Ricorsione

- quando una funzione chiama sè stessa, sia direttamente che per il tramite di altre funzioni, essa viene detta *ricorsiva*

- es: il fattoriale è una funzione intrinsecamente ricorsiva: $n! = n*(n - 1)!$ di ogni funzione ricorsiva si può avere un'implementazione iterativa:

```
// implementazione ricorsiva
int fattoriale(int n)
  { if (n == 0) return 1;   \\ condizione di terminazione
    else return n * fattoriale(n - 1); \\ passo
  }

// implementazione iterativa
fattoriale = 1;
for (int contatore = n; contatore >= 1; contatore --)
   fattoriale *= contatore ;
```


Ricorsione e iterazione

- Qualunque problema risolvibile ricorsivamente può essere risolto con un algoritmo iterativo;
  - per ogni funzione ricorsiva se ne può trovare un'altra che fa la stessa cosa attraverso un ciclo (senza richiamare se stessa)

- la ricorsione spesso produce soluzioni concettualmente più semplici
  - la corrispondente soluzione iterativa sarà normalmente più efficiente, sia in termini di occupazione di spazio di memoria che in termini di tempo di computazione.

Svantaggi della Ricorsione

- Spreco di tempo
  - Ogni chiamata della funzione richiede per se un tempo di esecuzione (indipendente da cosa farà la funzione)
- “Spreco” di memoria
  - Ad ogni chiamata bisogna memorizzare nello stack una serie di registri e paramteri
    - Es. indirizzo dell’istruzione da seguire quando la funzione terminerà la sua esecuzione.
    - Argomenti della funzione
    - Variabili locali

Esempio: Prodotto di due numeri naturali

Prodotto di due numeri naturali a e b

- Soluzione iterativa

$$a*b = \underbrace{a+a+a+\ldots+a}_{\text{b volte}}$$

- Soluzione ricorsiva

$a*b=a$	se $b=1$
$a*b=a*(b-1) + a$	altrimenti

Esempio: Serie di Fibonacci

```
0,1,1,2,3,5,8,13,21,44,…
```

Soluzione ricorsiva

```
fibonacci(0)=0
fibonacci(1)=1
fibonacci(n)=fibonacci(n-1)+fibonacci(n-2)
```

in modo ancora più compatto

```
fibonacci(n)=n                                   if n=0,1
fibonacci(n)=fibonacci(n-1)+fibonacci(n-2)   per n>=2
```


Serie di Fibonacci: soluzione iterativa

0,1,1,2,3,5,8,13,21,44,…

```
int fibonacci(int n)
{  int primo=0, secondo=1, temp;

   if (n<=1) return n;
   for (i=2; i<=n; i++)
   {  temp=primo;
      primo=secondo;
      secondo=primo + temp;
   }
   return secondo;
```


Ricorsione indiretta

- La funzione chiama se stessa non direttamente ma tramite una concatenazione di chiamate con altre funzioni.

Esempio

```
void A(int c)
{  if (c> 5) B(c);
   cout << c << “ “;}

void B(int c)
{  A(--c);  }

int main()
{
   A(25);
}
```


Esercizio: Calcolare la somma dei primi N numeri interi

Versione ricorsiva

$$\texttt{Sommatoria(N)} = \begin{cases} \texttt{1} & \texttt{se N=1} \\ \texttt{N+Sommatoria(N-1)} & \texttt{altrimenti} \end{cases}$$

Esercizio: Calcolare la somma dei quadrati dei primi N numeri interi

Versione ricorsiva

$$\texttt{Sommatoria(N)} = \begin{cases} \texttt{1} & \texttt{se N=1} \\ \texttt{N}^2\texttt{+Sommatoria(N-1)} & \texttt{altrimenti} \end{cases}$$

Esercizio: Massimo Comune Divisore (MCD)

MCD(m,n)

$$\text{MCD(m,n)} = \begin{cases} \text{n} & \textbf{se n<=m e n è divisore di m} \\ \text{MCD(n,m)} & \textbf{se m < n} \\ \text{MCD(n, resto di m diviso n)} & \textbf{altrimenti} \end{cases}$$

Esercizio: Massimo Comune Divisore (MCD)

MCD(m,n)

$m=r_0$

$n=r_1$

$r_0=r_1q_1+r_2 \qquad 0<r_2<r_1$

$r_1=r_2q_2+r_3 \qquad 0<r_3<r_2$

…

$r_{l-1}=r_lq_l \qquad 0<r_l<r_{l-1}$

Determiniamo l

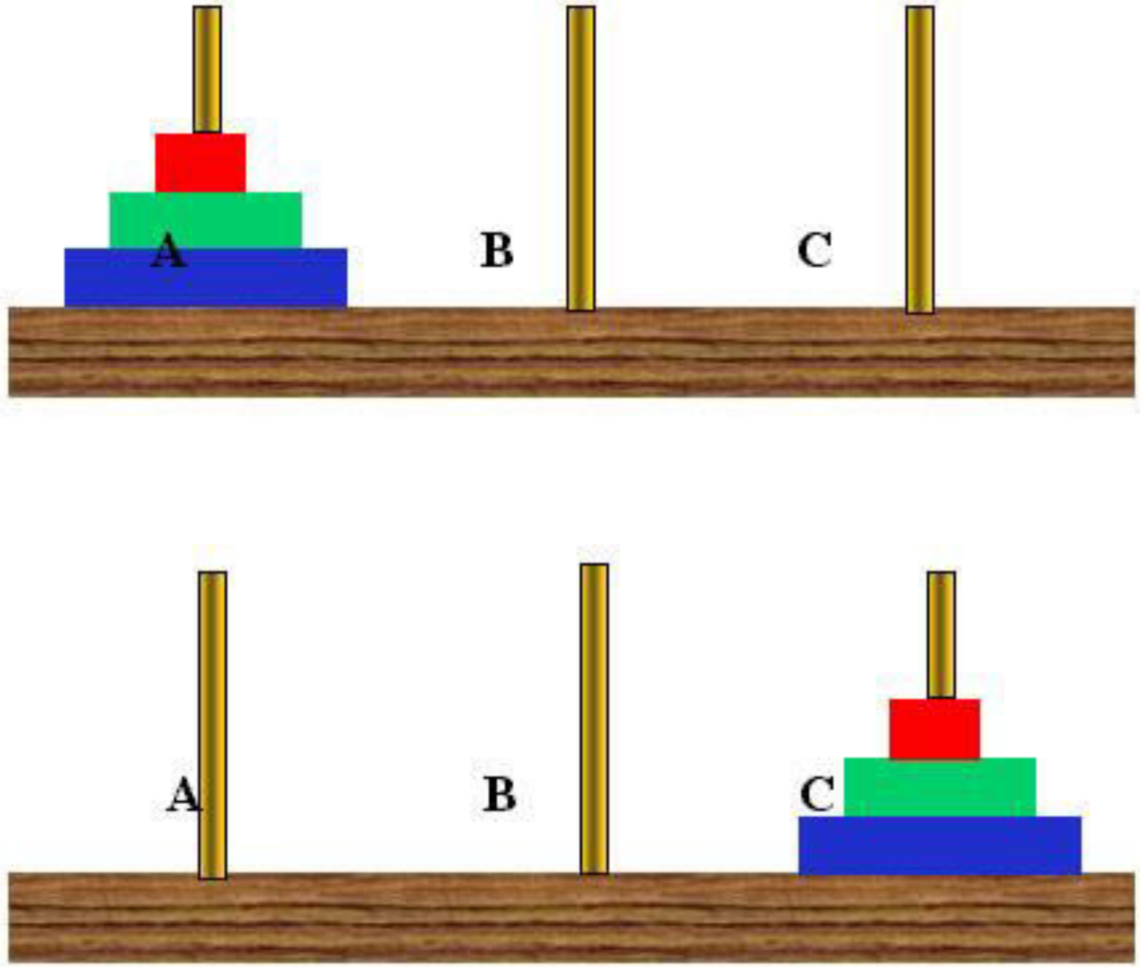
$$\phi = \frac{1+\sqrt{5}}{2}$$

$$\ell \leq (\log n / \log \phi) + 1$$

Invariante:

$$r_{\ell-i} \geq \phi^i$$

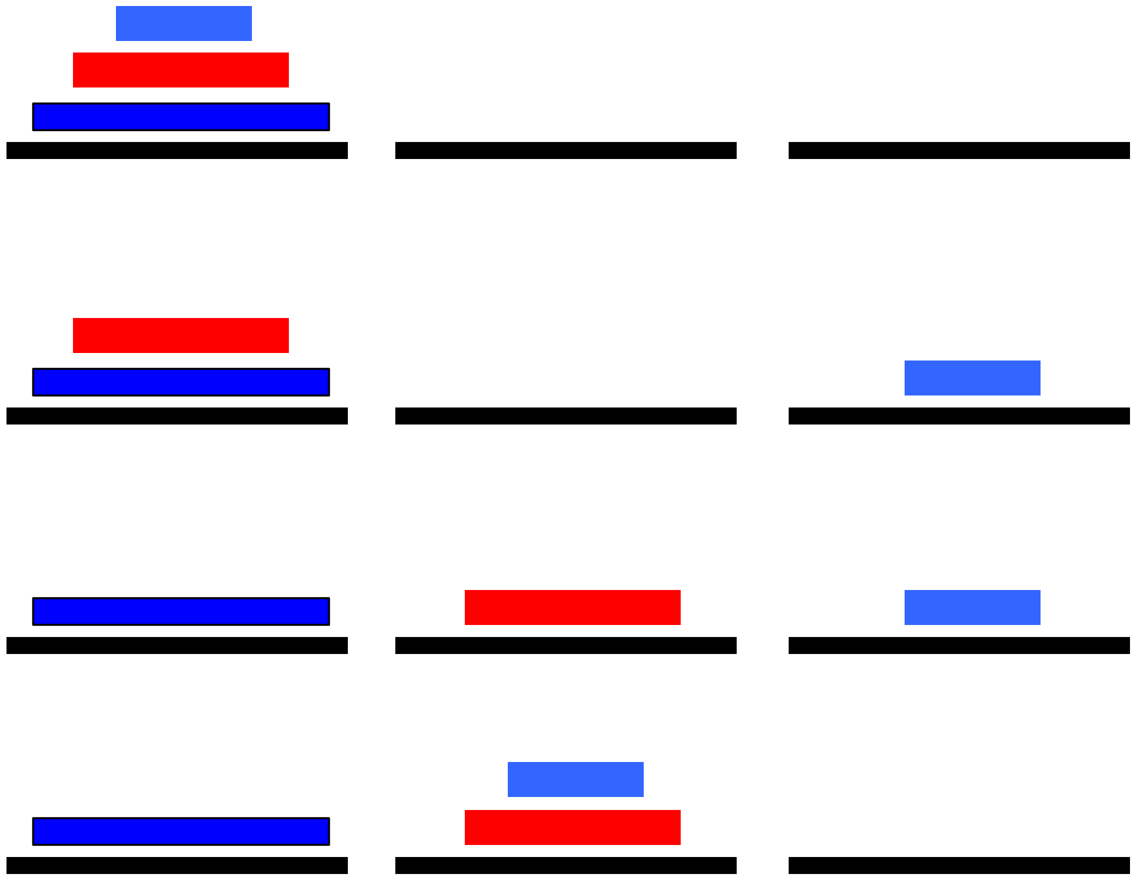
Torri di Hanoi

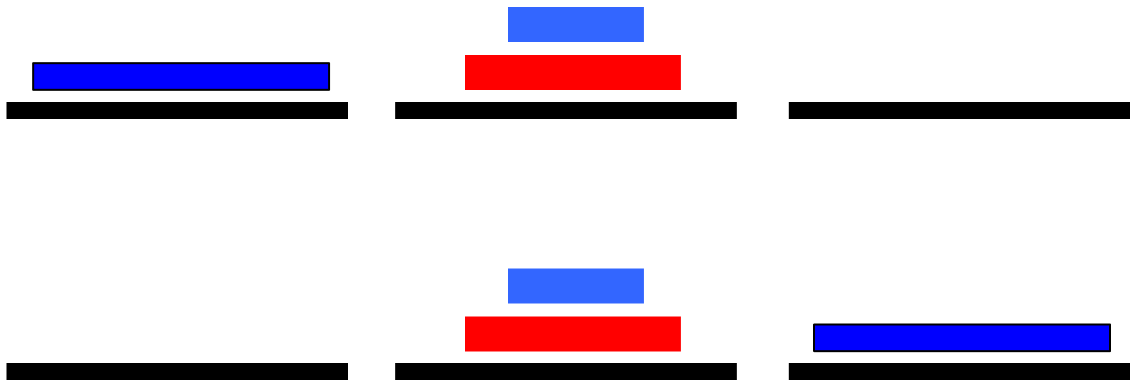


Configurazione iniziale e finale della *Torre di Hanoi* a tre dischi.

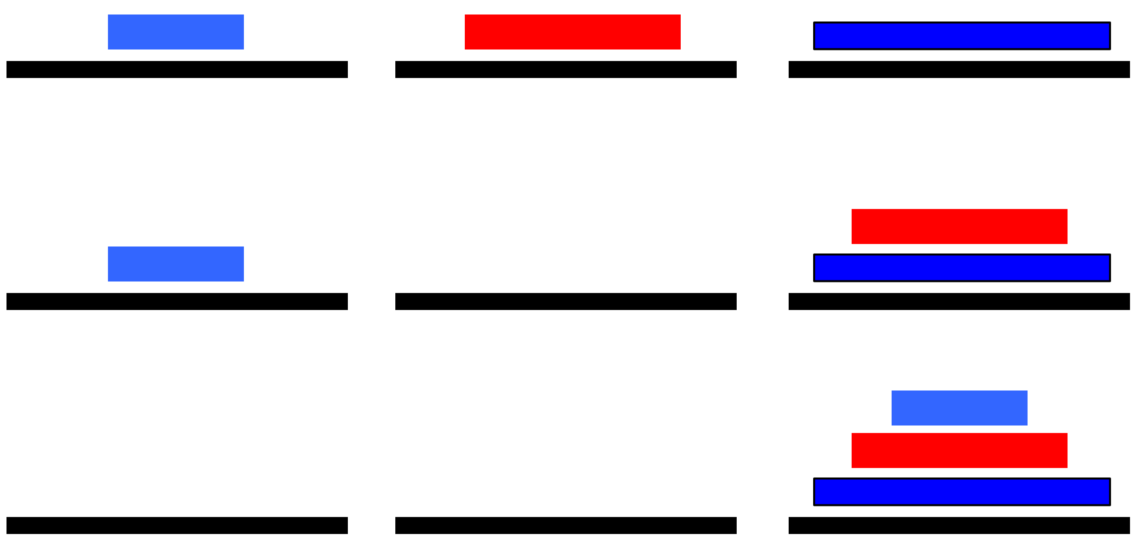

Luis Joyanes Aguilar
Fondamenti
di programmazione in C++
Algoritmi, strutture dati e oggetti
McGraw-Hill

Esempio: n=3





Situazione molto simile a quella iniziale!



Torri di Hanoi idee di base

- Se n=1 spostare (l’unico) disco sul piatto finale
- Se n>1
  - Spostare n-1 dischi dal piatto iniziale al piatto centrale
  - Spostare il disco più grande dal piatto iniziale al piatto finale
  - Spostare gli n-1 dischi rimanenti dal piatto centrale al piatto finale
    - Usando il piatto iniziale come piatto ausiliario

Complessità

- Quanti passi deve fare l'algoritmo per fermarsi?
- Nel caso delle torri di hanoi

$$\text{Num\_Spostamenti}(\texttt{hanoi(n)})=2(\text{Num\_Spostamenti}(\texttt{hanoi(n-1)}))+1$$

- Dimostriamo che tale numero è 2^n-1

