Capitolo 15

La ricorsione

Fondamenti di Informatica e Laboratorio - Modulo A

Corso di Laurea in Ingegneria Elettronica e Informatica

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

Ricorsione

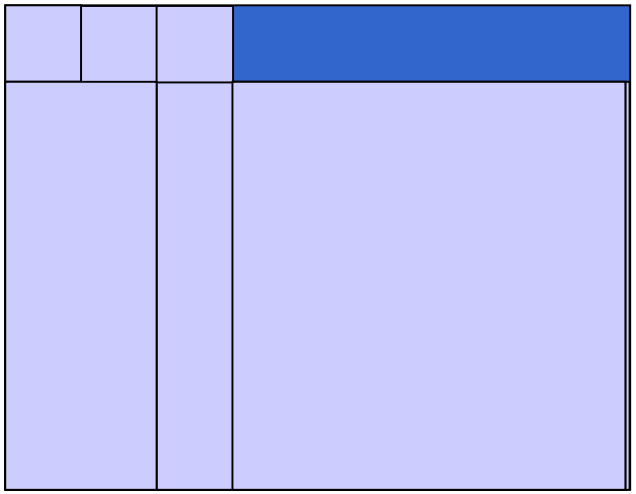
- Una funzione può invocare tutte le funzioni dichiarate prima di lei.
- In particolare, può richiamare anche se stessa.

- … ma …

serve a qualcosa???

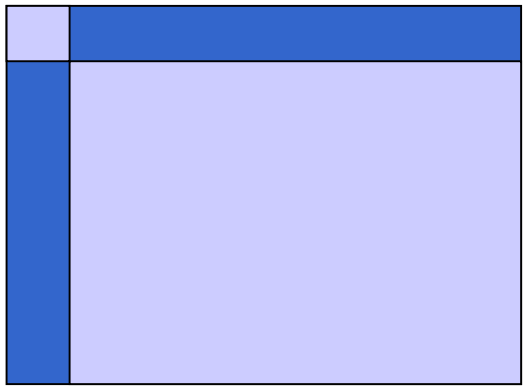
Esempio: determinante

- Il determinante di una matrice è definito in base al determinante di matrici più piccole



Esempio: determinante

- Il determinante di una matrice è definito in base al determinante di matrici più piccole
- Come fare a calcolarlo?
- Sarebbe bello scrivere



```
int det(matrice)
{   scegli una riga
    per tutti gli elementi
    {     calcola det(sottomatr)
          moltiplica per l'elemento
          (eventualm cambia di segno)
          aggiungi il risultato
    }
    return risultato
}
```

Es: Espressioni

- (Come visto in precedenza), un'espressione è

  <espressione> ::= *<variabile>* | *<costante>* |
  <espressione><operatore><espressione>|
  <operatore_unario><espressione>

- Quindi se voglio scrivere un programma che valuta delle espressioni, devo scrivere qualcosa del tipo:

```
int valuta(esp)
{  scomponi espressione in esp1 op esp2
   ris1=valuta(esp1);
   ris2=valuta(esp2);
   se (op=='+') return ris1+ris2;
   se (op=='*') return ris1*ris2;
   …
}
```

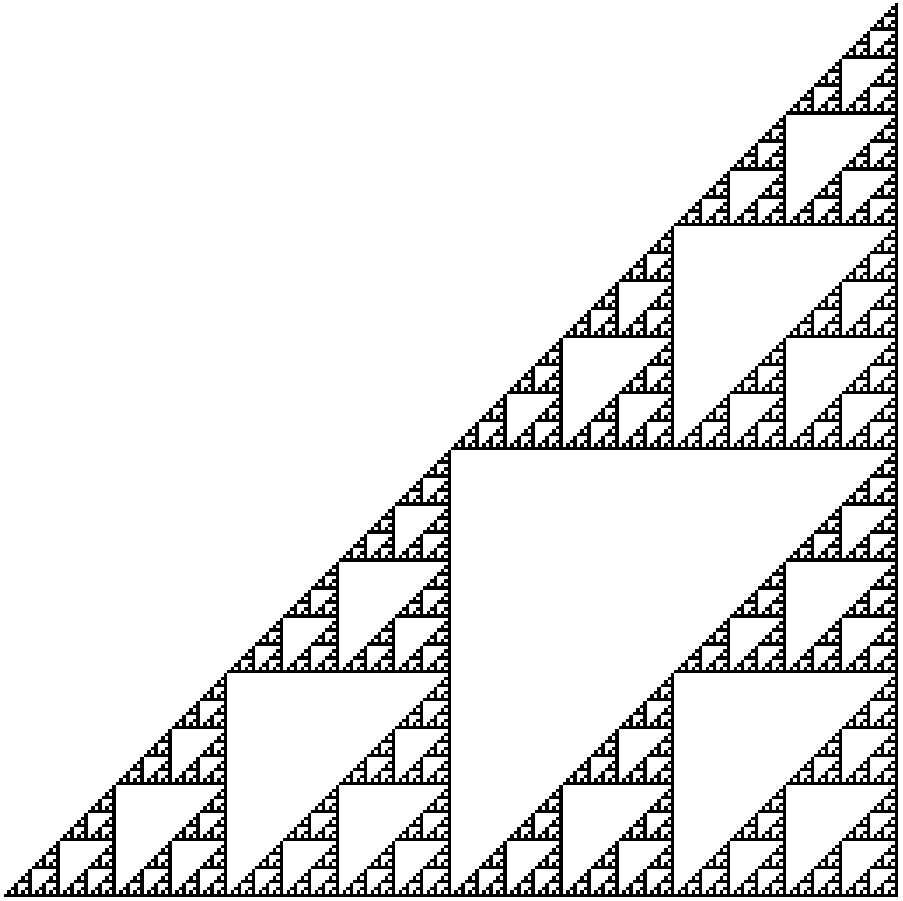
Es: calcolo della derivata

$$\frac{d(f(x)+g(x))}{dx} = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$$

$$\frac{d(f(x)\cdot g(x))}{dx} = \frac{df(x)}{dx} g(x) + f(x) \frac{dg(x)}{dx}$$

LA RICORSIONE

- Pensate a come si potrebbe disegnare la figura a lato (triangolo di Sierpinski)



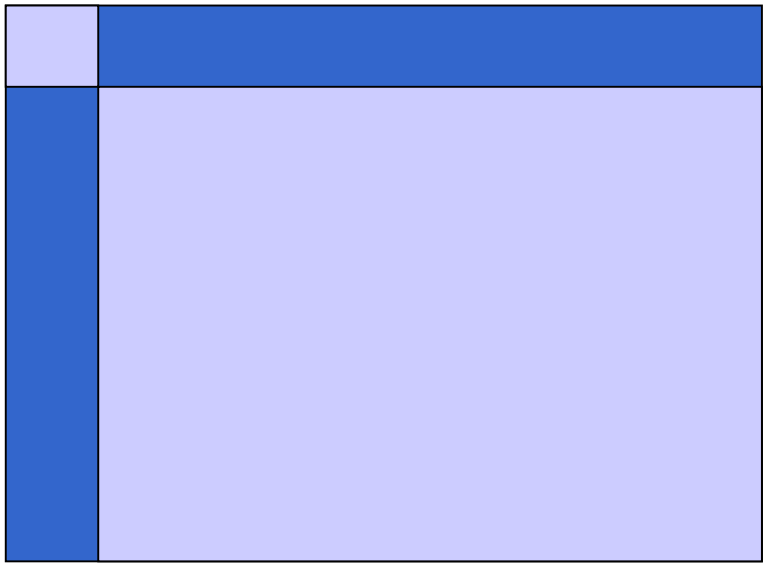
Quando mi fermo?

- Se non stabilisco un criterio per terminare, l'elaborazione continua in eterno!

Terminazione

- Tutte le funzioni ricorsive devono avere un caso *“base”* in cui si riesce a calcolare il risultato senza invocare la funzione ricorsivamente

Determinante



Espressioni

<*espressione*> ::=

<*variabile*> |

<*costante*> |

<*espressione*><*operatore*><*espressione*>|

<*operatore_unario*><*espressione*>

Es: calcolo della derivata

$$\frac{d(f(x)+g(x))}{dx} = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$$

$$\frac{d(f(x)\cdot g(x))}{dx} = \frac{df(x)}{dx} g(x) + f(x) \frac{dg(x)}{dx}$$

In pratica …

- TUTTE le procedure e funzioni che richiamano se stesse devono avere (almeno) un caso base

- In pratica, cominciano sempre con una selezione

```
if (condizione di uscita)
  risultato caso base
else
  parte con chiamata ricorsiva
```

LA RICORSIONE

- Una funzione matematica è definita ***ricorsivamente*** quando nella sua definizione compare un riferimento a se stessa
- La ricorsione consiste nella possibilità di *definire una funzione in termini di se stessa.*

- È basata sul *principio di induzione matematica*: se si può provare che una proprietà *P*
  - vale per $n=n_0$ (CASO BASE)
  - e, *assumendola valida per n*, allora vale per $n+1$

  allora *P* vale per ogni $n \geq n_0$

Giuseppe Peano

ESEMPIO: Fattoriale

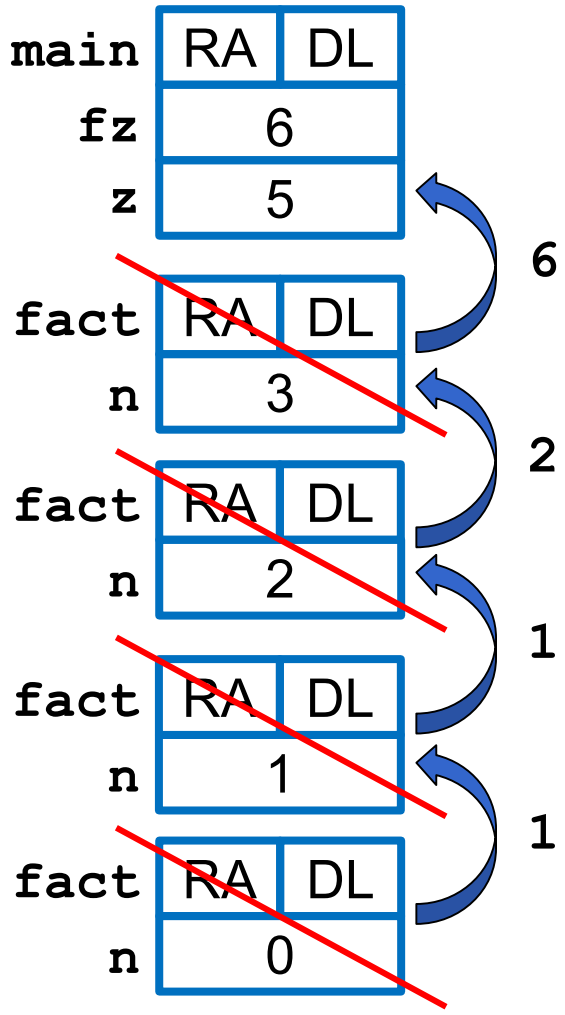
$$n! = \begin{cases} 1 & \text{se} \quad n = 0 \\ n\,(n\text{-}1)! & \text{se} \quad n > 0 \end{cases}$$

Codifica:

```
int fact(int n)
{    if (n==0)
          return 1;
     else return n*fact(n-1);
}
```

LA RICORSIONE: ESEMPIO

```
int fact(int n)
{ if (n==0)
        return 1;
  else return n*fact(n-1);
}
main()
{     int fz, z = 5;
      fz = fact(z-2);
}
```



Cosa succede nello stack ?

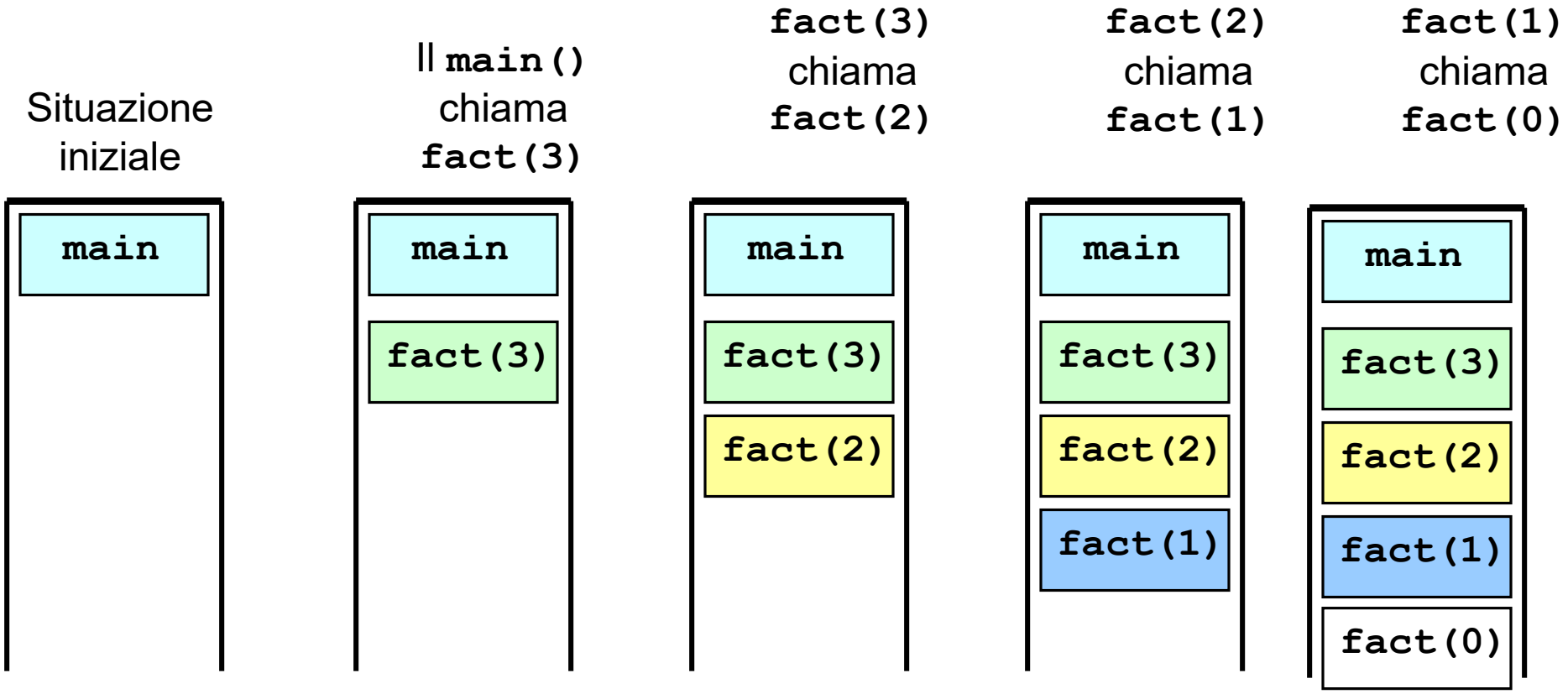
```
int fact(int n)
{   if (n<=0) return 1;
    else return n*fact(n-1);
}

main()
{   int fz,z = 5;
    fz = fact(z-2);
}
```

NOTA: Anche il **`main()`** è una funzione

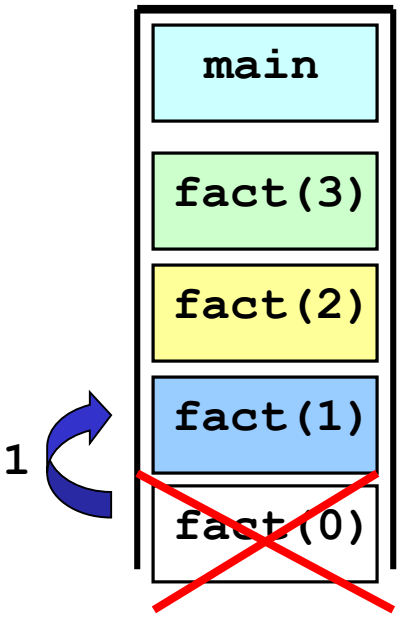
Seguiamo l'evoluzione dello stack durante l'esecuzione:

Cosa succede nello stack ?

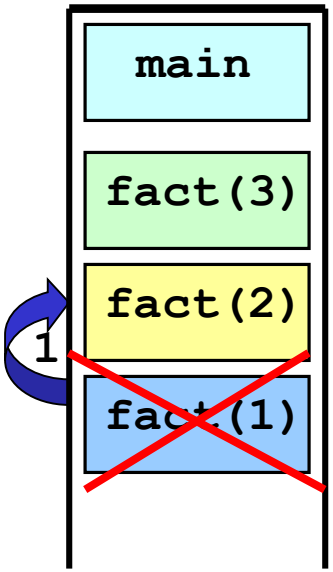


Cosa succede nello stack ?

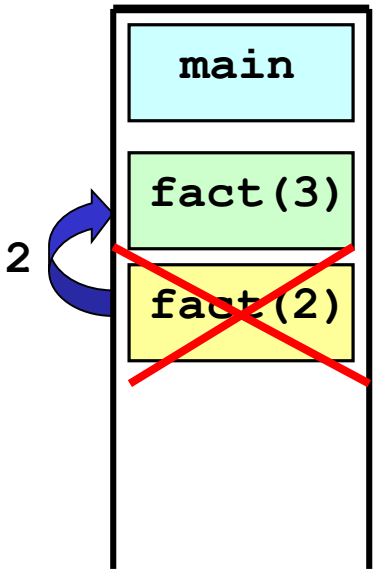
`fact(0)` termina restituendo il valore 1. Il controllo torna a `fact(1)`



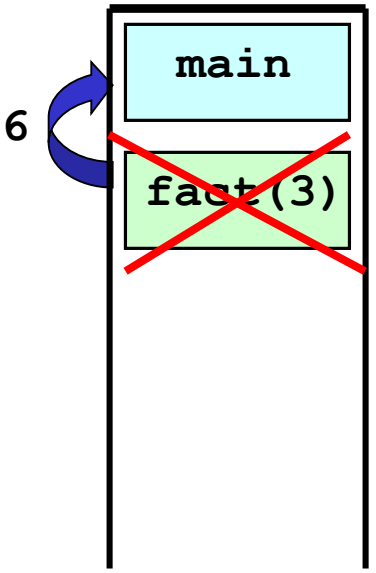
fact(1) effettua la moltiplicazione e termina restituendo il valore 1. Il controllo torna a fact(2)



fact(2) effettua la moltiplicazione e termina restituendo il valore 2. Il controllo torna a fact(3)



fact(3) effettua la moltiplicazione e termina restituendo il valore 6. Il controllo torna al main.



Calcolo della somma dei primi N numeri interi positivi

Capitolo 15 - La ricorsione

Calcolo della somma dei primi N numeri positivi

Fondamenti di Informatica e Laboratorio - Modulo A

Corso di Laurea in Ingegneria Elettronica e Informatica

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

Calcolo della somma dei primi N numeri interi positivi

- Si scriva una funzione ricorsiva che, dato un naturale *N*, calcola la somma dei primi *N* numeri interi positivi

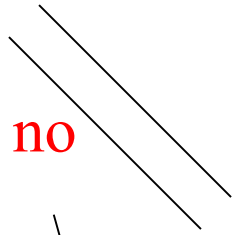
$$1+2+3+\ldots+(N-1)+N$$

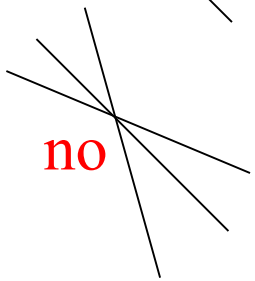
Problema

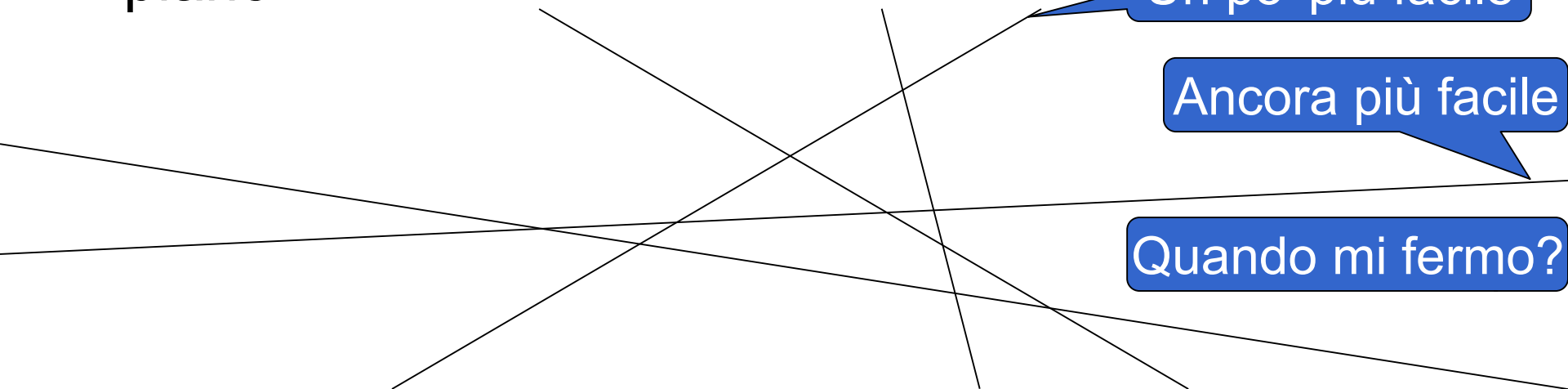
- Vengono tracciate *n* rette sul piano
- Non ci sono rette parallele
- Non accade mai che tre (o più) rette si incontrino nello stesso punto
- Si scriva una funzione

```
int regioni(int n)
```

che calcola il numero di regioni in cui viene diviso il piano

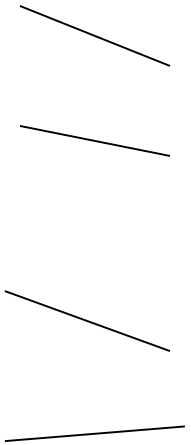






Problema

- Ora, supponiamo di sapere in quante regioni è diviso il piano con *n*-1 rette
- Aggiungiamo la retta *n*
- Quante nuove regioni vengono create?



LA RICORSIONE

- Operativamente, risolvere un problema con un approccio ricorsivo comporta
  1. identificare un "caso base" la cui soluzione sia nota
  2. riuscire a esprimere la soluzione al caso generico *n* in termini dello *stesso problema* *in uno o più casi più semplici* (*n*-1, *n*-2, etc).

Capitolo 15

La ricorsione - esercizio con array

Fondamenti di Informatica e Laboratorio - Modulo A

Corso di Laurea in Ingegneria Elettronica e Informatica

Anno accademico 2020/2021

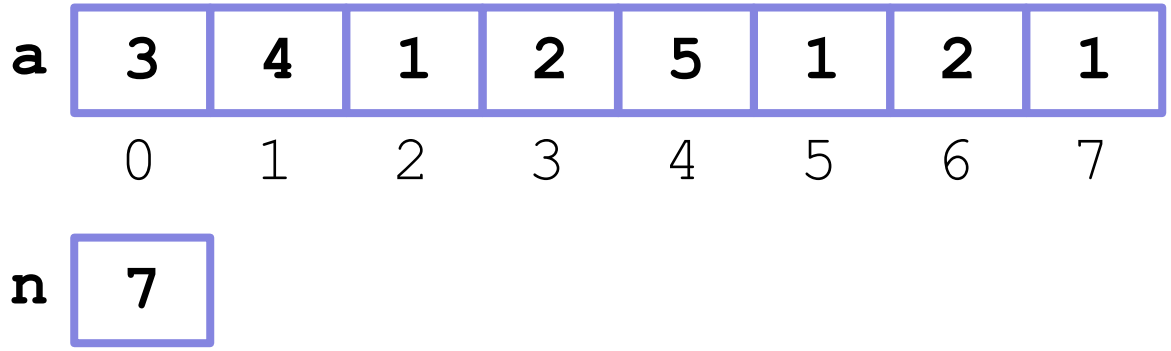
Prof. MARCO GAVANELLI

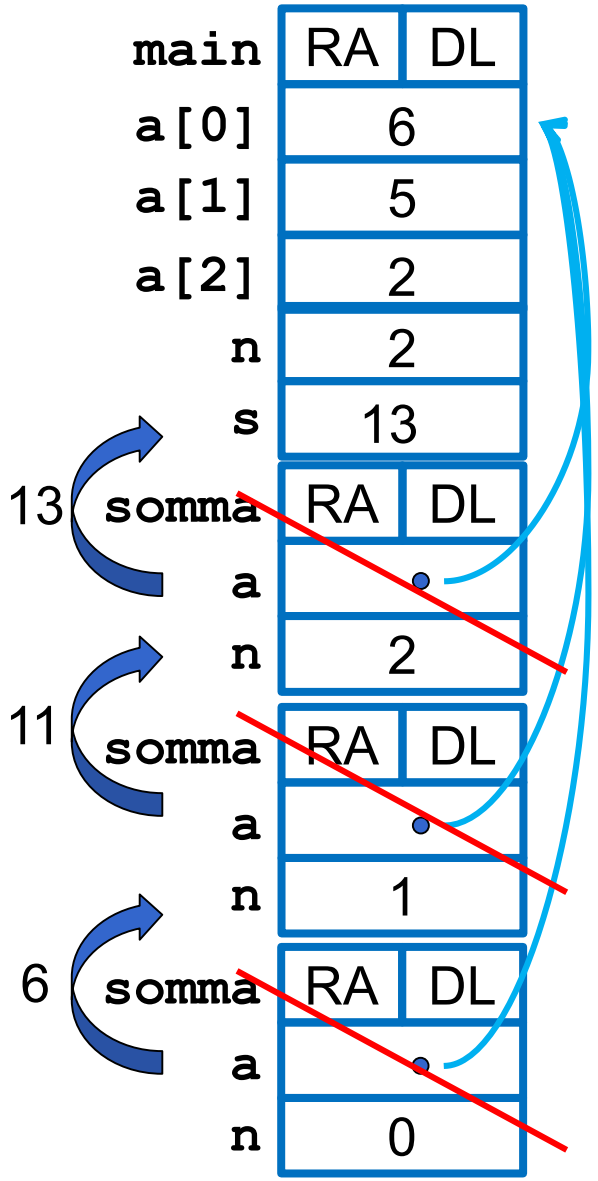
Somma elementi di un array

Si scriva una funzione ricorsiva

```
int somma(int a[], int n);
```

che calcola la somma degli elementi dell'array **a** dall'indice 0 fino all'indice **n**

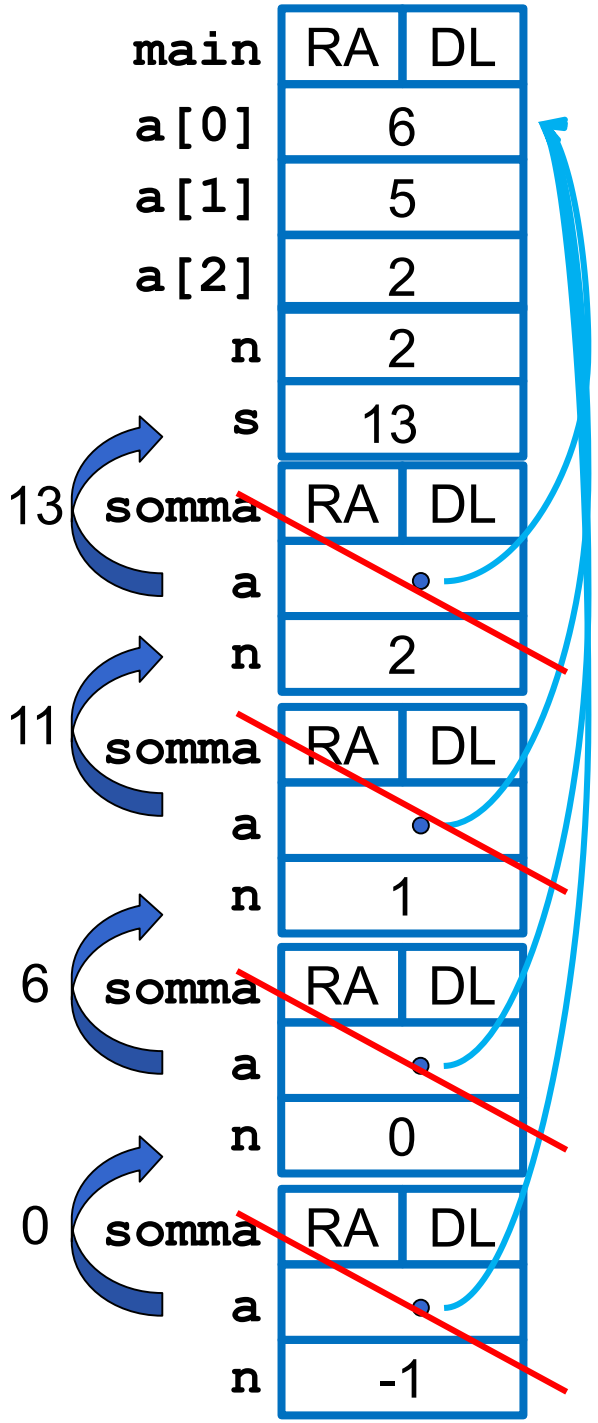




- Si mostri ora l'esecuzione della funzione con i record di attivazione quando viene invocata dal programma

```
int somma(int a[], int n)
{ if(n==0)
        return a[0];
  else return a[n]+somma(a,n-1);
}

main()
{ int a[4]={6,5,2}, n=2, s;
  s=somma(a,n);
}
```



- Si mostri ora l'esecuzione della funzione con i record di attivazione quando viene invocata dal programma

```
int somma(int a[], int n)
{ if(n<0)
        return 0;
  else return a[n]+somma(a,n-1);
}

main()
{ int a[4]={6,5,2}, n=2, s;
  s=somma(a,n);
}
```

Capitolo 15 - la ricorsione

Massimo di un array

Fondamenti di Informatica e Laboratorio - Modulo A

Corso di Laurea in Ingegneria Elettronica e Informatica

Anno accademico 2020/2021

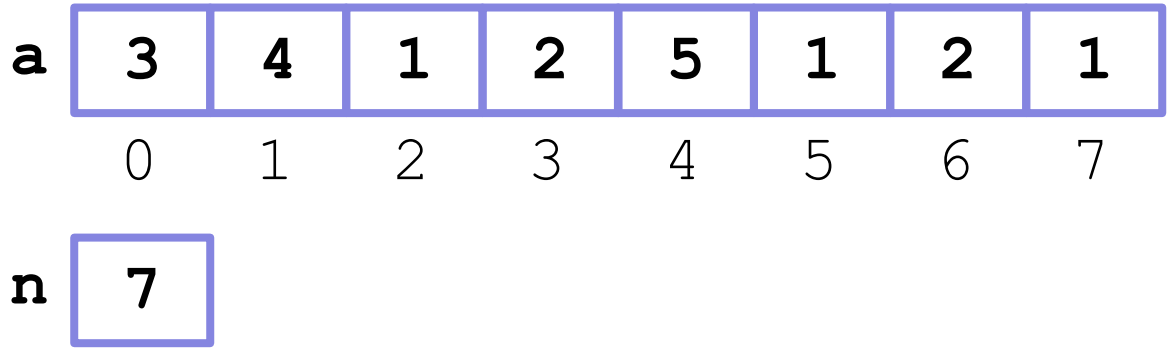
Prof. MARCO GAVANELLI

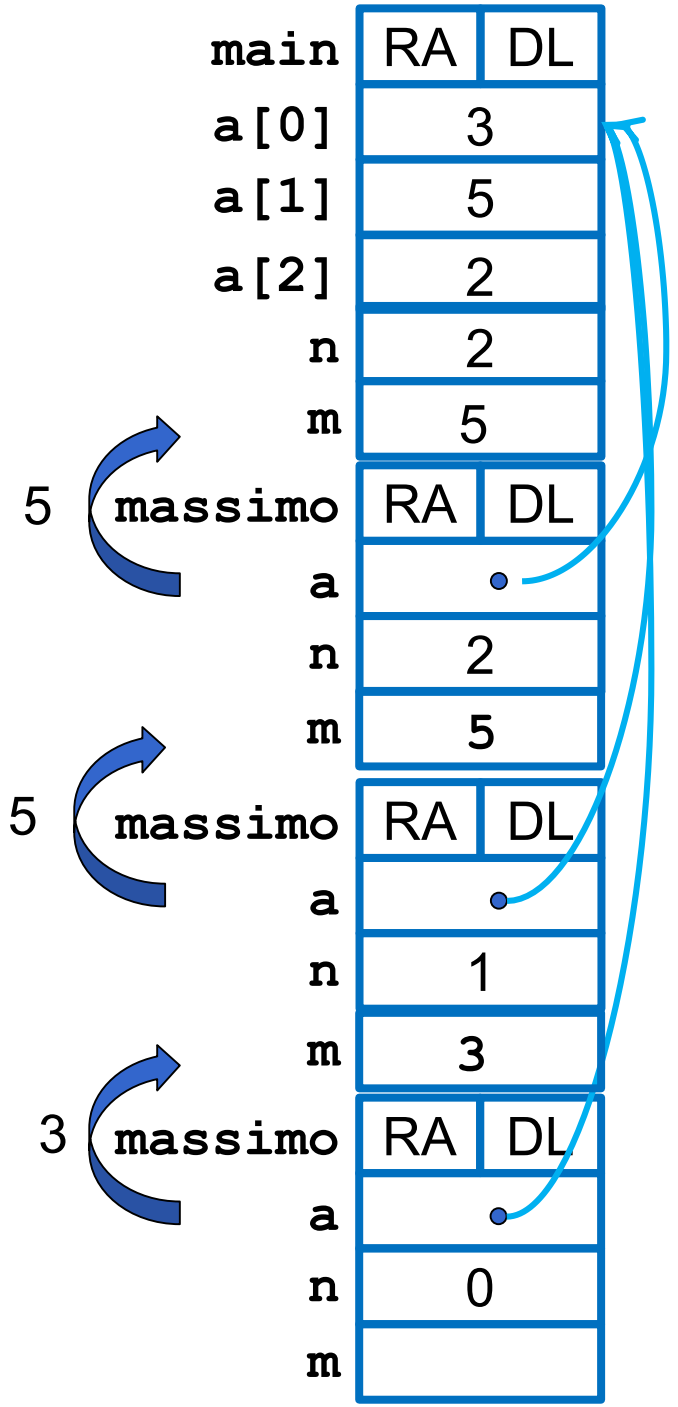
Massimo di un array

Si scriva una funzione ricorsiva

```
int massimo(int a[], int n);
```

che calcola il massimo degli elementi dell'array **a** dall'indice 0 fino all'indice **n**






```
int massimo(int a[], int n)
{ int m;
  if(n==0) return a[0];
  else
  { m = massimo(a,n-1);
    if (m<a[n])
       return a[n];
    else return m;
  }
}
main()
{ int a[4]={3,5,2}, n=2, m;
  m=massimo(a,n);
}
```

Capitolo 15 - la ricorsione

Ricerca binaria

Fondamenti di Informatica e Laboratorio - Modulo A

Corso di Laurea in Ingegneria Elettronica e Informatica

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

Ricerca in array ordinato

- Sapendo che il vettore è ***ordinato***, la ricerca può essere ottimizzata.
  - ***Vettore ordinato in senso crescente:***
    - Esiste una relazione d'ordine totale sul dominio degli elementi del vettore e:
    - Se `i<j` si ha `V[i]<V[j]`

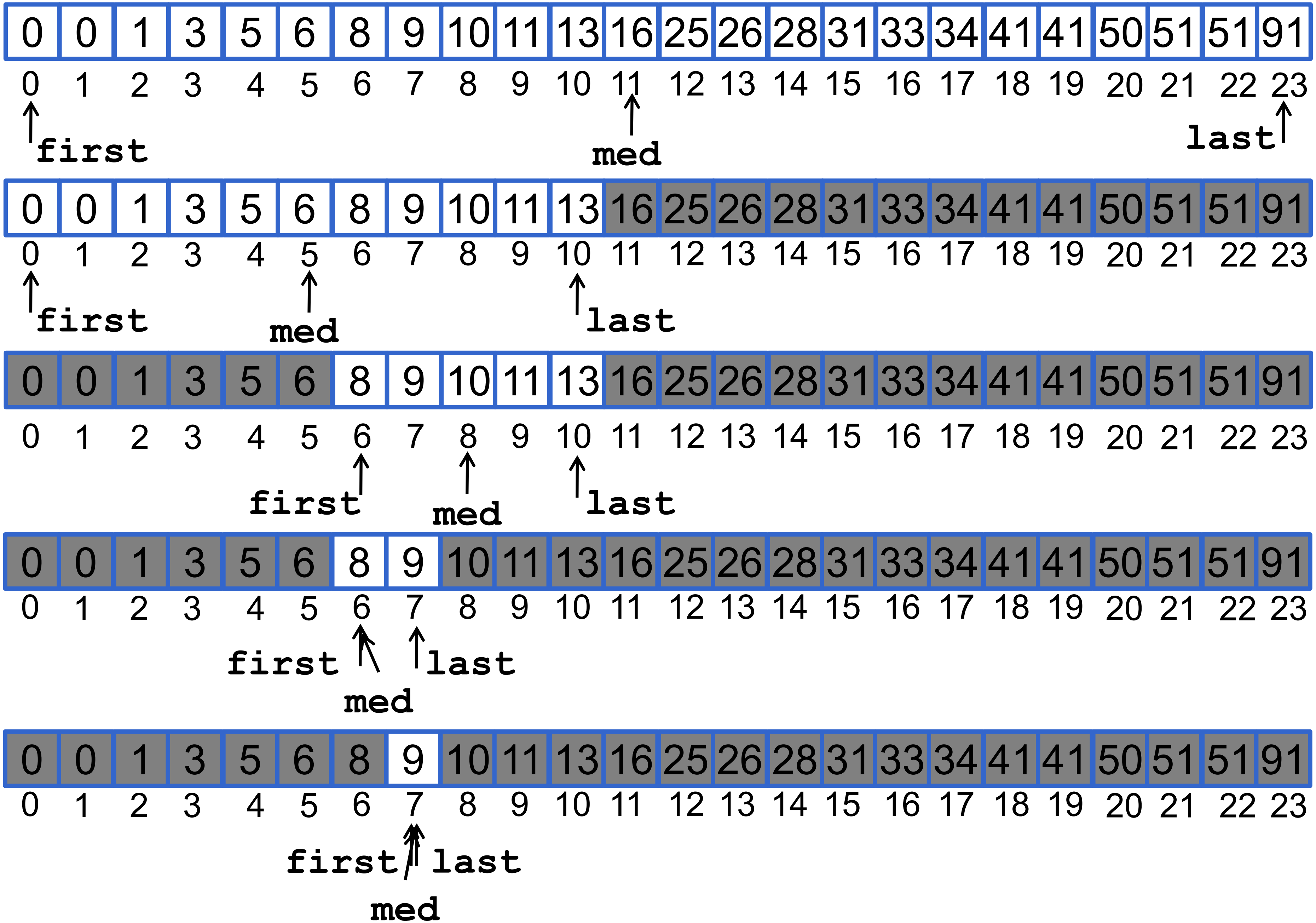
2	3	5	6	7	8	10	11

  - ***Vettore ordinato in senso non decrescente:***
    - Se `i<j` si ha `V[i]<=V[j]`

2	3	5	5	7	8	10	11

- In modo analogo si definiscono l'ordinamento in senso ***non crescente*** e ***decrescente***.

Si cerca il valore 9



Si cerca il valore **32**