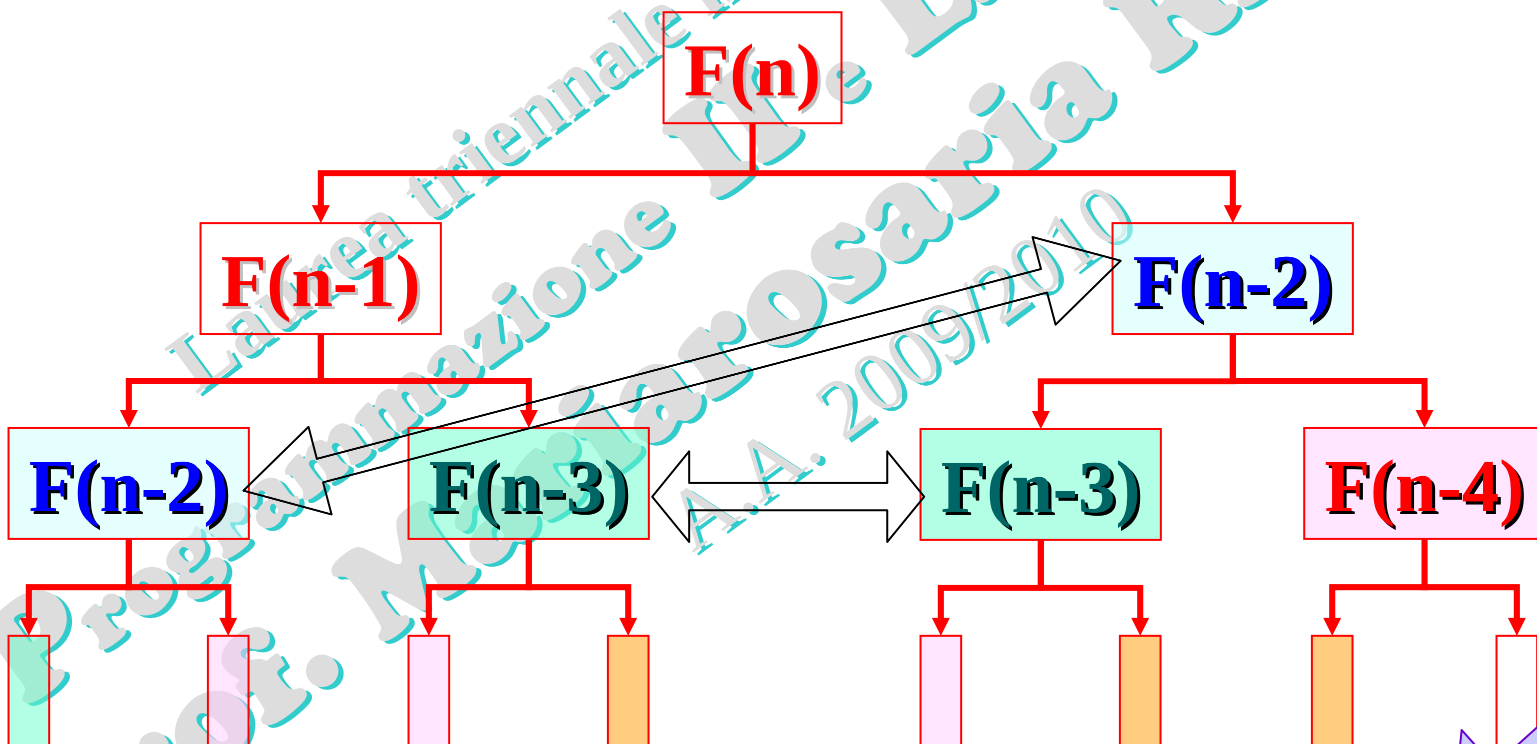
1 chiamata ricorsiva P(n)
⇔
1 attivazione

Profondità di ricorsione = $\mathcal{O}(n)$

ricorsione binaria ?

Es. 1: numeri di Fibonacci $F(n)=F(n-1)+F(n-2)$

chiamate ridondanti



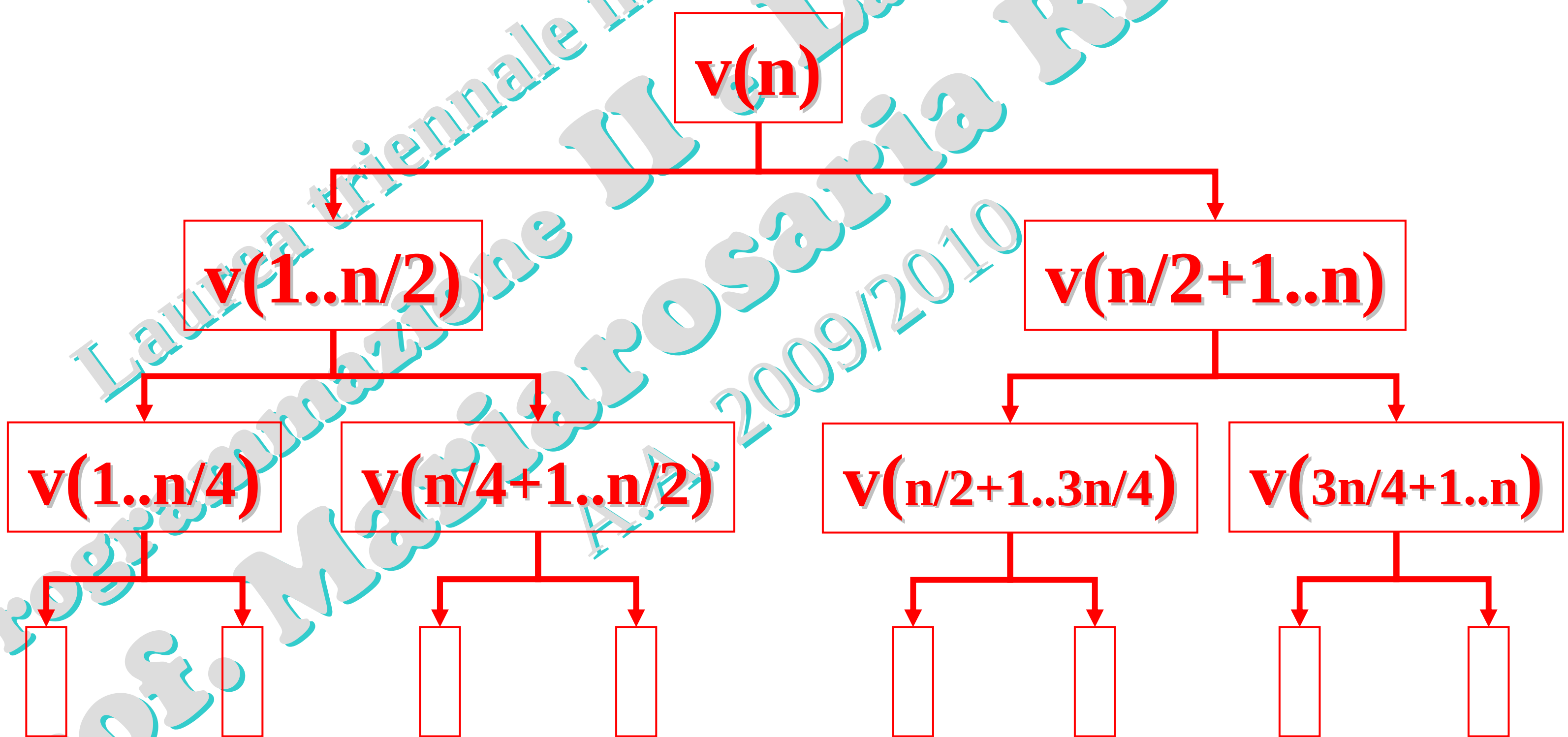
1 chiamata ricorsiva P(n) ⇔ 2 attivazioni

Profondità di ricorsione ≅ $\mathcal{O}(2^n)$

ricorsione binaria

Es. 2: massimo in un array

le chiamate operano su sottovettori disgiunti



Profondità di ricorsione = $\mathcal{O}(n)$

Gli esempi precedenti di ricorsione binaria (*sequenza di Fibonacci* e *massimo in un array*) sono algoritmi ricorsivi dalle prestazioni molto diverse in termini di complessità:

$$\textbf{Profondità di ricorsione} = \begin{cases} 2^n & \textbf{Fibonacci} \ \textit{(ordine esponenziale)} \\ n & \textbf{massimo} \ \textit{(ordine lineare)} \end{cases}$$

Entrambi suddividono il problema iniziale in due sottoproblemi, risolti poi indipendentemente:

- nel caso di Fibonacci i due sottoproblemi non sono indipendenti, bensì legati da una formula di ricorrenza $F(n)=F(n-1)+F(n-2)$;
- nel caso del massimo i due sottoproblemi sono indipendenti e risolti entrambi.

Profondità di ricorsione binaria

=

2^n
ordine esponenziale
(Fibonacci)

n
ordine lineare
(massimo)

ricorsione NO!

ricorsione SI!

non conviene!

Ricorsione binaria in Divide and Conquer (Divide et Impera)

Gli **algoritmi a ricorsione binaria realmente utilizzati** in pratica rientrano nella classe ***Divide et Impera*** ed hanno le seguenti caratteristiche:

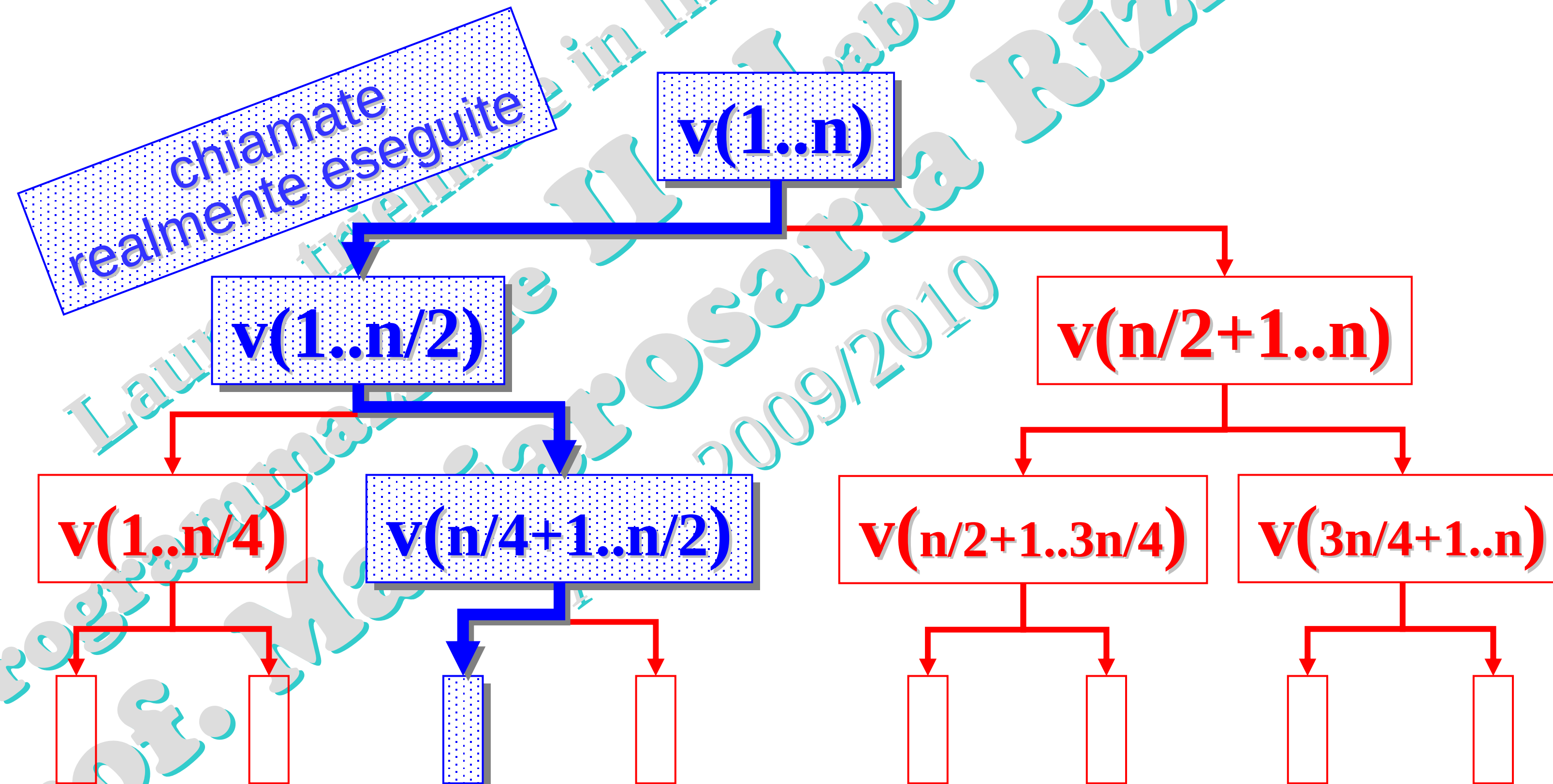
- dividono il problema iniziale in ***due sottoproblemi disgiunti*** (che hanno una dimensione (≈) pari alla metà del problema originario);
- ad ogni livello di ricorsione si risolve sempre ***uno solo*** dei due sottoproblemi.

Profondità di ricorsione = $\mathcal{O}(\log_2(n))$

ricorsione binaria ?

Es. 3: ricerca binaria in array

una sola chiamata per livello su sottovettori disgiunti



Profondità di ricorsione = $\mathcal{O}(\log_2(n))$

Profondità di ricorsione binaria

=

2^n	n	$log_2(n)$
ordine esponenziale	**ordine lineare**	**ordine logaritmico**
(Fibonacci)	(massimo)	(ricerca binaria)

?

ricorsione NO! **ricorsione SI!**

non conviene!