Ricorsione

Informatica A AA 2024/2025

Giacomo Boracchi

15 Novembre 2024

giacomo.boracchi@polimi.it

Slide credits Prof. Alessandro Campi

Fattoriale – versione iterativa

```
int fattoriale (int n) {
   int f=1;
   for ( ; n > 0; n-- )
        f = f * n;
   return f;
}
```

Lo "spirito" del metodo ricorsivo

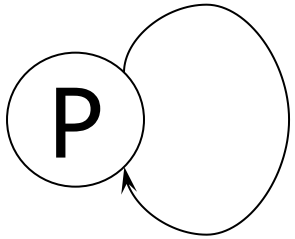
- Esiste un ***CASO BASE***, che rappresenta un sotto-problema facilmente risolvibile
  - Esempio: se **N=0**, so N! in modo "**immediato**" (vale 1)
- Esiste un ***PASSO INDUTTIVO*** che ci riconduce (prima o poi) al caso base
  - Consiste nell'esprimere la soluzione al problema (su dati di una "dimensione" generica) in termini di operazioni semplici e della soluzione allo **stesso** problema su dati "più piccoli" (che, per tali dati, **si suppone risolto *per ipotesi***)
    - Esempio: per **N *generico*** esprimo N! in termini di **N** (che è un dato direttamente accessibile) **moltiplicato per** (è una operazione semplice) il valore di **(N-1)!** (che so calcolare **per ipotesi induttiva**)

Fattoriale – versione ricorsiva

- 1) n! = 1 se n = 0
- 2) n! = n * (n - 1)! se n > 0
  - riduce il calcolo a un calcolo più semplice
  - ha senso perché si basa sempre sul fattoriale del numero più piccolo, che io conosco
  - ha senso perché si arriva a un punto in cui non è più necessario riusare la definizione 2) e invece si usa la 1)
  - 1) è il caso base, 2) è il passo induttivo

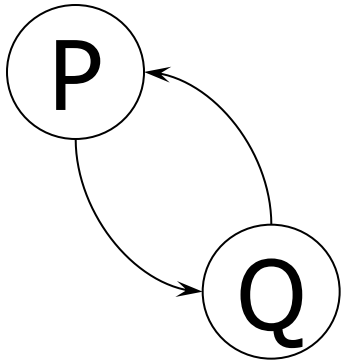
Ricorsione nei sottoprogrammi

- Dal latino re-currere
  - ricorrere, fare ripetutamente la stessa azione
- Un sottoprogramma P **invoca se stesso**
  - Direttamente
    - P invoca P



oppure

  - Indirettamente
    - P invoca Q che invoca P



Formulazione ricorsiva in C

```
int FattRic (int  n) {
        if (n == 0)
                return 1;
        else
                return n * FattRic(n–1);
}
```

Simulazione del calcolo

Invocazione di: FattRic(3)

3 = 0? No $\Rightarrow$ calcola fattoriale di 2 e moltiplica per 3

2 = 0? No $\Rightarrow$ calcola fattoriale di 1 e moltiplica per 2

1 = 0? No $\Rightarrow$ calcola fattoriale di 0 e moltiplica per 1

0 = 0? Si $\Rightarrow$ fattoriale di 0 è 1

$\Rightarrow$ fattoriale di 1 è 1 per fattoriale di 0, cioè $1 \times 1 = 1$

$\Rightarrow$ fattoriale di 2 è 2 per fattoriale di 1, cioè $2 \times 1 = 2$

$\Rightarrow$ fattoriale di 3 è 3 per fattoriale di 2, cioè $3 \times 2 = 6$

Esecuzione di funzioni ricorsive

- In un certo istante possono essere in corso ***diverse attivazioni*** dello **stesso** sottoprogramma
  - Ovviamente sono tutte sospese tranne una, l'ultima invocata, all'interno della quale si sta svolgendo il flusso di esecuzione
- Ogni attivazione esegue **lo stesso codice** ma opera su **copie distinte** dei parametri e delle variabili locali

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

val = 3
ris =

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

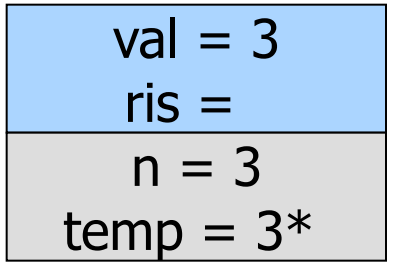
val = 3
ris =

temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

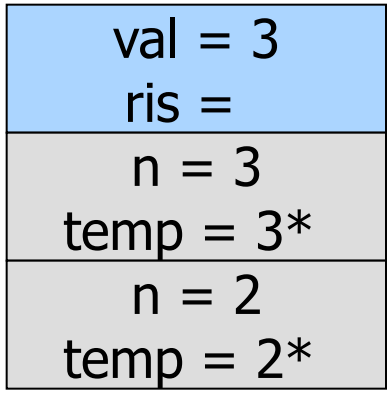


temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3



temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

val = 3 ris =
n = 3 temp = 3*
n = 2 temp = 2*
n = 1 temp = 1*

temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

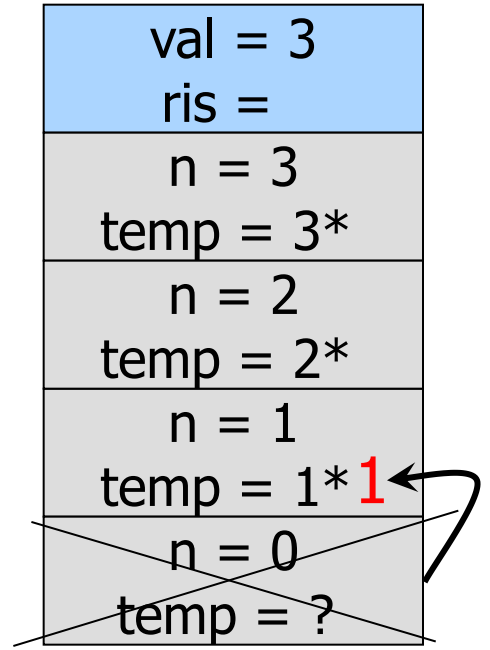
val = 3 ris =
n = 3 temp = 3*
n = 2 temp = 2*
n = 1 temp = 1*
n = 0 temp = ?

temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

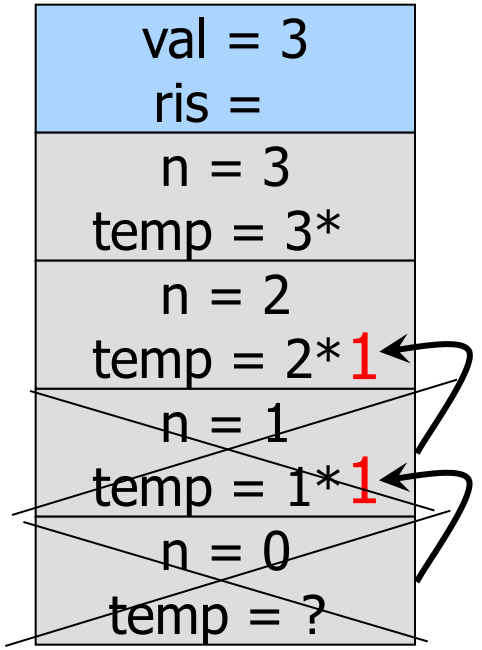


temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

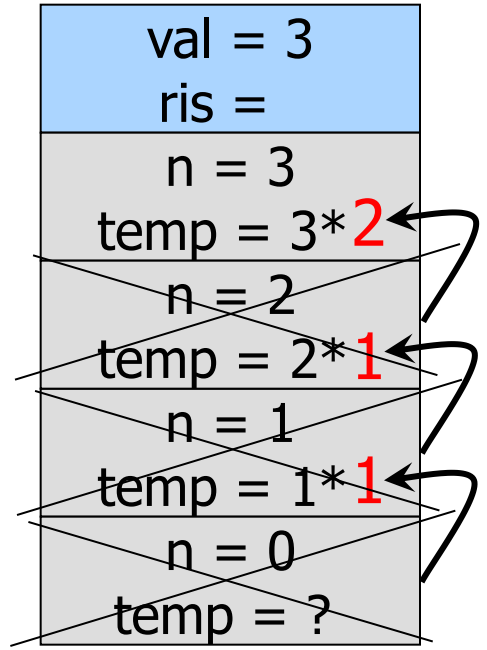


temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3

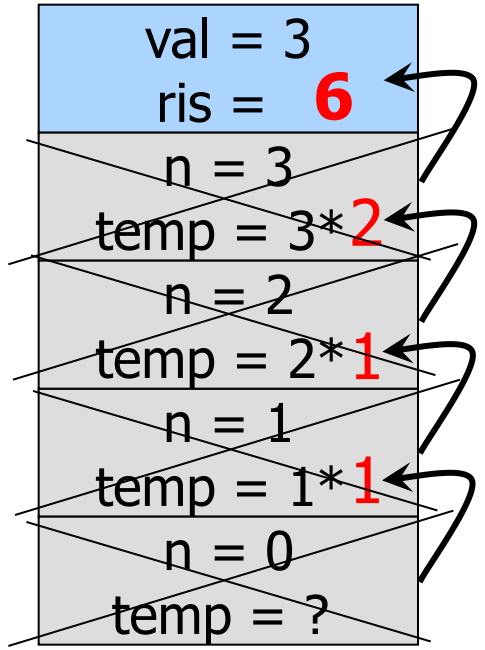


temp: cella temporanea per memorizzare il risultato della funzione chiamata

Il modello a runtime: esempio

```
int FattRic(int);

int main(){
        int val, ris;
        printf("dammi un naturale   ");
        scanf("%d", &val);
        ris = FattRic(val);
        printf("\nfattoriale= %d", ris);
        return 0;
}

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

assumiamo val = 3



temp: cella temporanea per memorizzare il risultato della funzione chiamata

Ma…

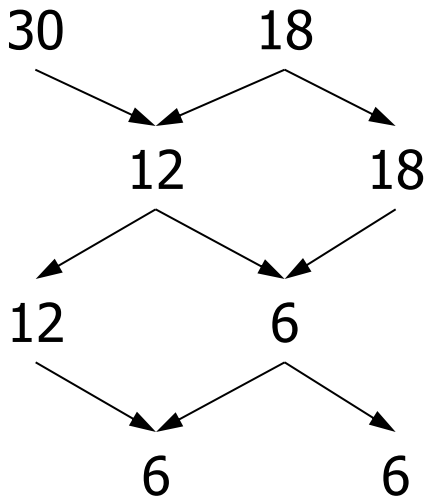
- … se ogni volta la funzione richiama se stessa… *perché la catena di invocazioni non continua* ***all'infinito****?*
- Quando si può dire che una ricorsione è ben definita?
- Informalmente:
  - Se per ogni applicazione del passo induttivo ci si avvicina alla situazione riconosciuta come caso base, allora la definizione non è circolare e la catena di invocazioni termina

Un altro esempio: MCD à-la-Euclide

- Il MCD tra M e N (M, N naturali positivi)
  - se M=N allora MCD è N
  - se M>N allora esso è il MCD tra N e M-N
  - se N>M allora esso è il MCD tra M e N-M

1 caso base

2 passi induttivi



MCD – versione iterativa

```
int EuclideIter (int m, int n) {
   while( m != n )
      if ( m > n )
         m = m – n;
      else
         n = n – m;
   return m;
}
```

MCD – versione ricorsiva

```
int Euclide (int m, int n) {
   if ( m == n )
      return n;
   if ( m > n )
      return Euclide(m – n, n);
   else
      return Euclide(m, n – m);
}
```

Funzione esponenziale (intera)

- Definizione iterativa:

  1) $x^y = 1$ se y = 0

  2) $x^y = x * x * \ldots x$ ***(y volte)*** se y > 0

- Definizione ricorsiva:

  1) $x^y = 1$ se y = 0

  2) $x^y = x * x^{(y-1)}$ se y > 0

Codice iterativo:

```
int esp (int x, int y) {
    int i, e = 1;
    for ( i = 1; i <= y; i++ )
        e = e * x;
    return e;
}
```

Codice ricorsivo:

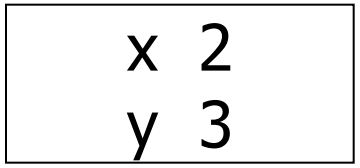
```
int esp (int x, int y) {
    if ( y == 0 )
        return 1;
    else
        return x * esp(x, y-1);
}
```

Attenzione ai parametri passati per indirizzo

```
/* incrementa il primo parametro
        del valore del secondo */
void incrementa(int *n, int  m)
{
        if (m != 0) {
                ( *n )++;
                incrementa(n, m–1);
        }
}
```

```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

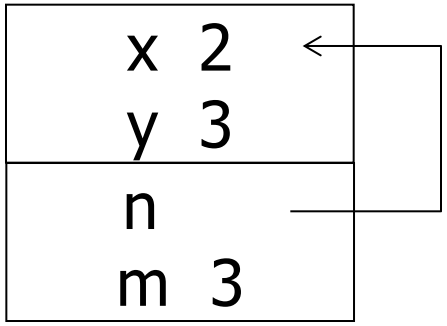
Modello a run-time




```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```

```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

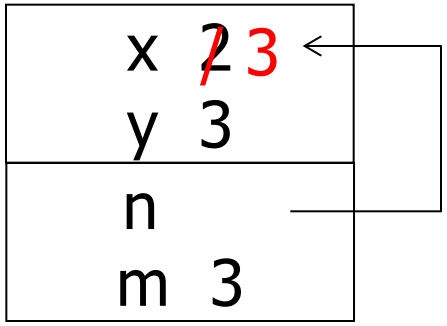
Modello a run-time




```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```

```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

Modello a run-time

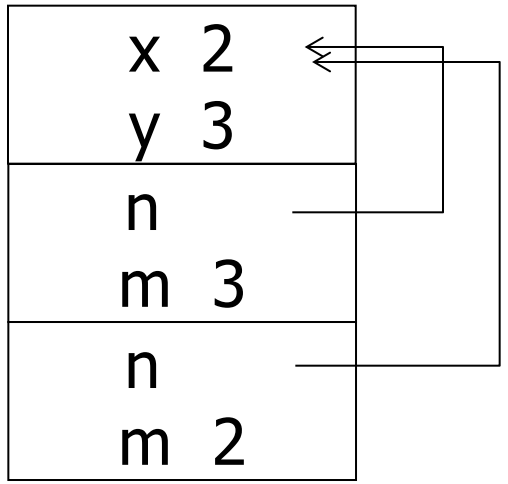



```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```

```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

Modello a run-time

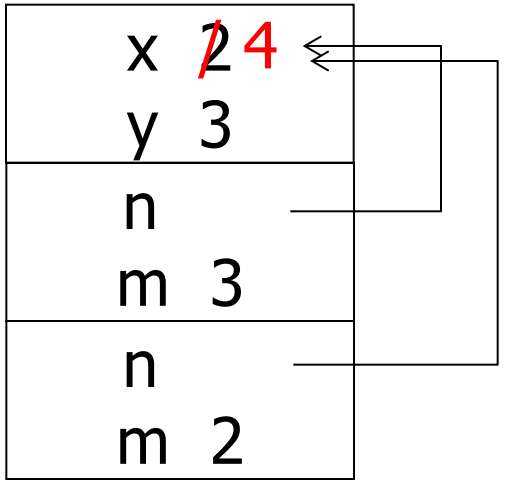
```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```




```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

Modello a run-time

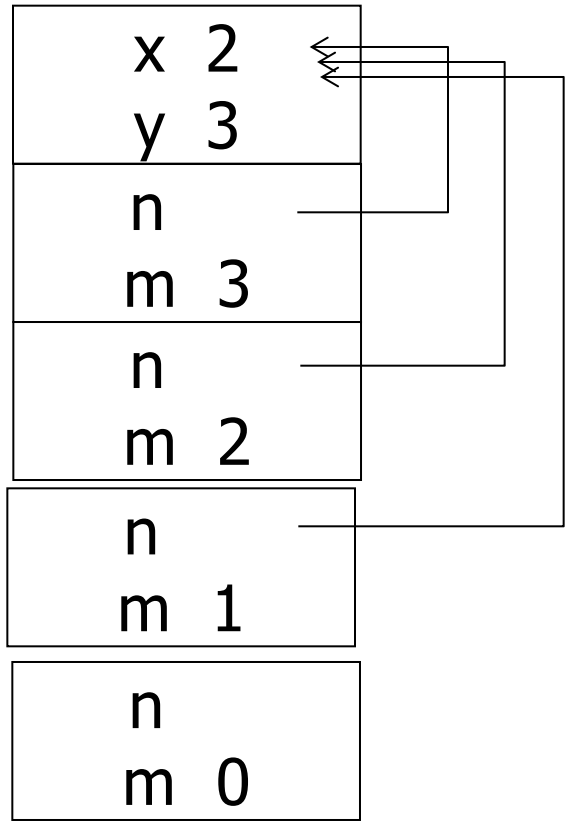
```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```




```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

Modello a run-time

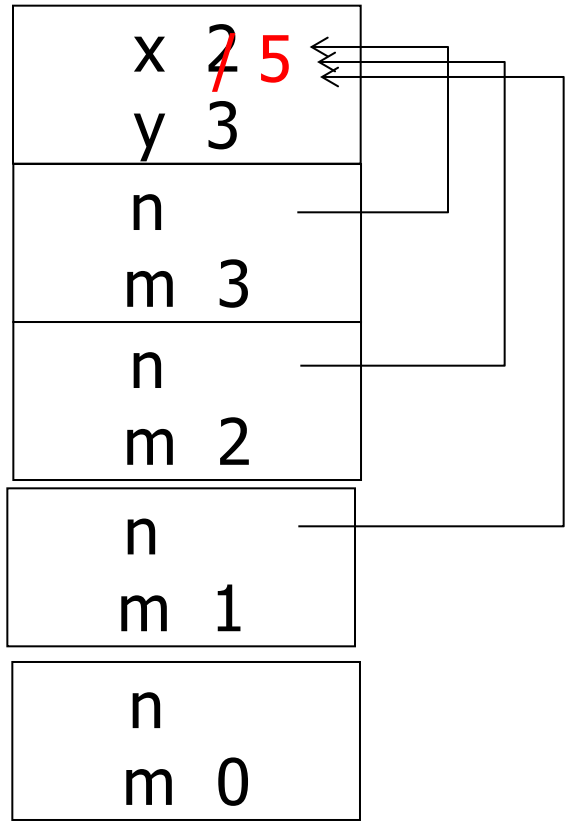
```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```




```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

Modello a run-time

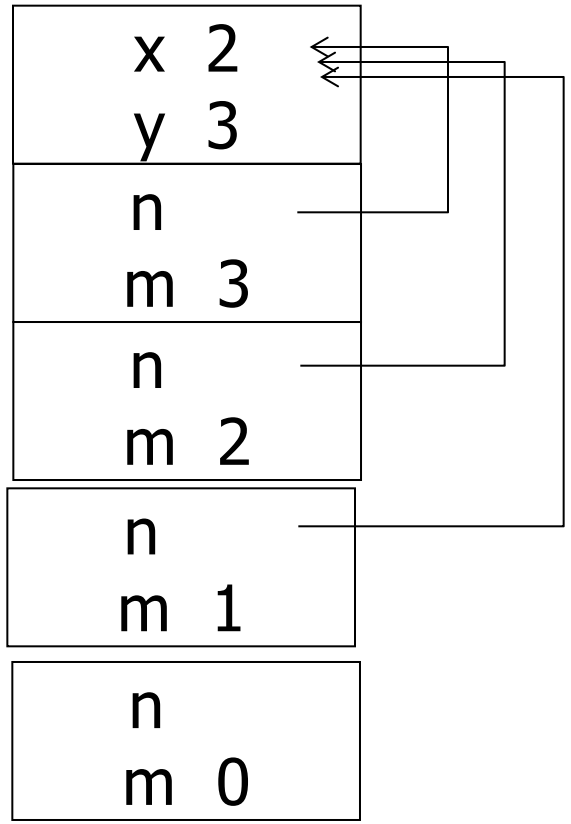
```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```




```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

Modello a run-time

```
void incrementa(int *n, int  m)
{
    if (m != 0) {
          ( *n )++;
          incrementa(n, m–1);
          }
}
```




```
int x, y;
...
x = 2;
y = 3;
incrementa(&x, y);

/* NB la chiamata iniziale
ha forma diversa dalla
chiamata ricorsiva */
```

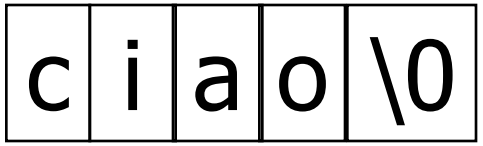
Terminazione (ancora!)

- Attenzione al rischio di ***catene infinite*** di chiamate
- Occorre che le chiamate siano soggette a una condizione che prima o poi assicura che la catena termini
- Occorre anche che l'argomento sia "progressivamente ridotto" dal passo induttivo, in modo da tendere prima o poi al caso base

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```

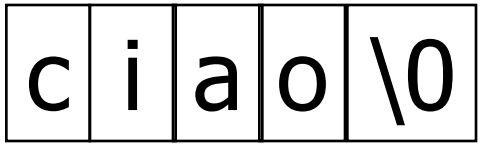


- Soluzione ricorsiva:

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



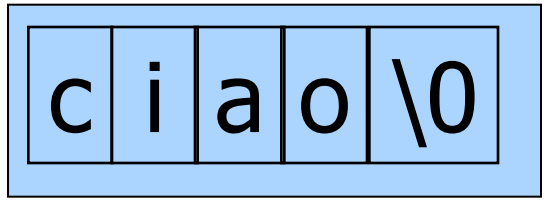
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



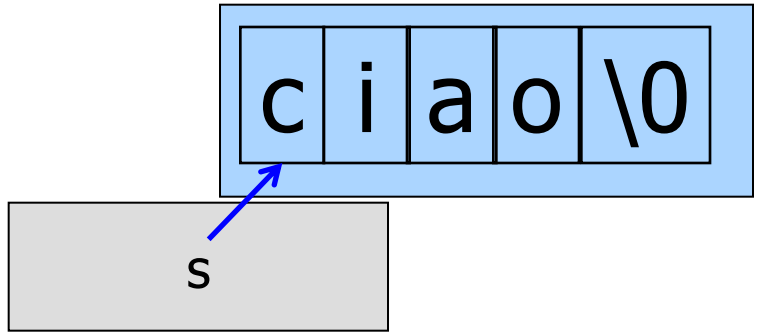
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



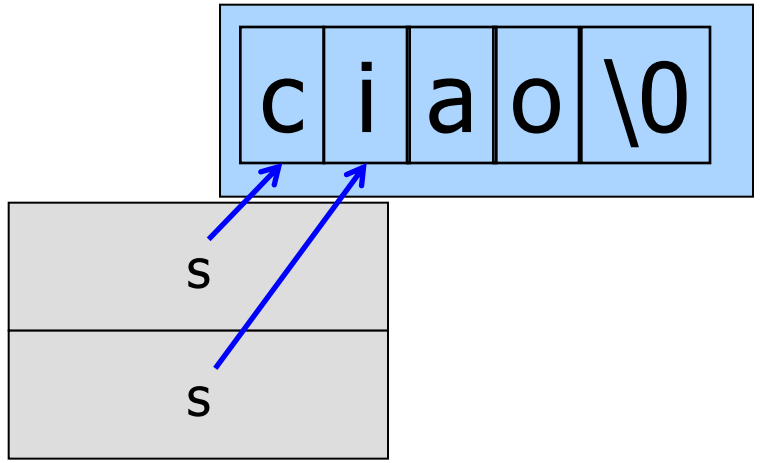
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



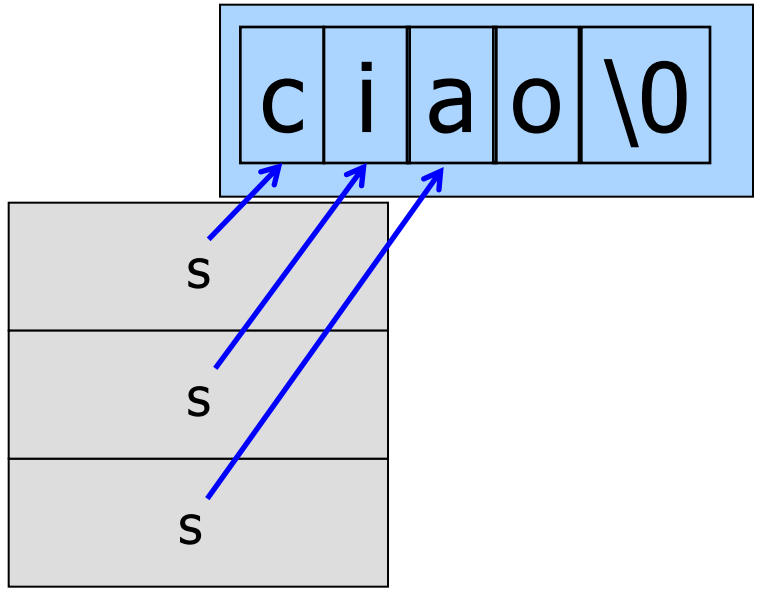
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



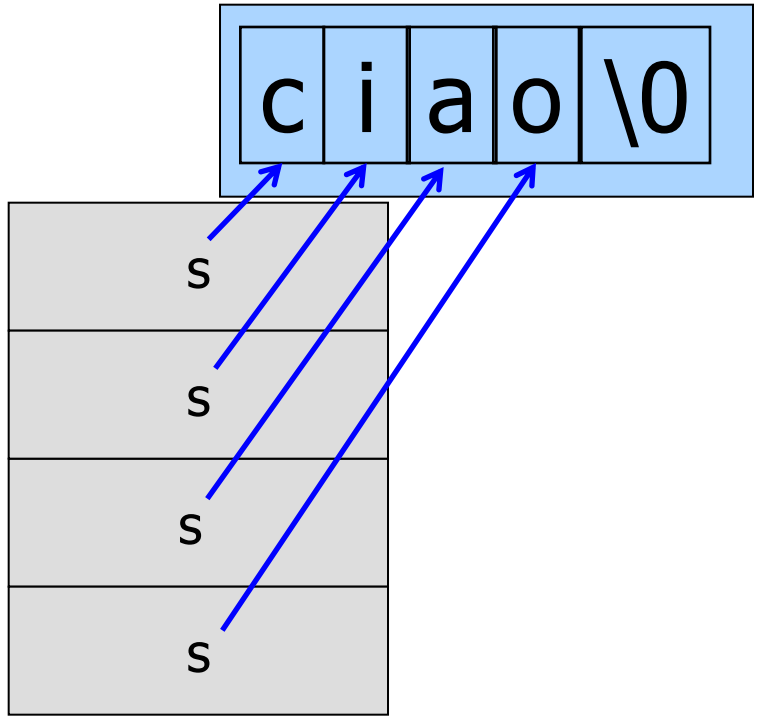
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

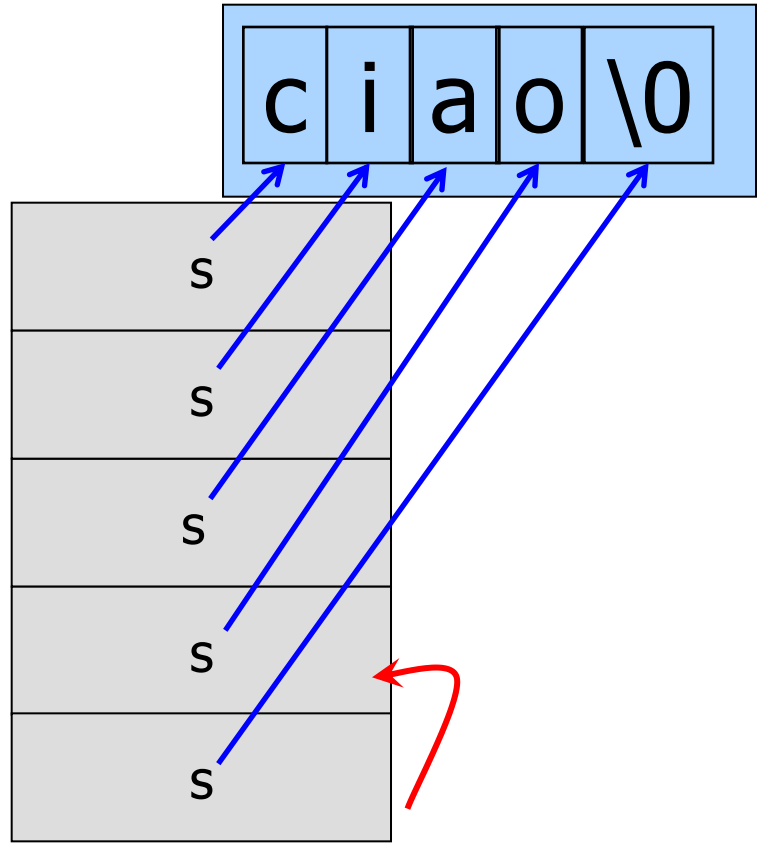
Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```

- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```



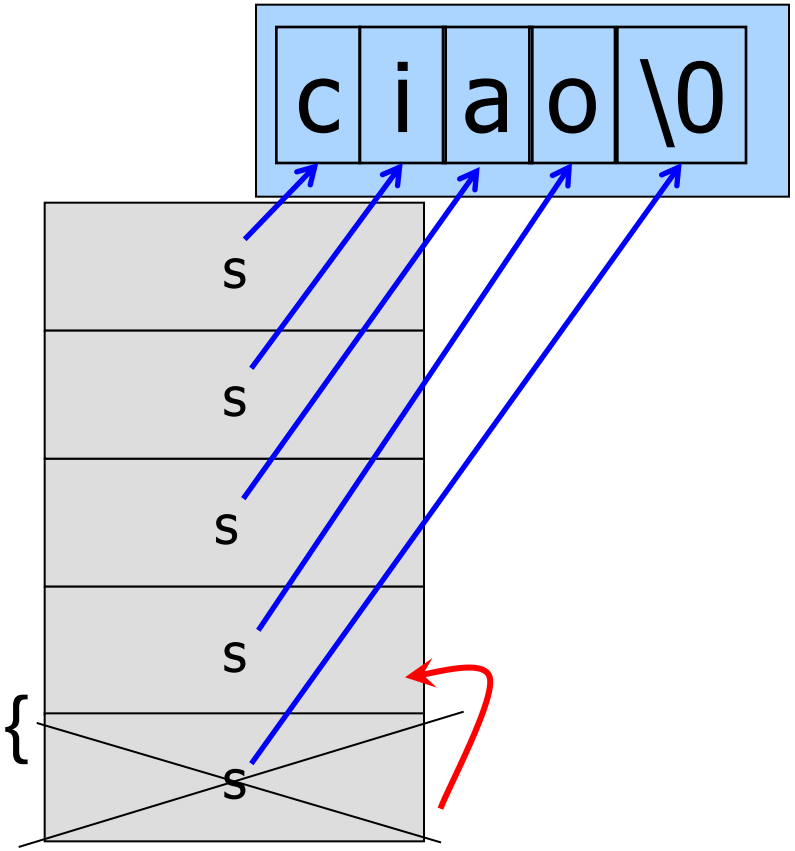
Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```

- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```



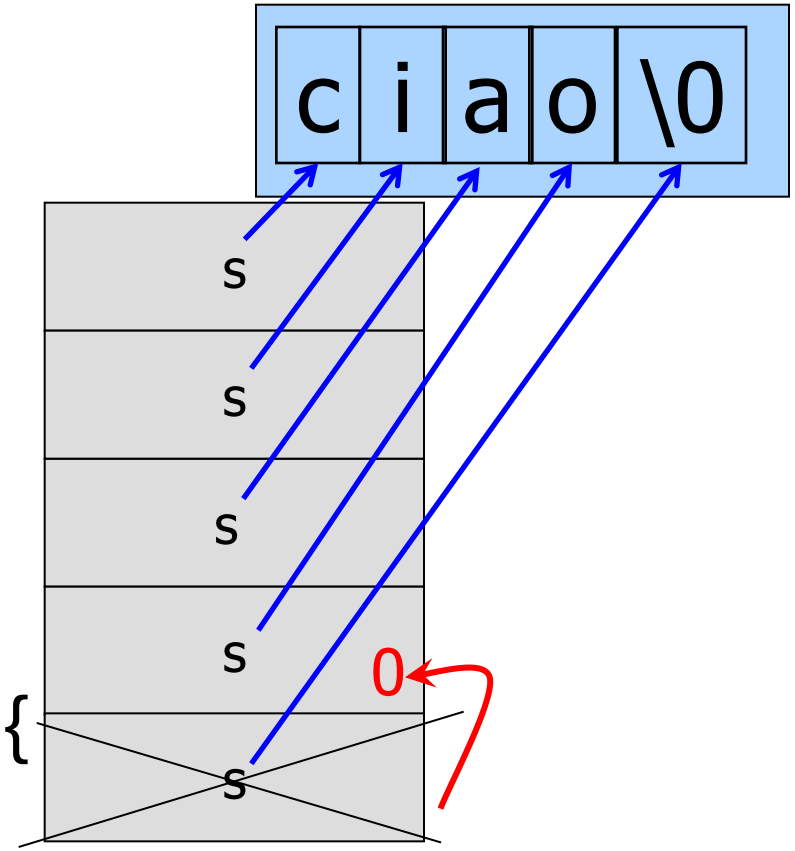
Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```

- Soluzione ricorsiva:

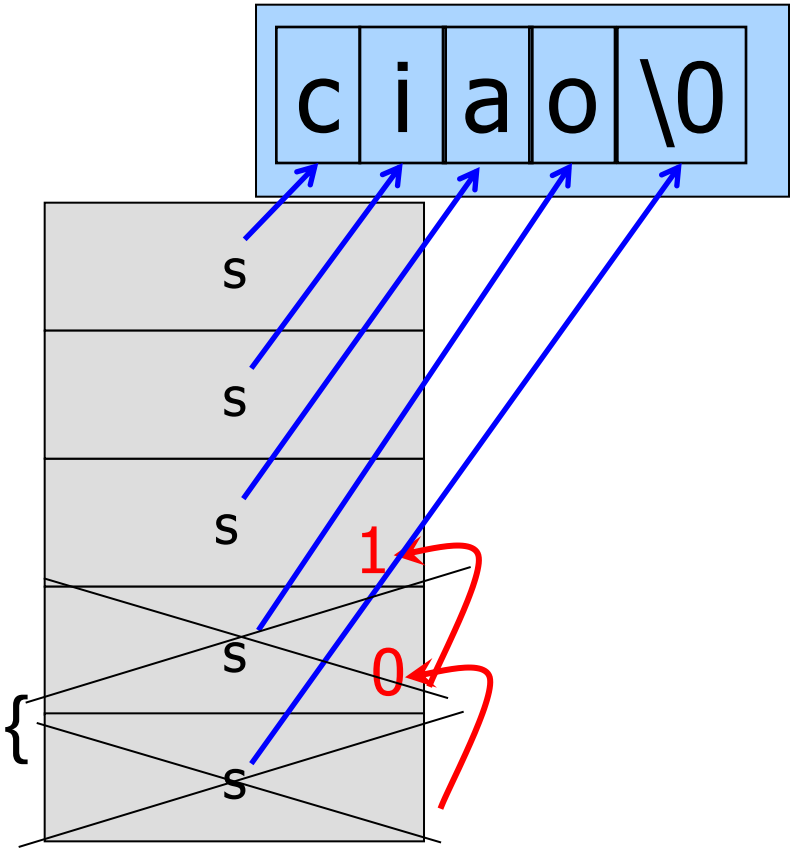
```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```



Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



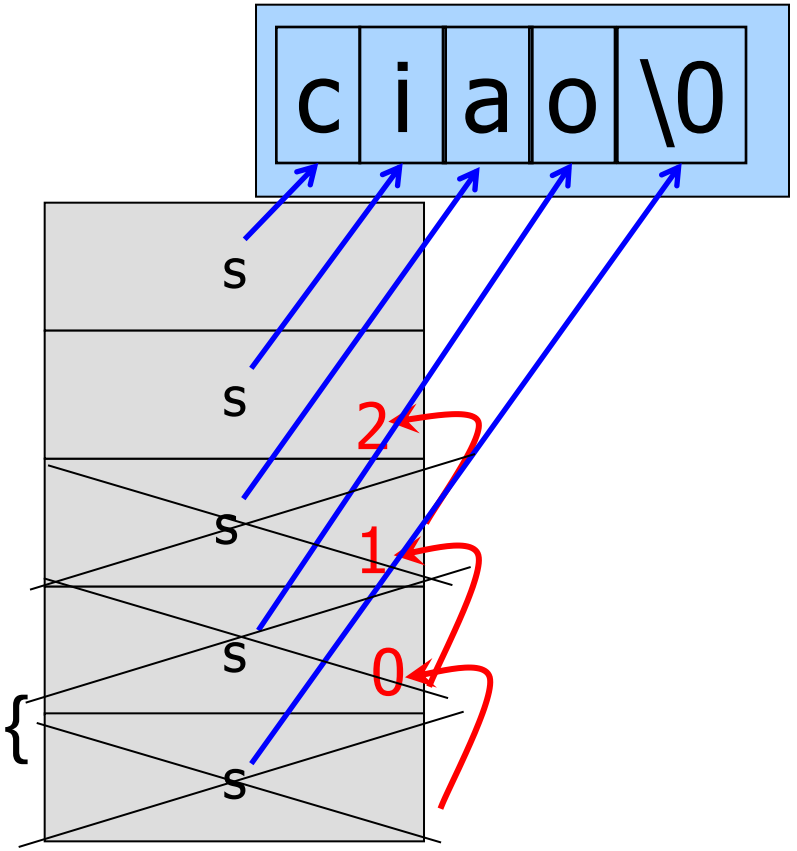
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



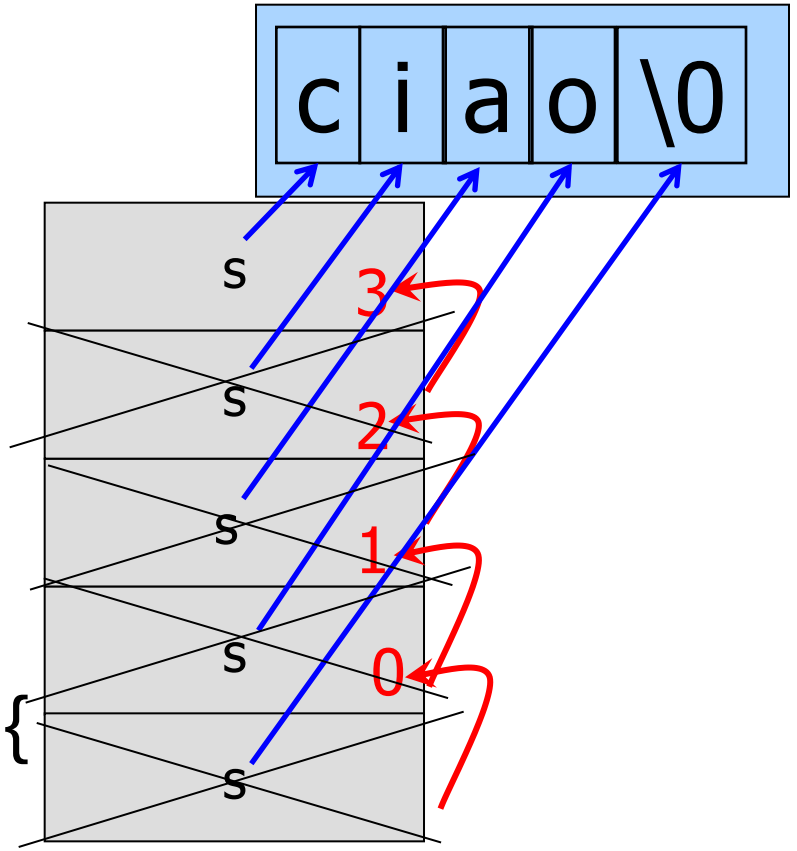
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



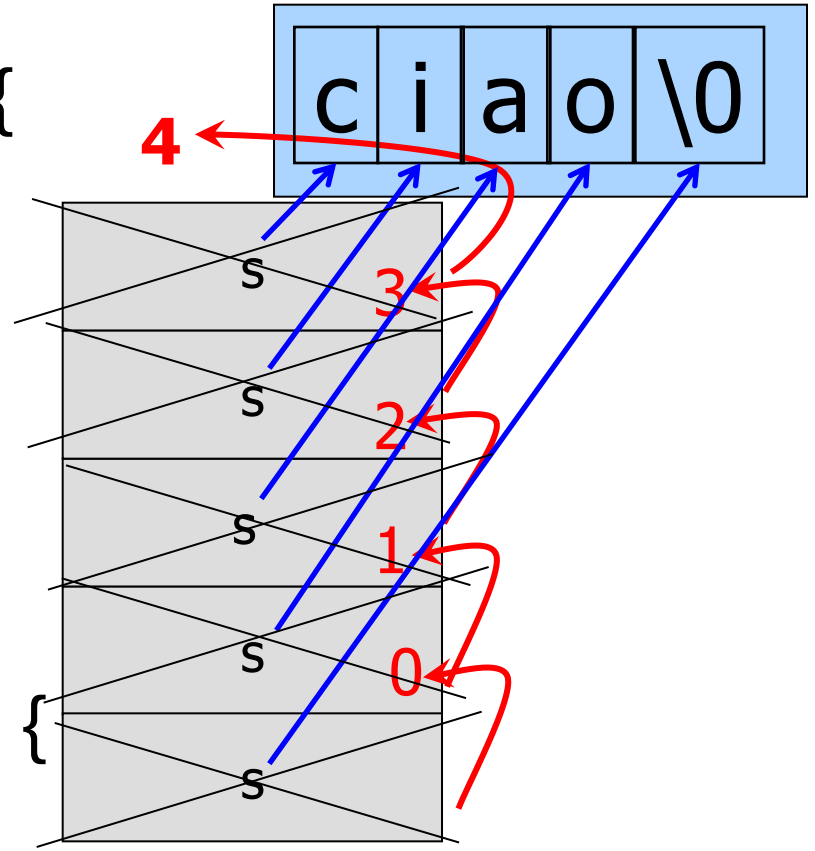
- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```



- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(&s[1]);
}
```

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```

- Soluzione ricorsiva:

```
int  lunghezzaR(char s[]) {
        if (s[0] == '\0')
                return 0;
        else
                return 1 + lunghezzaR(++s);
}
```

Soluzione analoga

Esempio: lunghezza stringhe

- Soluzione iterativa:

```
int  lunghezzaI(char s[]) {
        int i = 0;
        while ( s[i] != '\0' )
                i++;
        return i;
}
```

- S

E’ sbagliato perché con il post-incremento non sposta mai s prima dell’invocazione e invoca sempre la funzione sullo stesso carattere di inizio. Quindi il programma da luogo ad una serie di chiamate infinite

```
        else
                return 1 + lunghezzaR(s++);
}
```

Esempio: inversione stringa

- Scrivere una funzione che calcoli la stringa inversa di una stringa data
- Il chiamante passa anche il puntatore alla stringa che conterrà l'inversa
- Si restituisce la stringa inversa
- Servirsi di
  - **strcpy(char *dest, char *src)**

    (copia la stringa src nella stringa dest)
  - **strncat(char *dest, char *src, int n)**

    (incolla n caratteri di src in coda a dest)

Esempio: inversione stringa

```
char * inversione(char s[], char t[])
{
   // caso base
   if (strlen(s) == 1)  return strcpy(t,s);

   // passo induttivo
   return strncat( inversione(&s[1],t), s, 1);
}
```

Le parole palindrome

- Una parola è un *palindromo* (dal greco: παλιν-, *ancora*, *indietro*, *di nuovo* e -δρομος, *corsa*, *percorso*) se la si può leggere indifferentemente da destra a sinistra e viceversa
- (ovviamente) tutte le parole di un solo carattere sono considerate palindrome
  - "a" -> sì
  - "db" -> no (anche se è graficamente simmetrica…)
  - "Anna" -> no (l'analisi è case sensitive)
  - "anno" -> no
  - "anilina" -> sì
  - "onorarono" -> sì
  - "saippuakivikauppias" -> sì (*venditore di liscivia*, in finlandese)

Le Parole Palindrome

Escludendo gli spazi:

- «etna gigante»
- «ai lati d’italia»
- «o mordo tua nuora o aro un autodoromo»
- «i topi non avevano nipoti»

Esercizio

- Si scriva un programma che memorizza in un array di caratteri una parola letta da stdin e verifica se la parola è o non è palindroma
- **int palindromo(char parola[], ...)**
  - Restituisce 1 o 0
  - Si chiede, in particolare, di farne una versione iterativa e una versione ricorsiva
- **Versione iterativa**: confronto tra tutte le coppie di lettere simmetriche rispetto al "centro"
  - (attenzione, la parola può avere un numero pari o dispari di caratteri)
- **Versione ricorsiva**: un palindromo è tale se...

Palindromi in versione iterativa

```
int palindromoI( char par[] )
{
  int da = 0;                          // assegna gli indici del primo e ultimo
  int a  = strlen( par ) - 1;          // carattere della parola

  while( da < a ) {                    // scandisce la parola facendo muovere
    if ( par[da] != par[a] )           // gli indici verso il centro di par
      return 0;                        // e dice FALSE alla prima differenza
    ++da; --a;
  }
  return 1;                            // se arriva alla fine senza differenze, TRUE
}
```

Palindromi in versione ricorsiva

Un palindromo è tale se:

- la parola è di lunghezza 0 o 1; **Caso base**

  oppure

- il primo e l'ultimo carattere della parola sono uguali **e inoltre** la sotto-parola che si ottiene ignorando i caratteri estremi è a sua volta un palindromo

Passo induttivo

Il passo induttivo riduce la ***dimensione*** *del problema!*

Palindromi in versione ricorsiva

```
int palindromoR( char par[], int da, int a ) {
  if ( da >= a )          // applica la definizione
    return 1;
  else
    return (  par[da] == par[a] &&
              palindromoR( par, da+1, a-1 )  );
}
```

Attenzione

- Come si fa la prima chiamata?
- Si suppone che il chiamante conosca (o calcoli) la lunghezza della stringa.
  - Esempio: palindromoR("oro", 0, 2)
  - oppure: palindromoR(str, 0, strlen(str)-1)
- oppure ancora…

...introduciamo un'altra funzione che fa0 da **wrapper** per avviare la prima invocazione

```
int palindromo(char s[]) {
    return palindromoR(s, 0, strlen(s)-1);
}
```

Palindromi ricorsivi: variante

```
int palindromoR( char par[], int da, int a )
{
  if ( da >= a )          // applica la definizione
    return 1;
  else
    return   ( palindromoR( par, da+1, a-1 ) &&
                 par[da] == par[a]);
}
```

Qual è la differenza?

Ancora stringhe palindrome

- Stringa palindroma se:
  - è la stringa vuota, oppure
  - ha un solo carattere, oppure
  - è una stringa palindroma racchiusa tra due caratteri, primo e ultimo, uguali

int Palindrome(char PC[], char UC[]);

```
#define LunghMaxStringa  100

int  Palindrome(char PC[], char UC[]);

int main() {
      char str[LunghMaxStringa];
      int LunghStr;
      scanf("%s", str);  /* NB assumiamo non ci siano spazi */
      LunghStr = strlen(str);
      if ( LunghStr == 0 )
            printf("La stringa è palindroma");
      else {
            printf("La stringa");
            if ( ! Palindrome( &str[0], &str[LunghStr–1] ) )
                  printf(" NON");
            printf(" è palindroma\n");
      }
      return 0;
}
```

```
int Palindrome(char *PC, char *UC)
{
   if ( PC >= UC ) /* stringa vuota o di 1 carattere */
      return 1;
   if (*PC != *UC)  /* primo e ultimo car. diversi */
      return 0;
   else
      return Palindrome( PC+1, UC–1 );
                /* chiamata ricorsiva escludendo
                   primo e ultimo carattere */
}
```

Osservazioni sul programma

- Legge una stringa (termina con '\0')
- Ne calcola la lunghezza con strlen()
- Se stringa vuota o di un carattere $\Rightarrow$ palindroma
- Se caratteri agli estremi diversi $\Rightarrow$ NON lo è
- Altrimenti applica la funzione Palindrome alla stringa privata degli estremi
  - elegante uso di due puntatori ($\Rightarrow$ indirizzi del primo e dell'ultimo carattere della parte non ancora esaminata dell'array)
  - spostati avanti e indietro a ogni chiamata ricorsiva

Altri tipi di palindromi (1)

- Palindromi **a parola**:

  "Fall leaves after leaves fall"

- Palindromi **a frase** (ignorando spazi e punteggiatura):

  "I topi non avevano nipoti"

  "Avida di vita, desiai ogni amore vero, ma ingoiai sedativi, da diva"

  "Sun at noon, tan us!"

  – Molti esempi notevoli:
    - G. Perec, "9691" (> 5000 caratteri)
      – "Trace l'inégal palindrome. Neige [...] ne mord ni la plage ni l'écart."
    - G. Varaldo, "11 Luglio 1982" (> 4000 caratteri)
      – Ai lati, a esordir, dama [...] a Madrid, rosea Italia!
    - Batman, "Una storia italiana" (> 1000 caratteri)
      – O idolo, se vero, mal onori parole [...] rapirono l'amore, v'è sol odio.

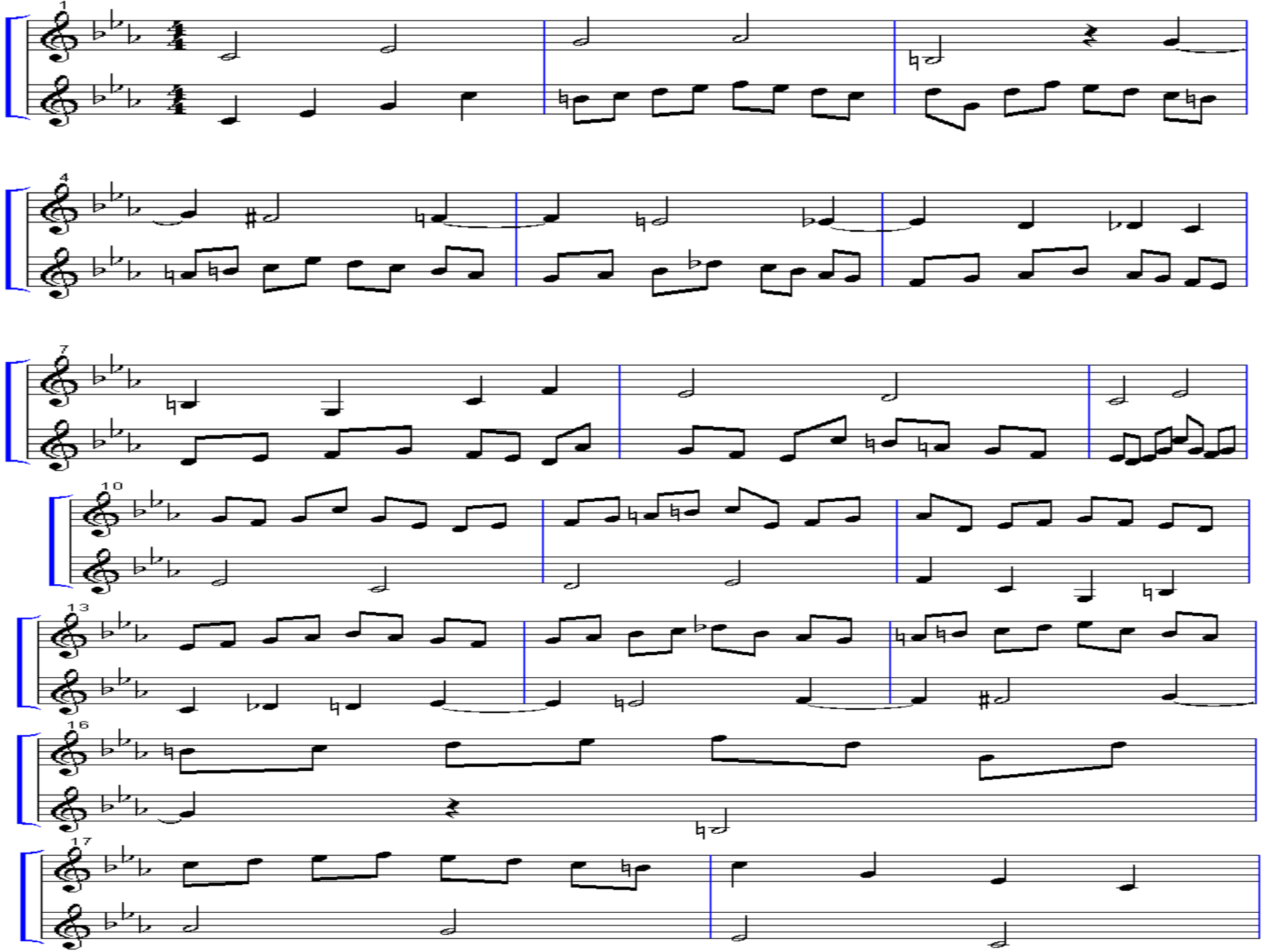
Altri tipi di palindromi (2)

- Palindromi **a riga**
  - J. A. Lindon, "Doppelganger"

    "Entering the lonely house with my wife,
    I saw him for the first time
    peering furtively from behind a bush
    […]
    Peering furtively from behind a bush,
    I saw him, for the first time
    entering the lonely house with my wife."
- Esercizio: implementare funzioni di verifica palindromi a riga, a frase, a parola

Digressione: palindromi ovunque

- I palindromi esistono anche in matematica (numeri palindromi)
- …in pittura…
- …in musica…
  - J. S. Bach ha scritto un "canone cancrizzante" a due voci
    - Scambiando la prima e la seconda voce, e leggendo la partitura da sinistra a destra, si ottiene ancora lo stesso brano



Ricorsione o iterazione?

- Spesso le soluzioni ricorsive sono eleganti
- Sono vicine alla *definizione* del problema
- Però possono essere inefficienti
- Chiamare un sottoprogramma significa allocare memoria a run-time

N.B. è **sempre** possibile trovare un corrispondente iterativo di un programma ricorsivo

Un altro esempio: la serie di Fibonacci

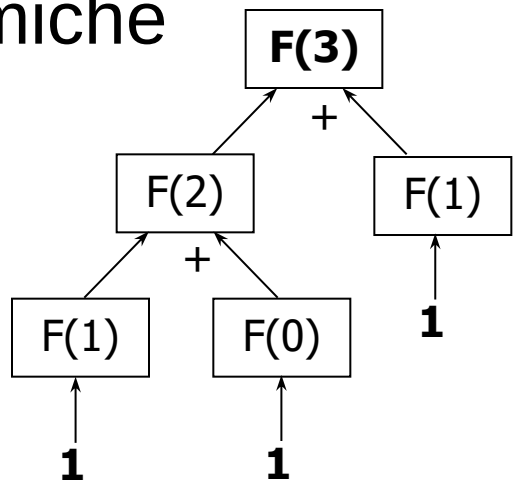
- Fibonacci (1202) partì dallo studio sullo sviluppo di una colonia di conigli in circostanze ideali

  - Partiamo da una coppia di conigli
  - I conigli possono riprodursi all'età di un mese
  - Supponiamo che dal secondo mese di vita in poi, ogni femmina produca una nuova coppia
  - e inoltre che i conigli non muoiano mai…

  – Quante coppie ci sono dopo n mesi?

Definizione ricorsiva della serie

- I numeri di Fibonacci
  - Modello a base di molte dinamiche evolutive delle popolazioni



$F = \{f_0, \ldots, f_n\}$

- $f_0 = 1$
- $f_1 = 1$

} casi base (2!)

- Per $n > 1$, $f_n = f_{n-1} + f_{n-2}$

} 1 passo induttivo

Notazione "funzionale": $F(i) = f_i$

Numeri di Fibonacci in C

```
int Fibo (int  n) {
   if (n == 0 || n == 1)
      return 1;
   else
      return (Fibo (n-1) + Fibo(n-2));
}
```

Ovviamente supponiamo che n>=0

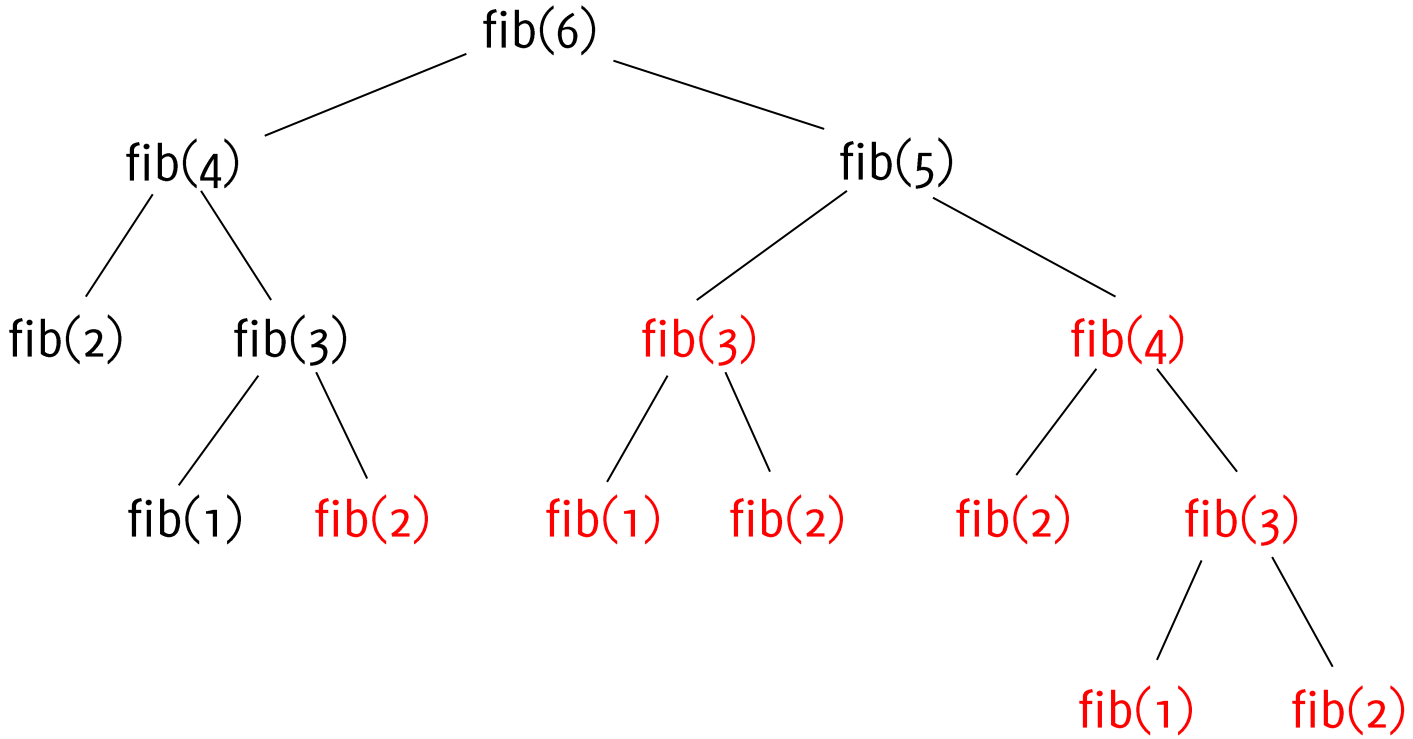
Calcolo numeri di fibonacci

```
int fibonacci (int  n) {
        if (n == 0 || n == 1)
                return 1;
        else  return (    fibonacci(n–1) +
                          fibonacci(n–2)     );
}
```

Drammaticamente inefficiente!
Calcola più volte l'i-esimo numero di fibonacci!

Ricorsione Eccessiva

Soluzione elegante ma dispendiosa: numero esorbitante di chiamate ricorsive



Provate a far calcolare fib(5), poi fib(10), fib(15), fib(20), fib(25), fib(30), ...

Quante volte viene calcolato fab(3)????

Meglio usare una soluzione non ricorsiva...

Numeri di Fibonacci e memoization

- La prima volta che calcolo un dato numero di Fibonacci lo memorizzo in un array
- Dalla seconda volta in poi, anziché ricalcolarlo, lo leggo direttamente dall'array
- Mi occorre un valore "sentinella" con cui inizializzare l'array che mi indichi che il numero di Fibonacci corrispondente non è ancora stato calcolato
  - Qui posso usare ad esempio 0

Inizializzazione dell'array memo

```
#define MAX 100
main() {
    int i, n;
    long memo[MAX];
    for (i=2; i < MAX; i++)        // inizializzazione
        memo[i]=0;
    memo[0] = 1;
    memo[1] = 1;  // casi base
    printf("Un intero: ");
    scanf("%d",&n);
    printf("fib(%d) = %d", n, fib(n,memo));
}
```

Calcolo con memorizzazione

```
long fib(int n,long memo[]) {
    if (memo[n] != 0)
        return memo[n];
    memo[n] = fib(n-1,memo) + fib(n-2,memo);
    return memo[n];
}
```

Drastica riduzione della *complessità* (aumento di efficienza)
Questa soluzione richiede un tempo *lineare* in n
La soluzione precedente richiede un tempo *esponenziale* in n

Ricerca Binaria

- Scrivere un programma C che implementi l'algoritmo di ricerca dicotomica in un vettore ordinato in senso crescente, con procedimento ricorsivo.
- Dato un valore ***"f"*** da trovare e un vettore ***"array"*** con due indici ***"i,l"***, che puntano rispettivamente al primo e ultimo elemento; L'algoritmo di ricerca dicotomica prevede che se l'elemento f non è al centro del vettore cioè in posizione "***m = (i+l)/2"*** allora deve essere ricercato necessariamente soltanto in uno dei due sottovettori a destra o a sinistra dell'elemento centrale.

- Se ***i > l***, allora l'elemento cercato ***f*** non è presente nel vettore **(caso base)**
- Se ***(f == array [(i+l) / 2])***, allora ***f*** è presente nel vettore. **(caso base)**
- Altrimenti **(passo induttivo)**
  - Se ***(f > array[(i+l) / 2])*** allora la ricerca deve continuare nel sottovettore individuato dagli elementi con indici nell'intervallo ***[m +1, l]***
  - Se ***(f < array[(i+l) / 2])*** allora la ricerca deve continuare nel sottovettore individuato dagli elementi con indici nell'intervallo ***[i, m - 1]***

```c
#include<stdio.h>
#define N 10

int ricercaBinaria(int *, int, int, int, int*);

int main()
{
    int vet[N] = {1, 5, 7, 8, 10, 12, 45, 56, 67, 78};
    int n, len, pos = -1, found;
    len = N - 1;

    printf("inserire elemento da cercare: ");
    scanf("%d", &n);

    found = ricercaBinaria(vet, 0, len, n, &pos);

    printf("\nil vettore contiene %d: %d (pos %d)", n, found, pos);
}
```

```
int ricercaBinaria(int vet[], int a, int b, int x, int *pos)
{
    int m;
    if(x == vet[a])
    {   *pos = a;
        return 1;
    }
    if(x == vet[b])
    {  *pos = b;
       return 1;
    }
    m = (b + a) / 2;
    if(x == vet[m])
    {   *pos = m;
        return 1;
    }
    if(a > b)
        return 0;
    if(x > vet[m])
       return ricercaBinaria(vet, m + 1, b, x, pos);
    else
       return ricercaBinaria(vet, a, m - 1, x, pos);
    }
}
```

Salvo anche la posizione con un passaggio per riferimento **pos**

```
int ricercaBinaria(int vet[], int a, int b, int x, int *pos)
{
     int m;
     if(x == vet[a])
     {    *pos = a;
          return 1;
     }
     if(x == vet[b])
     {   *pos = b;
         return 1;
     }
     m = (b + a) / 2;
     if(x == vet[m])
     {    *pos = m;
          return 1;
     }
     if(a > b)
          return 0;
     if(x > vet[m])
         return ricercaBinaria(vet, m + 1, b, x, pos);
     else
         return ricercaBinaria(vet, a, m - 1, x, pos);
     }
}
```

Per finalità di debug posso aggiungere il seguente comando prima di ogni return

```
printf("\na = %d, b = %d, *pos = %d", a, b, *pos);
```

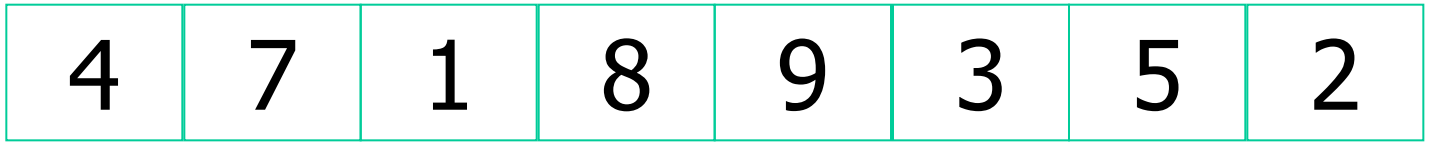
Lo stesso meccanismo..

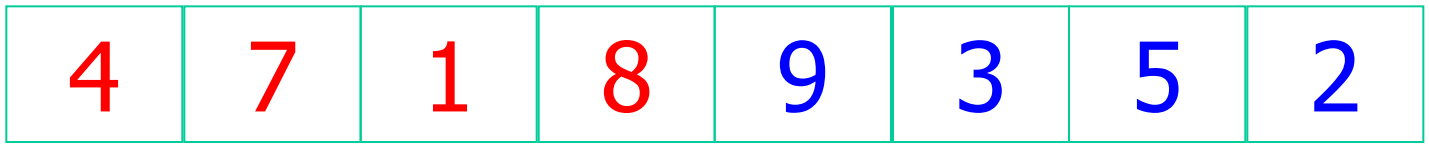
La ricerca binaria si può usare ad esempio, per:

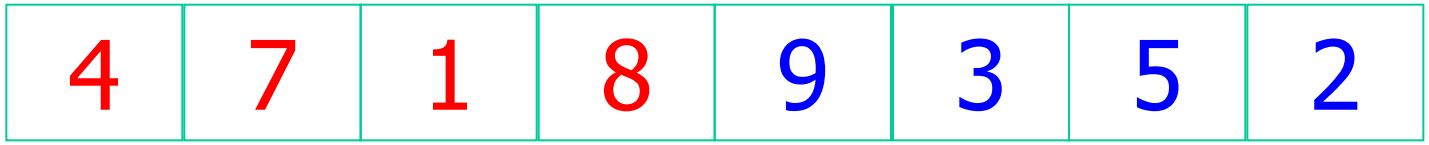
- eseguire la ricerca di una scheda (struttura) in uno schedario (array di strutture)
- Calcolare (in maniera approssimata) uno zero di una funzione continua in un intervallo. Si usi il teorema di Bolzano per capire se la funzione avrà almeno uno zero.
- Etc...

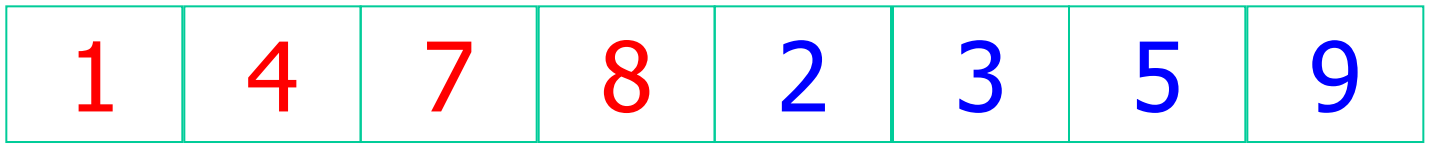
Mergesort

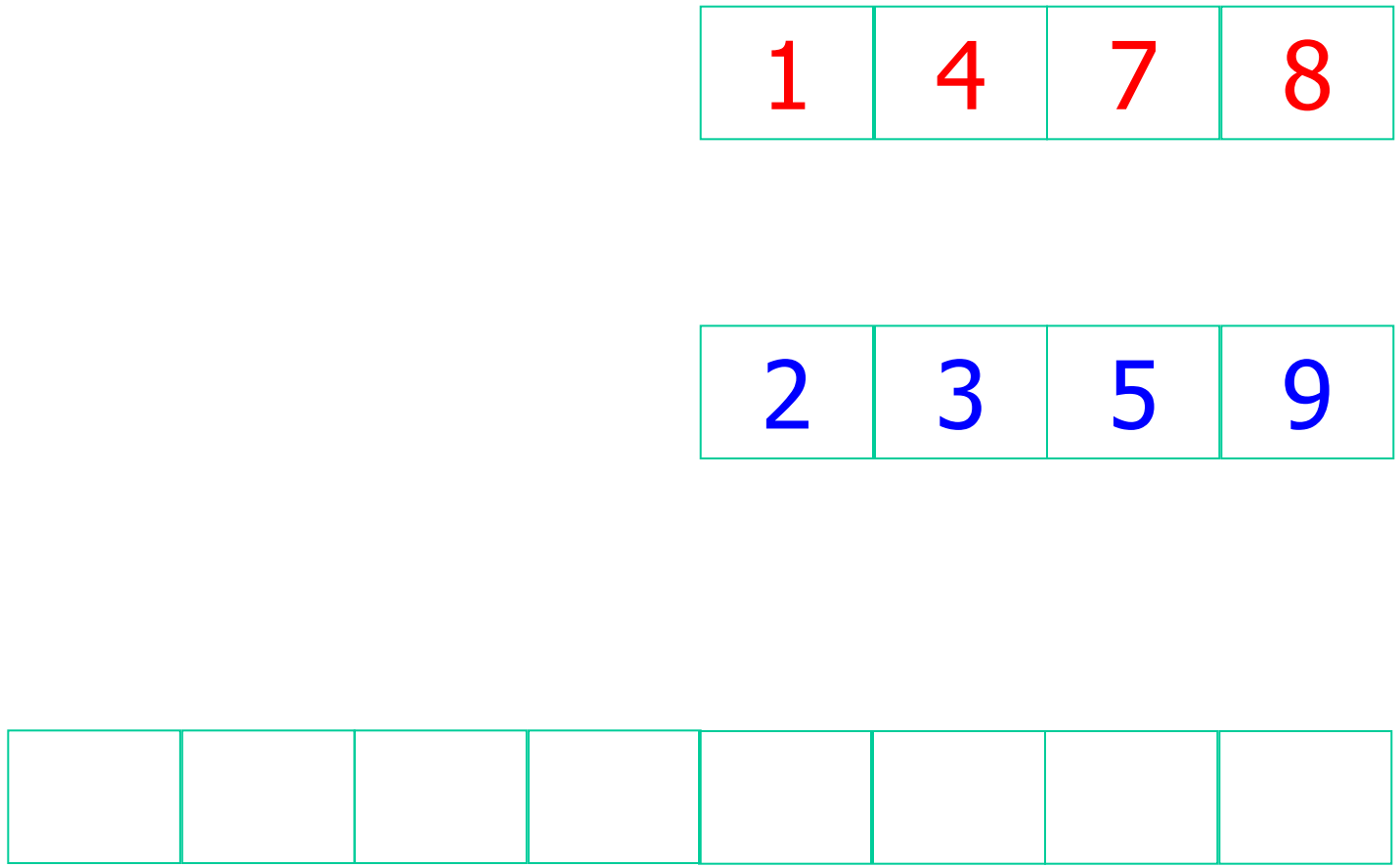
- Scrivere una funzione f che riceve in input un array di interi e un intero che ne rappresenta la dimensione e che ordina l’array

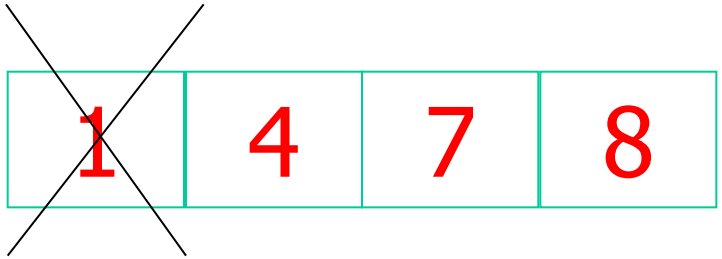

4 7 1 8 9 3 5 2

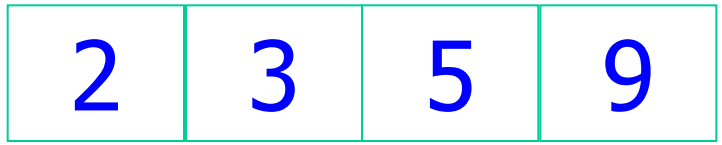

4 7 1 8 9 3 5 2

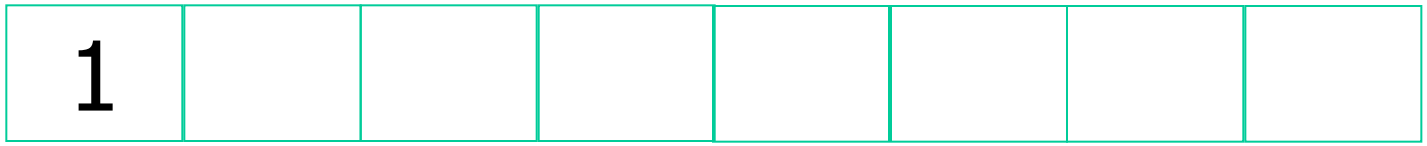

4 7 1 8 9 3 5 2

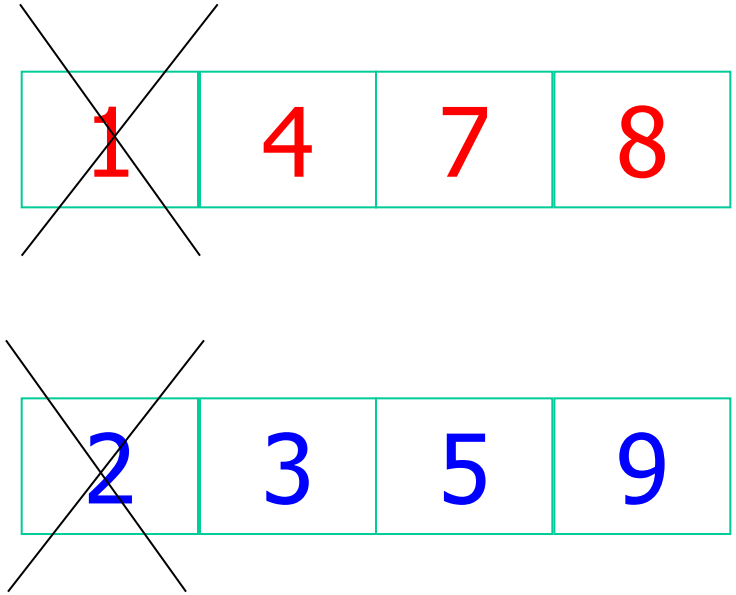

1 4 7 8 2 3 5 9


1 4 7 8
2 3 5 9

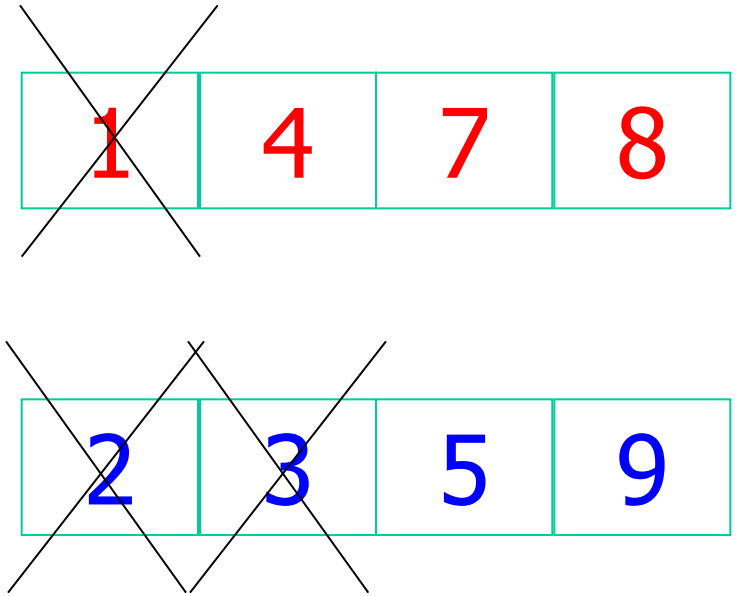

1
4
7
8


2
3
5
9

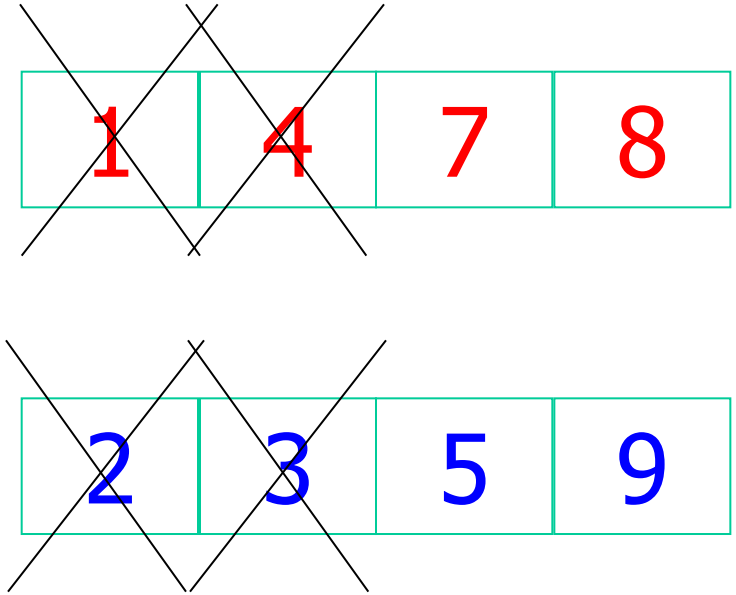

1



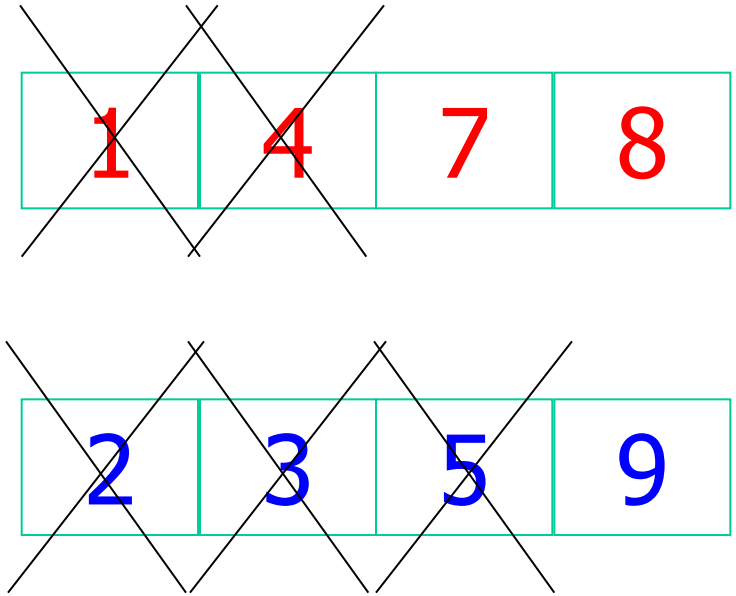
1	2						



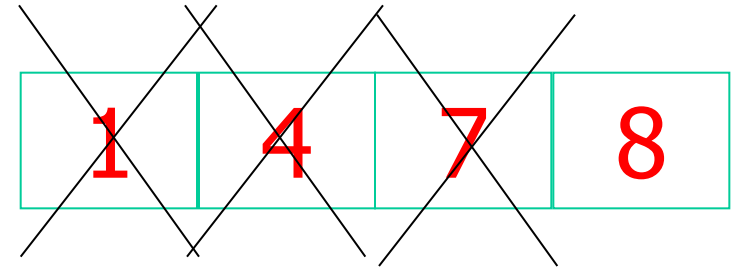
1	2	3					

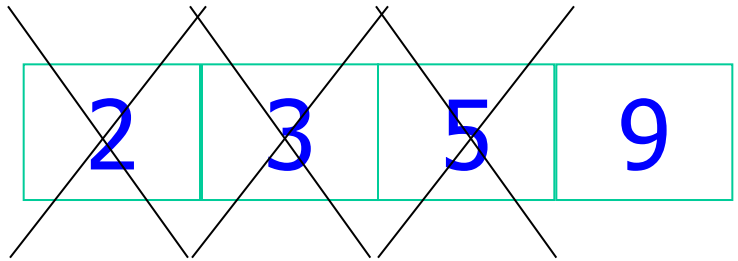


1	2	3	4				

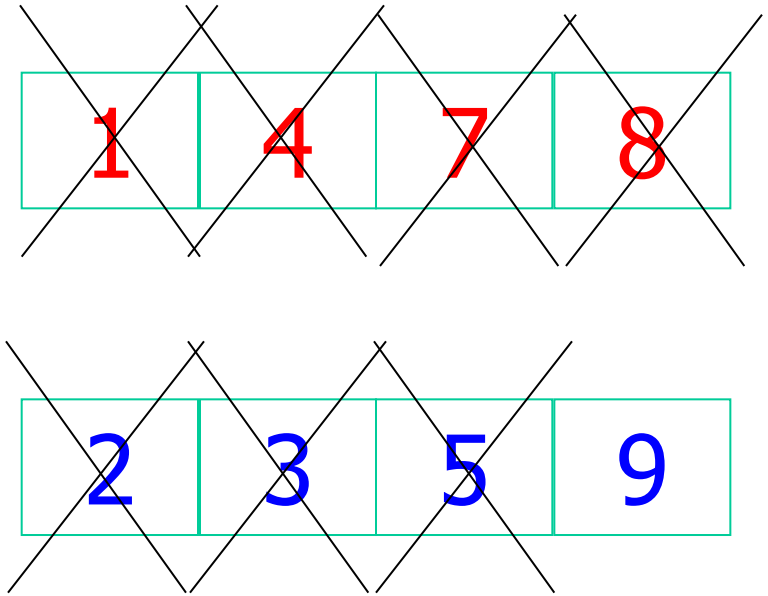


1	2	3	4	5			

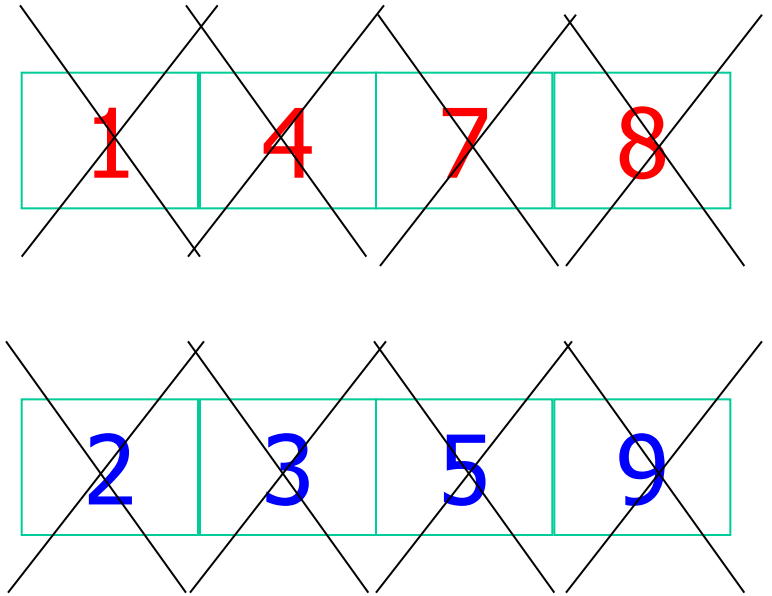




1	2	3	4	5	7		

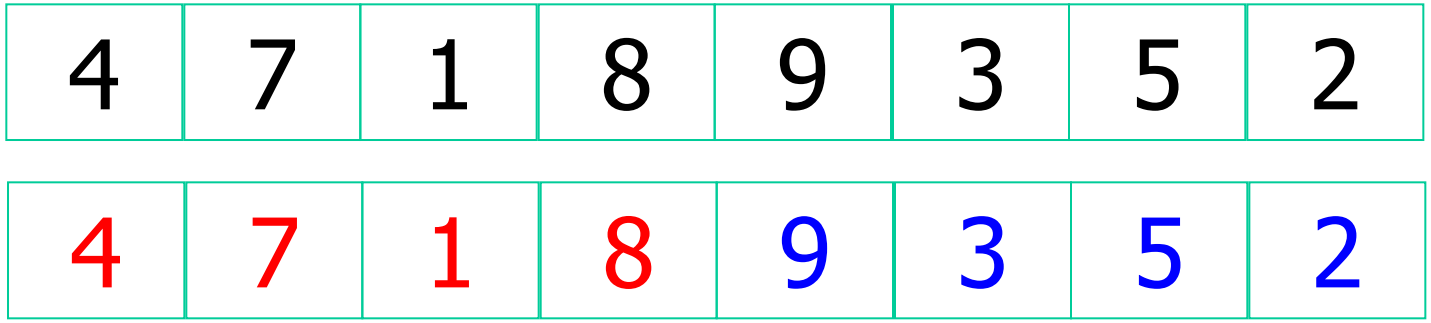


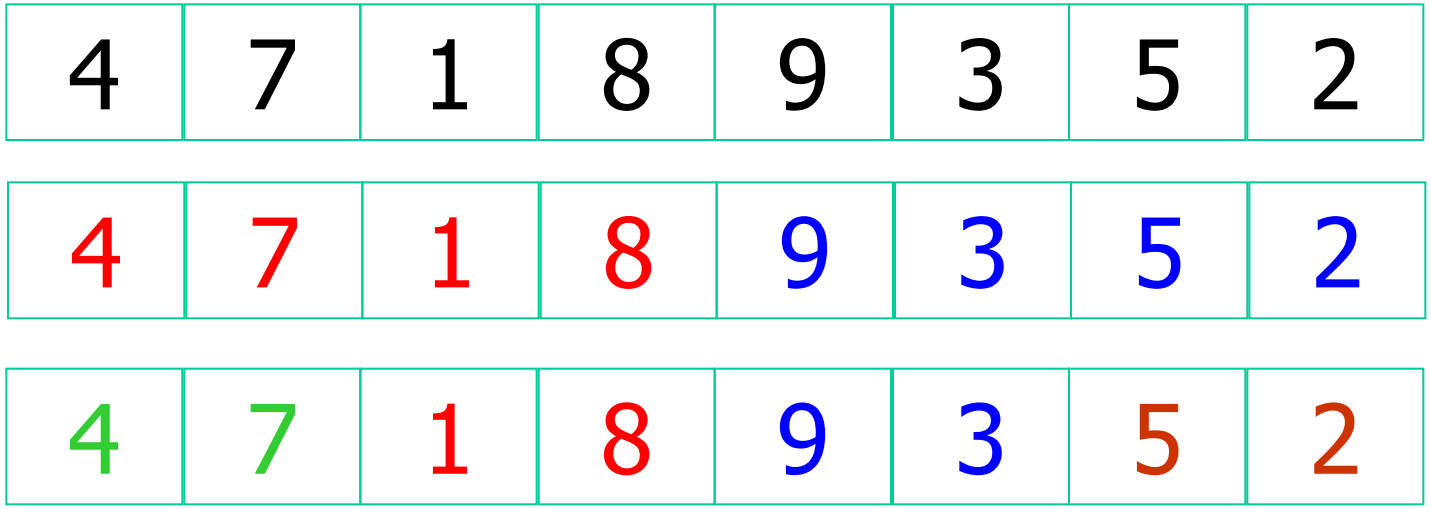
1	2	3	4	5	7	8	



1	2	3	4	5	7	8	9

4	7	1	8	9	3	5	2


4 7 1 8 9 3 5 2
4 7 1 8 9 3 5 2


4 7 1 8 9 3 5 2
4 7 1 8 9 3 5 2
4 7 1 8 9 3 5 2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	3	9	2	5

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	3	9	2	5

1	4	7	8	2	3	5	9

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	9	3	5	2

4	7	1	8	3	9	2	5

1	4	7	8	2	3	5	9

1	2	3	4	5	7	8	9

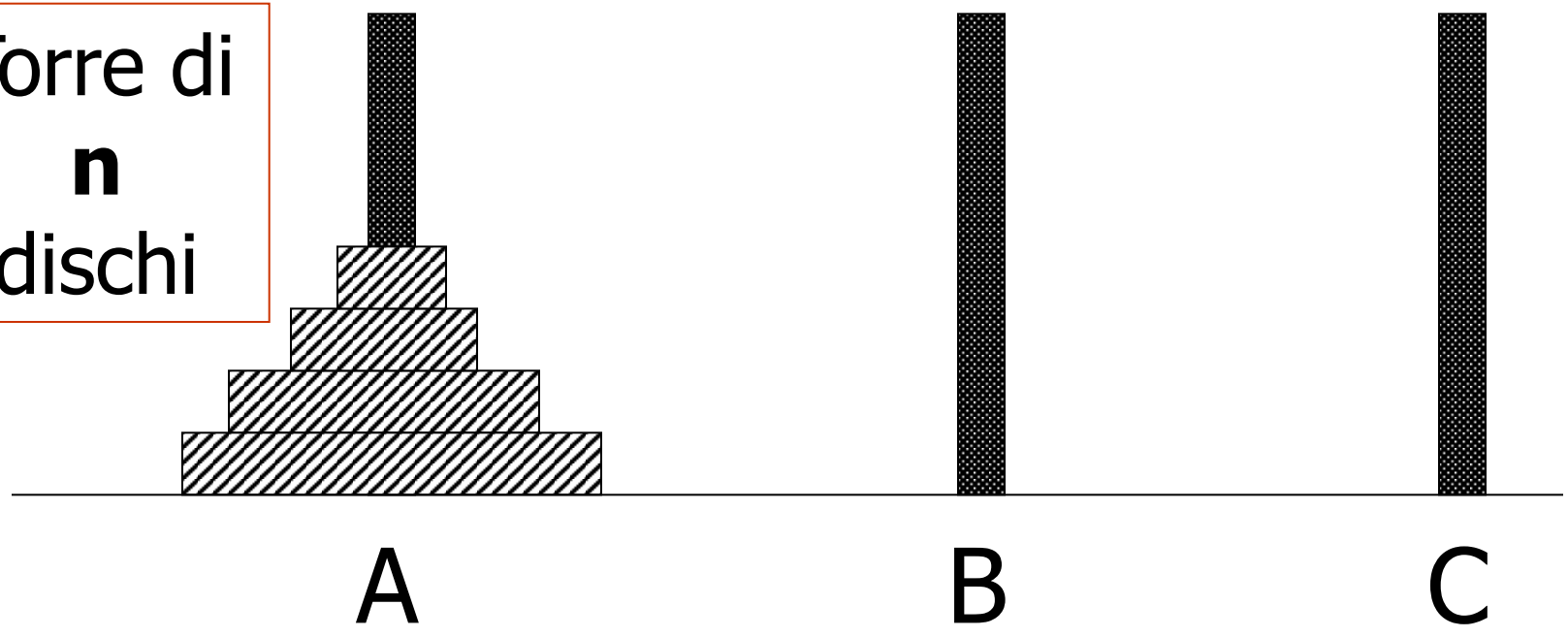
```
void ordina(int a[],int N) {
     mergesort(a,0,N-1);
}
void mergesort(int a[],int low,int high) {
     int mid;

     if(low<high) {
          mid=(low+high)/2;
          mergesort(a,low,mid);
          mergesort(a,mid+1,high);
          merge(a,low,high,mid);
     }
}
```

```
void merge(int a[], int low, int high, int mid) {
	int i=low, j=mid+1, k=low, c[50];
	while((i<=mid)&&(j<=high)) {
		if(a[i]<a[j]) {
			c[k]=a[i];
			k++;  i++;
		} else {
			c[k]=a[j];
			k++;  j++;
		}
	}
	while(i<=mid) {
		c[k]=a[i];
		k++; i++;
	}
	while(j<=high) {
		c[k]=a[j];
		k++; j++;
	}
	for(i=low;i<k;i++)
		a[i]=c[i];
}
```

Le torri di Hanoi

Spostare tutta la torre da A a C **spostando un cerchio alla volta** e **senza mai mettere un cerchio più grosso su uno più piccolo**

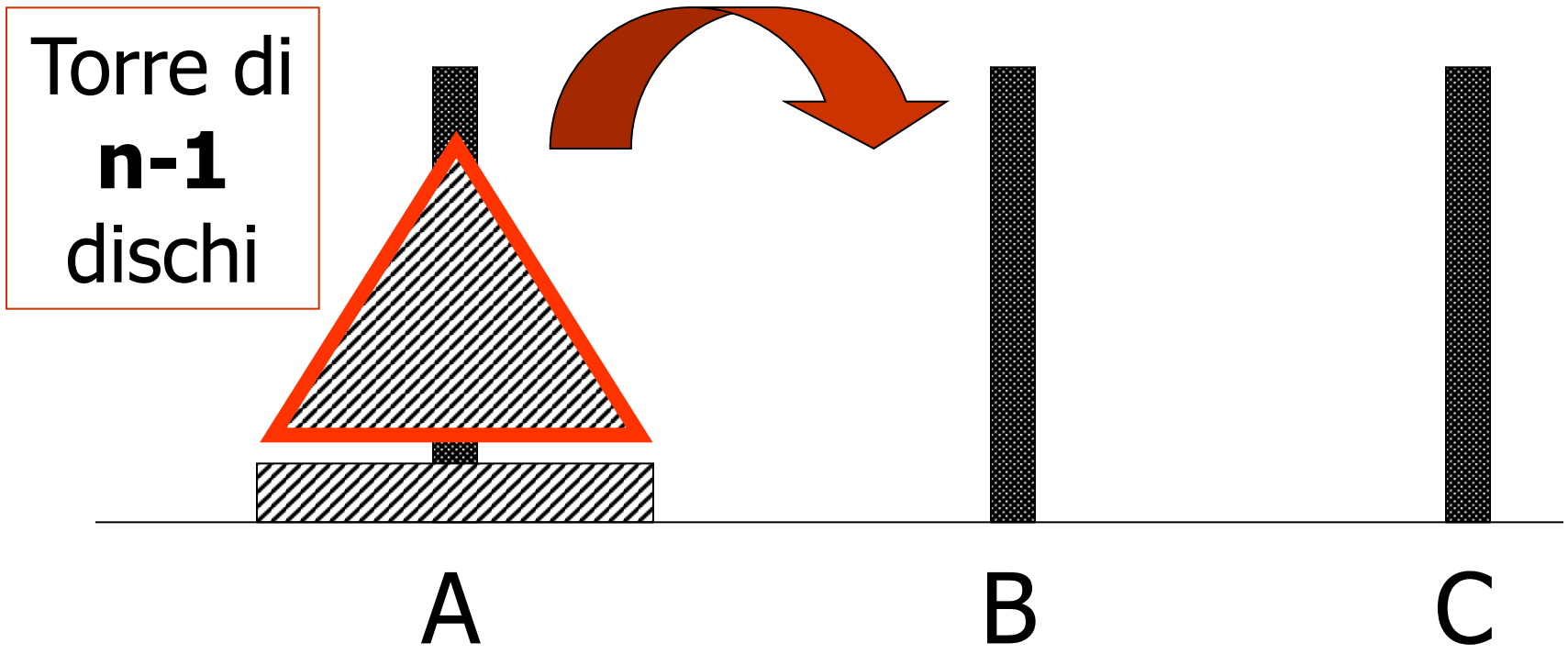
Torre di **n** dischi



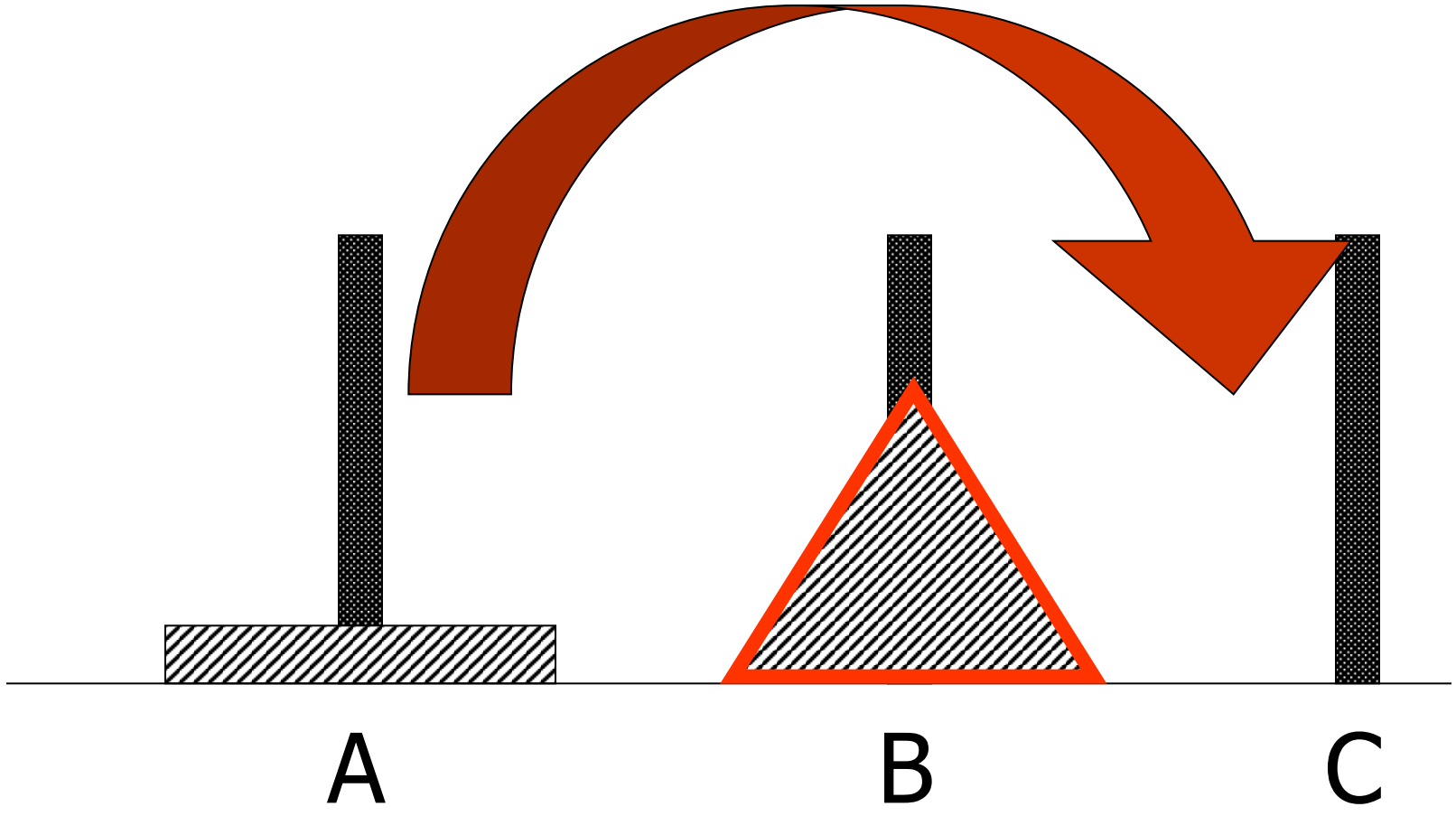
FORMULAZIONE RICORSIVA?

Le torri di Hanoi

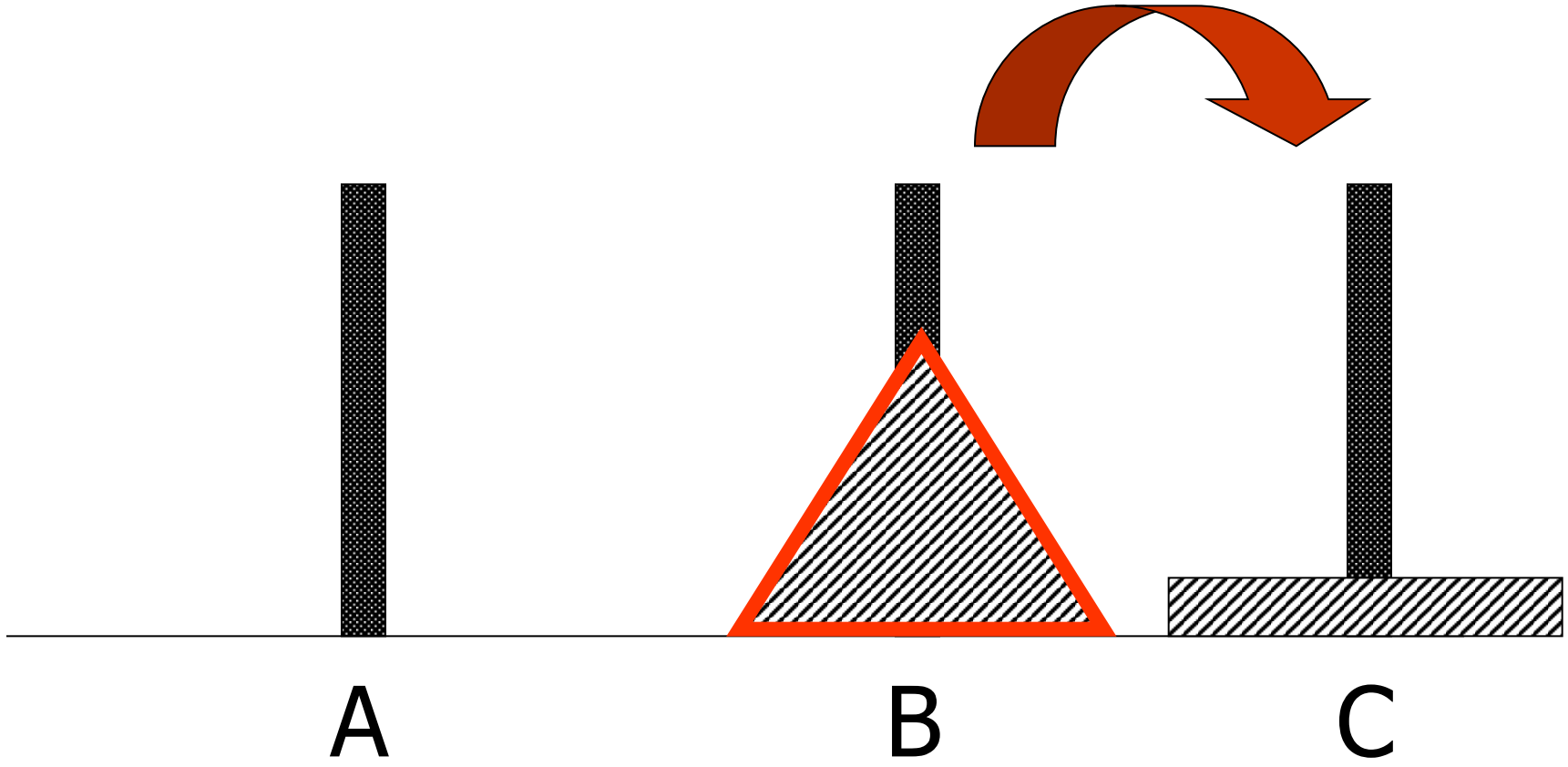
FORMULAZIONE RICORSIVA



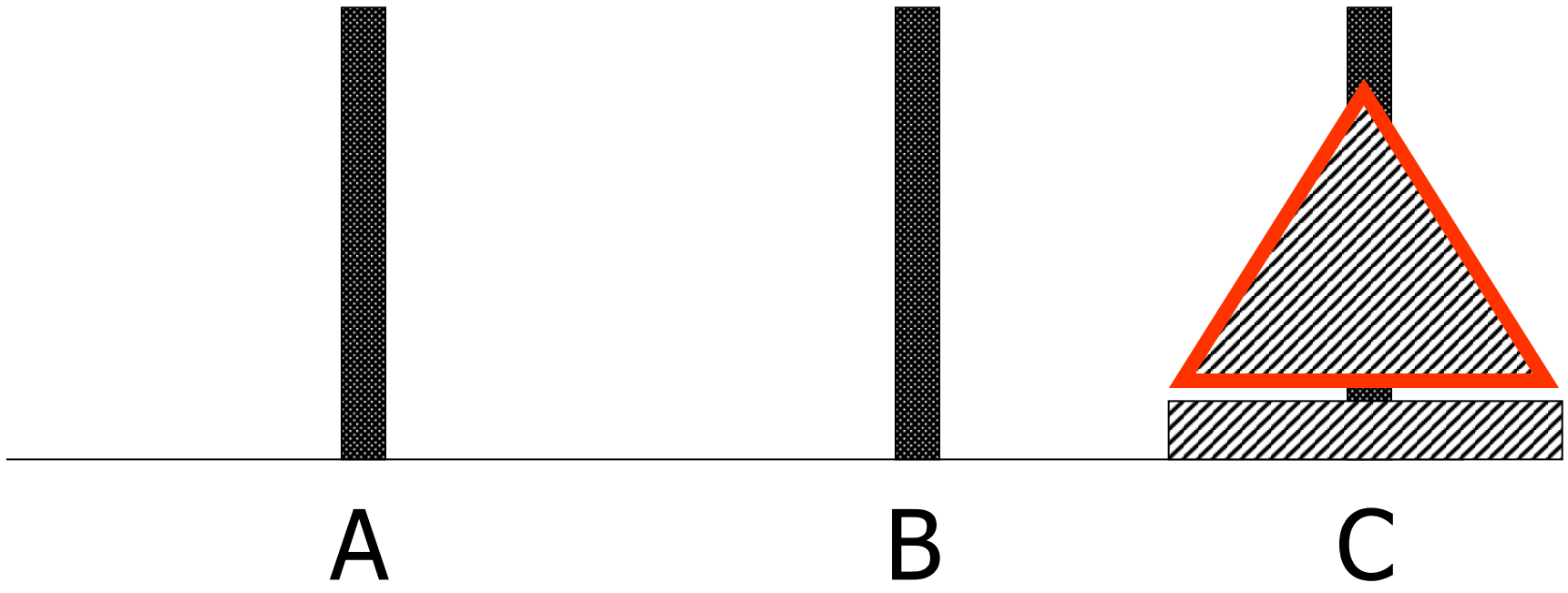
Le torri di Hanoi



Le torri di Hanoi



Le torri di Hanoi



Il metodo

- Il programma deve stampare una serie di comandi di spostamento
  - So spostare la torre di 1 elem. da A a C

    (caso di base)
  - Per spostare la torre di N elem. da A a C
    - sposto la torre di N-1 cerchi da A a B
    - sposto il cerchio restante in C
    - sposto la torre di N-1 elementi da B a C

Chiamata iniziale:

hanoi (12, 'A', 'C', 'B')
/*significa che abbiamo una torre di 12 cerchi
da trasferire da A a C potendo usare B * /

```
void hanoi (int n, char a, char c, char b) {
   if ( n != 0 ) {
      hanoi (n-1, a, b, c);
      printf ("sposta cerchio da %c a %c\n", a, c);
      hanoi (n-1, b, c, a);
   }
}
```

Una variante più "stringata"

hanoi (int n, **int** from, **int** to);
/* Significa che abbiamo una torre di n cerchi da trasferire da **from** a **to**; adottiamo una codifica dei nomi dei tre pioli che permetta di ricavare in modo immediato il "nome" di ogni piolo partendo dai "nomi" degli altri due: i pioli sono ora indicati da *interi* 1, 2, 3, e non dei caratteri 'A', 'B', 'C' */

```
void hanoi (int n, int from, int to) {
   if (n != 0) {
      hanoi (n-1, from, 6 – from – to);
      printf ("sposta cerchio da %d a %d", from, to);
                  hanoi (n-1, 6 – from – to, to);
         }
}
```

Hanoi: soluzione iterativa

- Non è così evidente…
- Stabiliamo un "senso orario" tra i pioli: 1, 2, 3 e poi ancora 1, ecc.
- Per muovere la torre nel prossimo piolo in senso orario bisogna ripetere questi due passi:
  - sposta il disco più piccolo in senso orario
  - fai l'unico altro spostamento possibile con un altro disco