Algoritmi e Strutture Dati

Luciano Gualà
guala@mat.uniroma2.it
www.mat.uniroma2.it/~guala

Esercizio

Analizzare la complessità nel caso medio del primo algoritmo di pesatura (Alg1) presentato nella prima lezione. Rispetto alla distribuzione di probabilità sulle istanze, si assuma che la moneta falsa possa trovarsi in modo equiprobabile in una qualsiasi delle n posizioni.

Alg1 ($X=\{x_1, x_2, \ldots, x_n\}$)

1. **for** i=2 **to** n **do**
2. **if** peso(x_1) > peso(x_i) **then return** x_1
3. **if** peso(x_1) < peso(x_i) **then return** x_i

$$\sum_{I \text{ di dim } n} \Pr(I)\ \#\text{pesate}(I) =$$

$$\sum_{j=1}^{n} \underbrace{\Pr(\text{“moneta falsa è in posizione } j \text{ in } I\text{”})}_{1/n}\ \underbrace{\#\text{pesate}(I)}_{\substack{1 \text{ se } j=1,\\ j-1 \text{ altrimenti}}} = (1/n)\Big(1+\sum_{j=2}^{n}(j-1)\Big)$$

$$=(1/n)\Big(1+\sum_{j=1}^{n-1} j\Big) = (1/n)\big(1+ (n-1)\,n/2\big) = 1/n + (n-1)/2$$

Un problema simile: ricerca di un elemento in un array/lista non ordinata

l'algoritmo torna la posizione di x in L se x è presente, -1 altrimenti

```
algoritmo RicercaSequenziale(array L, elem x) → intero
1.   n = lunghezza di L
2.   i=1
3.   for i=1 to n do
4.        if  (L[i]=x) then return i \\trovato
5.   return  -1 \\non trovato
```

T(n): #elementi acceduti (linea 4) su un array di dimensione n

$T_{worst}(n) = n$ — $x \notin L$ oppure è in ultima posizione

$T_{avg}(n) = (n+1)/2$ — assumendo che $x \in L$ e che si trovi con la stessa probabilità in una qualsiasi posizione

Un problema simile: ricerca di un elemento in un array/lista non ordinata

l'algoritmo torna la posizione di x in L se x è presente, -1 altrimenti

algoritmo RicercaSequenziale(array L, elem x) $\rightarrow$ intero
1. n = lunghezza di L
2. i=1
3. **for** i=1 to n **do**
4.   **if** (L[i]=x) **then return** i *trovato*
5. **return** -1 *non trovato*

T(n): #operazioni RAM su un array di dimensione n

$T_{worst}(n) = \Theta(n)$ $x \notin L$ oppure è in ultima posizione

$T_{avg}(n) = \Theta(n)$ assumendo che $x \in L$ e che si trovi con la stessa probabilità in una qualsiasi posizione

Una variante: ricerca di un elemento in un array ordinato

Algoritmo di ricerca binaria: uno strumento molto potente

gli indici i e j indicano la porzione di L in cui cercare l'elemento x

l'algoritmo torna la posizione di x in L, se x c'è, -1 altrimenti

```
algoritmo RicercaBinariaRic(array L, elem x, int i, int j) --> intero
if (i>j) then return -1
m=⌊ (i+j)/2⌋
if (L[m]=x) then return m
if (L[m]>x) then return RicercaBinariaRic(L, x, i, m-1)
            else return RicercaBinariaRic(L, x, m+1,j)
```

$$T(n)=T(n/2)+O(1) \Longrightarrow T(n)=O(\log n)$$

Esempi su un array di 9 elementi

Cerca 2

Cerca 1

Cerca 9

Cerca 3

3<4 quindi i e j si invertono

ricorsione,tecniche di progettazione e equazioni di ricorrenza

Sommario

- Algoritmi ricorsivi: come analizzarli?
- Complessità di algoritmi ricorsivi e equazioni di ricorrenza
- Una tecnica di progettazione algoritmica: divide et impera
- Metodi per risovere equazioni di ricorrenza:
  - iterazione
  - albero della ricorsione
  - sostituzione
  - teorema Master
  - cambiamento di variabile

Algoritmi ricorsivi: come analizzarli?

algoritmo `fibonacci2`*(intero n)* $\rightarrow$ *intero*

if (n$\leq$2) **then return** 1

else return `fibonacci2`(n-1) + `fibonacci2`(n-2)

$$T(n)=T(n-1)+T(n-2)+O(1)$$

Algoritmi ricorsivi: come analizzarli?

Algoritmo di ricerca binaria: uno strumento molto potente

gli indici i e j indicano la porzione di L in cui cercare l'elemento x

l'algoritmo torna la posizione di x in L, se x c'è, -1 altrimenti

algoritmo RicercaBinariaRic(array *L*, elem *x*, int *i*, int *j*) -> intero

1. **if** (i>j) **then return** -1

2. m=$\lfloor (i+j)/2 \rfloor$

3. **if** (L[m]=*x*) **then return** m

4. **if** (L[m]>x) **then return** RicercaBinariaRic(L, *x*, i, m-1)

5. **else return** RicercaBinariaRic(L, *x*, m+1,j)

$$T(n)=T(n/2)+O(1)$$

Algoritmi ricorsivi: come analizzarli?

Alg4 (X)

1. **if** ($|X|=1$) **then** return unica moneta in X
2. dividi X in tre gruppi X_1, X_2, X_3 di dimensione bilanciata
   siano X_1 e X_2 i gruppi che hanno la stessa dimensione (ci sono sempre)
3. **if** peso(X_1) = peso(X_2) **then return** Alg4(X_3)
4. **if** peso(X_1) > peso(X_2) **then return** Alg4(X_1)
   else return Alg4(X_2)

$$T(n)=T(n/3)+O(1)$$

Equazioni di ricorrenza

la complessità computazionale di un algoritmo ricorsivo può essere espressa in modo naturale attraverso una equazione di ricorrenza

esempi:

$$T(n) = T(n/3) + 2T(n/4) + O(n \log n)$$

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

$$T(n) = T(n/3) + T(2n/3) + n$$

casi base:
T(costante)=cost
(a volte T(1)=1)

Metodo dell'iterazione

Idea: "srotolare" la ricorsione, ottenendo una sommatoria dipendente solo dalla dimensione n del problema iniziale

Esempio: $T(n) = c + T(n/2)$

$T(n/2) = c + T(n/4)$ ...

$$\begin{aligned} T(n) &= c + T(n/2) \\ &= 2c + T(n/4) \\ &= 2c + c + T(n/8) \\ &= 3c + T(n/8) \\ &\ \ \dots \\ &= i\,c + T(n/2^i) \end{aligned}$$

Per $i=\log_2 n$: $T(n) = c \log_2 n + T(1) = \Theta(\log n)$

Metodo dell'iterazione

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

$$
\begin{aligned}
T(n) &= T(n-1) + 1 \\
&= T(n-2) + 1 + 1 \\
&= T(n-2) + 2 \\
&= T(n-3) + 1 + 2 \\
&= T(n-3) + 3 \\
&= T(n-4) + 4 \\
&\dots \\
&= T(n-i) + i
\end{aligned}
$$

Per $i=n-1$: $T(n) = T(1) + n-1 = \Theta(n)$

Metodo dell'iterazione

Esempio: $T(n) = 2T(n-1)+1$

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&= 2(2T(n-2)+1) + 1 \\
&= 4T(n-2)+2+1 \\
&= 4(2T(n-3)+1)+2+1 \\
&= 8T(n-3)+4+2+1 \\
&= 16T(n-4)+8+4+2+1 \\
&\dots \\
&= 2^i T(n-i) + \sum_{j=0}^{i-1} 2^j
\end{aligned}
$$

per $i=n-1$

$$T(n) = 2^{n-1}T(1) + \sum_{j=0}^{n-2} 2^j = \Theta(2^n)$$

Metodo dell'iterazione

Esempio: $T(n) = T(n-1) + T(n-2) + 1$

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&= T(n-2)+T(n-3)+1+T(n-3)+T(n-4)+1+1 \\
&= T(n-2)+2T(n-3)+T(n-4)+3 \\
&= T(n-3)+T(n-4)+1+2(T(n-4)+T(n-5)+1)+T(n-5)+T(n-6)+1+3 \\
&= T(n-3)+3T(n-4)+3T(n-5)+T(n-6)+7 \\
&\dots
\end{aligned}
$$

Esercizi

risolvere usando il metodo dell'iterazione:

Esercizio 1: $T(n) = T(n-1) + n,$
$T(1) = 1$

Esercizio 2: $T(n) = 9\ T(n/3) + n,$
$T(1) = 1$

(soluzione sul libro di testo: Esempio 2.4)

Analisi dell'albero della ricorsione

(un modo grafico di pensare il metodo dell'iterazione)

Idea:

- disegnare l'albero delle chiamate ricorsive indicando la dimensione di ogni nodo
- stimare il tempo speso da ogni nodo dell'albero
- stimare il tempo complessivo "sommando" il tempo speso da ogni nodo

Suggerimento 1: se il tempo speso da ogni nodo è costante, T(n) è proporzionale al numero di nodi

Suggerimento 2: a volte conviene analizzare l'albero per livelli:
-analizzare il tempo speso su ogni livello (fornendo upper bound)
-stimare il numero di livelli

tecnica albero della ricorsione

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

$T(1) = 1$

n

$n - 1$

$n-2$

$\vdots$

$n-i$

$\vdots$

2

1

quanto costa ogni nodo? ...uno!

quanti nodi ha l'albero? n

$T(n) = \Theta(n)$

tecnica albero della ricorsione

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

$T(1) = 1$

n

$|$

$n - 1$

$|$

$n-2$

$\vdots$

$n-i$

$\vdots$

2

$|$

1

quanto costa ogni nodo? al più n

quanti nodi ha l'albero? n

$$T(n)=O(n^2)$$

vale $T(n)=\Theta(n^2)$?

tecnica albero della ricorsione

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

$T(1)= 1$

n

$n-1$

$n-2$

$\vdots$

$n-i$

$\vdots$

2

1

$n/2$ nodi ognuno dei quali costa $\geq n/2$

quanto costa ogni nodo? al più n

quanti nodi ha l'albero? n

$\Rightarrow$ $T(n)=O(n^2)$

vale $T(n)=\Theta(n^2)$?

$T(n)\geq n/2 \, n/2=n^2/4$

$\Rightarrow$ $T(n)=\Omega(n^2)$

tecnica albero della ricorsione

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

$T(1) = 1$

n

$n-1$

$n-2$

$\vdots$

$n-i$

$\vdots$

2

1

$n/2$ nodi ognuno dei quali costa $\geq n/2$

$T(n) \geq n/2 \, n/2 = n^2/4$

$\Rightarrow T(n) = \Omega(n^2)$

quanto costa ogni nodo? al più n

quanti nodi ha l'albero? n

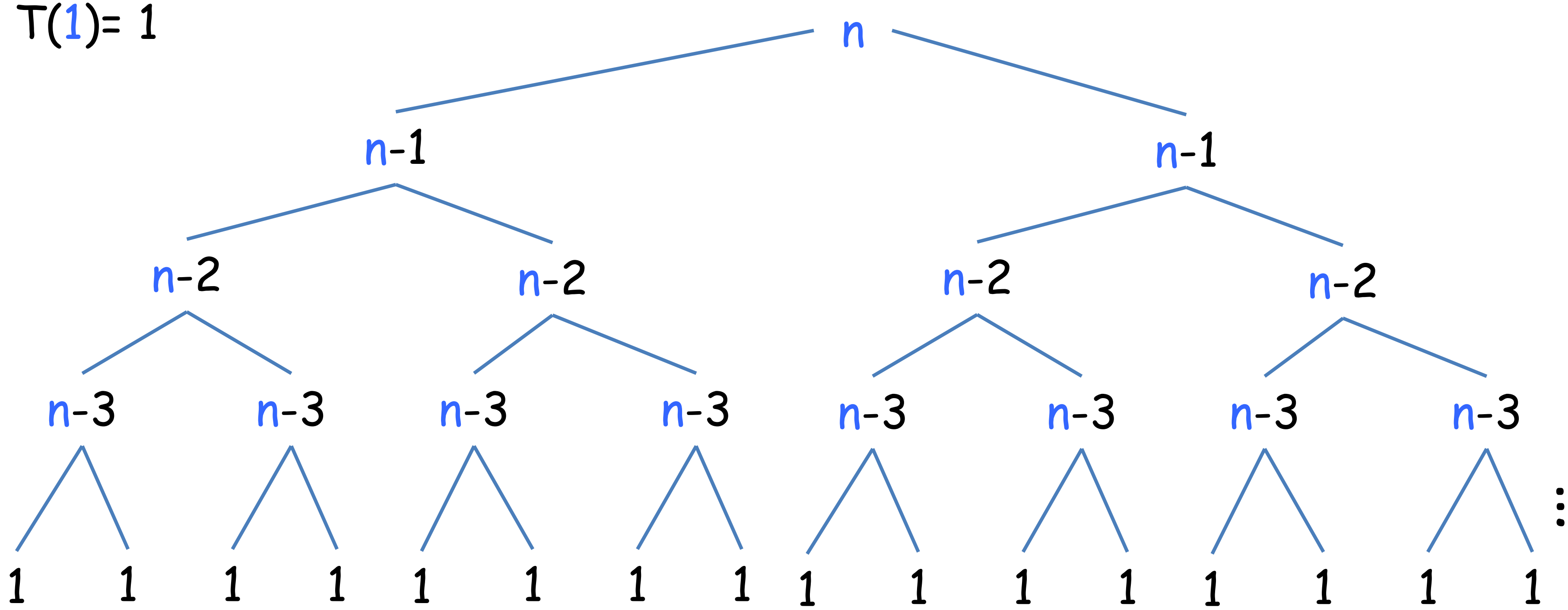
$\Rightarrow T(n) = O(n^2)$

$$\boxed{T(n) = \Theta(n^2)}$$

tecnica albero della ricorsione

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

$T(1) = 1$



albero binario completo!

quanto costa ogni nodo? ...uno!

quanto è alto l'albero? ...$n-1$!

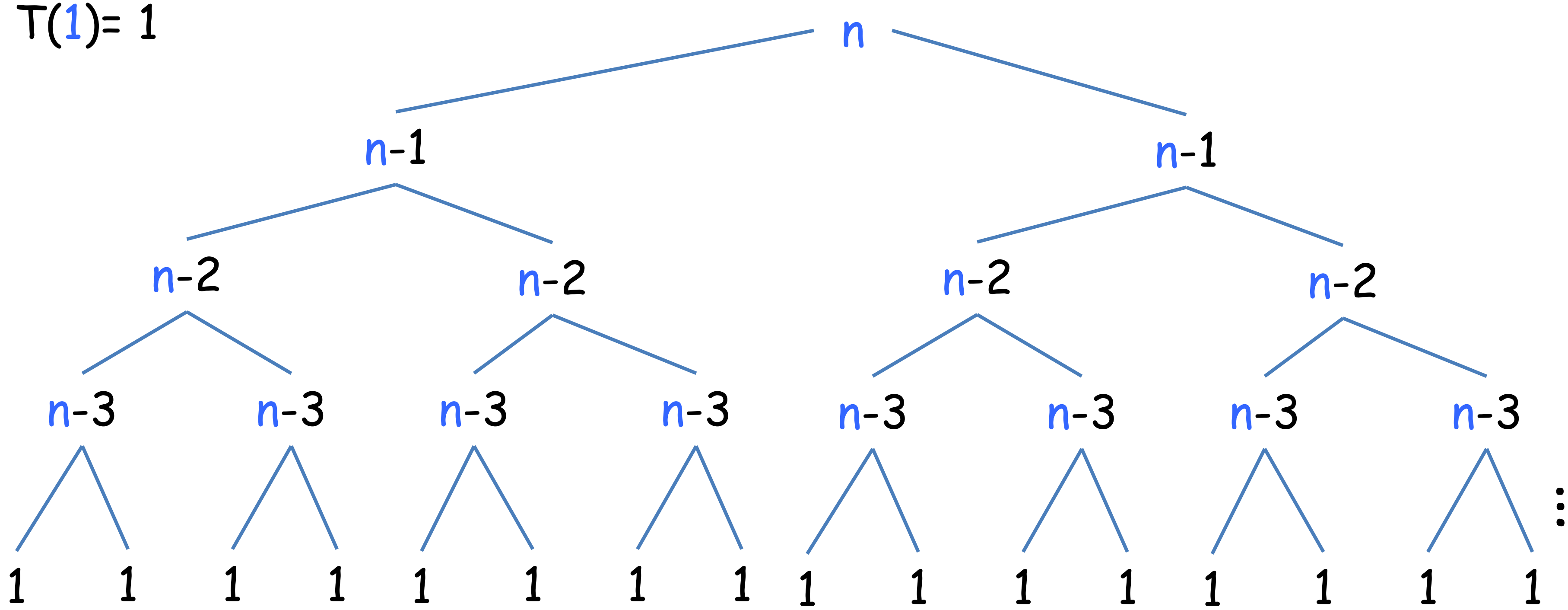
quanti nodi ha un albero binario completo di altezza h? $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$

$\Rightarrow$ $T(n) = 2^n - 1 = \Theta(2^n)$

tecnica albero della ricorsione

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

$T(1) = 1$



albero binario completo!

quanto costa ogni nodo? ...al più n

quanto è alto l'albero? ...$n-1$

quanti nodi ha un albero binario completo di altezza h? $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$

$\Rightarrow$ $T(n) \leq n2^n = \Theta(n2^n)$

$T(n) = O(n2^n)$

tecnica albero della ricorsione

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

$T(1) = 1$

Un'idea: usare maggiorazioni per fornire upper bound

$T(n) \leq R(n)$

$R(n) = 2\,R(n-1) + 1$

$\Rightarrow$

$R(n) = \Theta(2^n)$

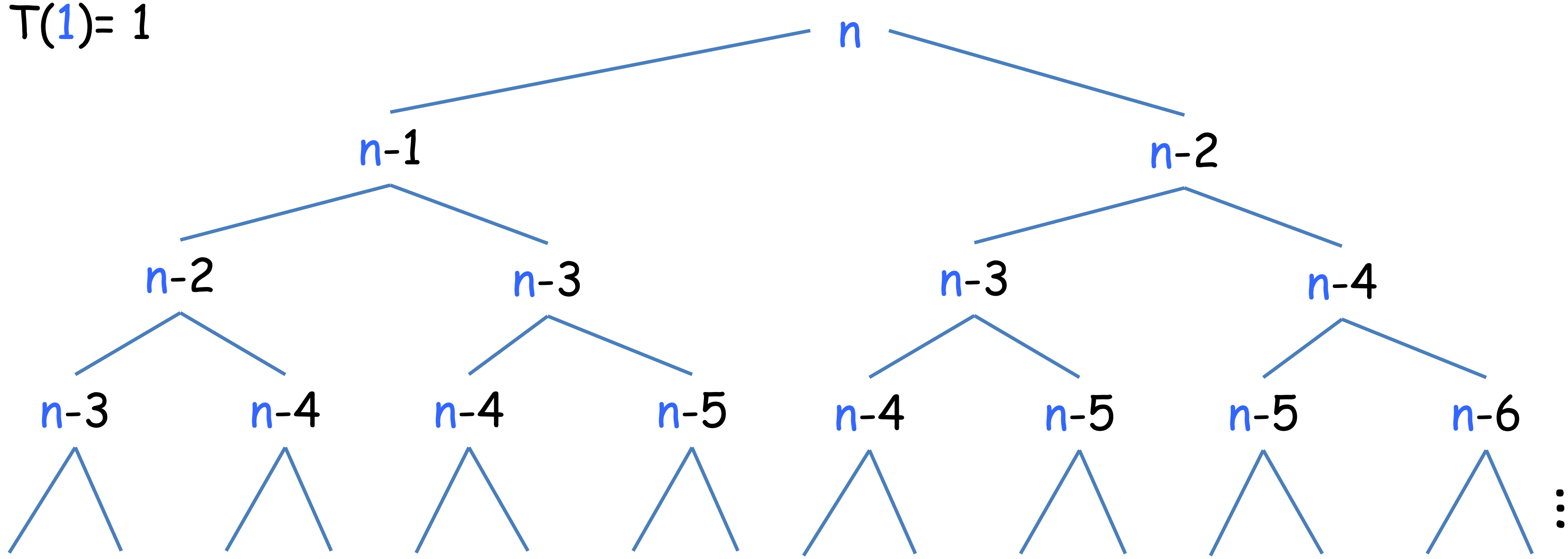
$\Rightarrow$ $T(n) = O(2^n)$

vale $T(n) = \Theta(2^n)$?

tecnica albero della ricorsione

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

$T(1) = 1$



albero chiamate ricorsive dell'algorito `Fibonacci2`!

quanto costa ogni nodo? …uno

quanti nodi ha l'albero? $\Theta(\phi^n)$

$$T(n) = \Theta(\phi^n)$$

$$[T(n) = o(2^n)]$$

Analisi dell'albero della ricorsione

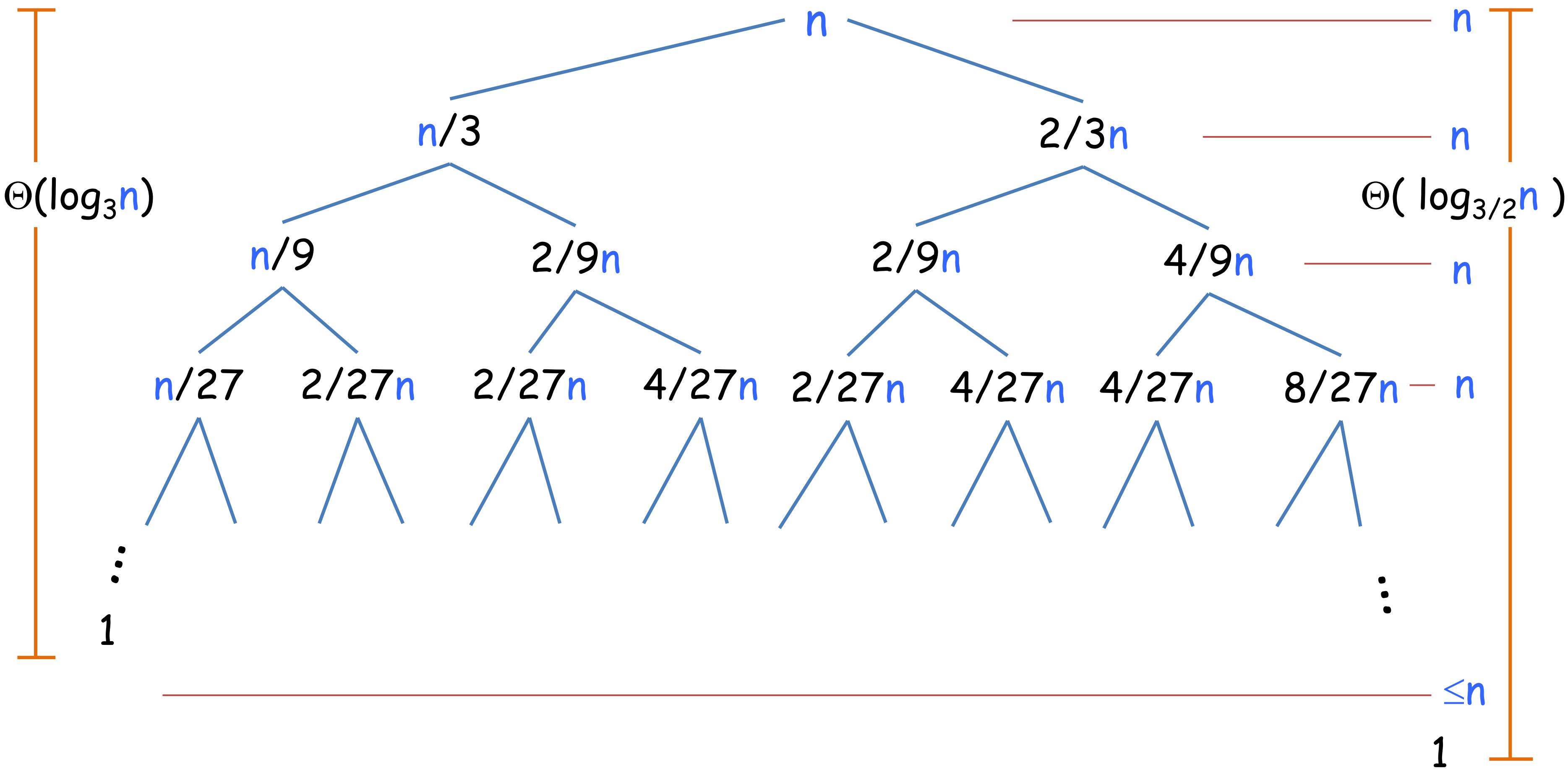
due esempi:

Esempio 1: $T(n) = T(n/3) + T(2n/3) + n,$
$T(1) = 1$

Esempio 2: $T(n) = 2\ T(n-2) + 1,$
$T(1) = 1$

$T(n) = T(n/3) + T(2/3\,n) + n$

$T(1) = 1$



quanto costa ogni livello? $\leq n$

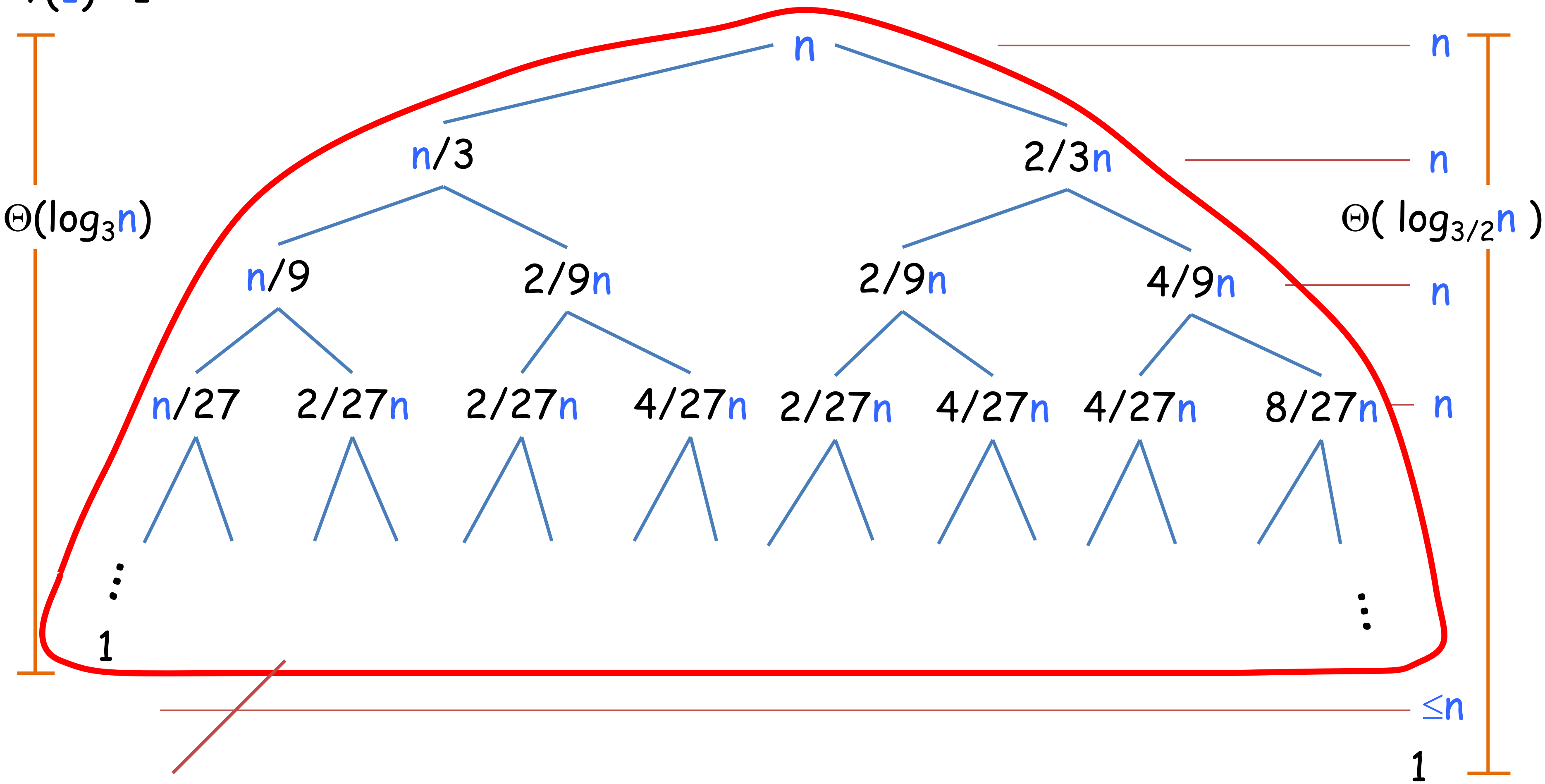
quanti livelli ha l'albero? $O(\log_{3/2} n)$

$\Rightarrow$ $T(n) = O(n \log n)$

vale $T(n) = \Theta(n \log n)$?

$T(n) = T(n/3) + T(2/3\ n) + n$

$T(1) = 1$



$T(n) \geq n \log_3 n$

$\Rightarrow T(n) = \Omega(n \log n)$

$\Rightarrow T(n) = \Theta(n \log n)$

Sommario

- Algoritmi ricorsivi: come analizzarli?
- Complessità di algoritmi ricorsivi e equazioni di ricorrenza
- Una tecnica di progettazione algoritmica: divide et impera
- Metodi per risovere equazioni di ricorrenza:
  - iterazione
  - albero della ricorsione
  - sostituzione
  - teorema Master
  - cambiamento di variabile

Metodo della sostituzione

Idea:

1. indovinare la (forma della) soluzione
2. usare induzione matematica per provare che la soluzione è quella intuita
3. risolvi rispetto alle costanti

Metodo della sostituzione

Esempio: $T(n) = n + T(n/2)$, $T(1)=1$

Proviamo a dimostrare che la soluzione sia $T(n) \leq c\,n$ per una costante c opportuna

Passo base: $T(1)=1 \leq c\,1$ per ogni $c \geq 1$

Passo induttivo:
assumiamo $T(k) \leq c\,k$ per ogni $k<n$

$T(n)= n + T(n/2) \leq n+c\,(n/2) = (c/2+1)\,n$

Quindi: quando $T(n) \leq c\,n$?

devo avere: $c/2+1 \leq c$
da cui segue: $c \geq 2$

$T(n) \leq 2n$ ⇨ $T(n)=O(n)$

Esercizi

risolvere usando il metodo della sostituzione:

Esercizio: $T(n) = 4T(n/2) + n$,
$T(1) = 1$

(…e fare esperienza della tecnicità del metodo.)

$T(n) = 4T(n/2) + n,$
$T(1) = 1$

ipotizzo: $T(n)=O(n^3)$

proviamo a dimostrare che: $T(n) \leq c\, n^3$

assumiamo: $T(k) \leq c\, k^3$ per ogni $k < n$

$$
\begin{aligned}
T(n) &= 4T(n/2)+n \\
&\leq 4c\,(n/2)^3 + n \\
&= \tfrac{1}{2} c\, n^3 + n \\
&= \underbrace{c\, n^3}_{\text{goal}} - \underbrace{\left(\tfrac{1}{2} c n^3 - n\right)}_{\text{residuo}} \\
&\leq c n^3
\end{aligned}
$$

se $\frac{1}{2} c n^3 - n \geq 0$

per esempio: $c \geq 2$

caso base:

$T(1)=1 \leq c$

per esempio: $c \geq 1$

$\Rightarrow$

$T(n) \leq 2n^3$

$T(n)=O(n^3)$

$T(n) = 4T(n/2) + n,$
$T(1) = 1$

ipotizzo: $T(n)=O(n^2)$

proviamo a dimostrare che: $T(n) \leq c\ n^2$

assumiamo: $T(k) \leq c\ k^2$ per ogni $k < n$

$$
\begin{aligned}
T(n) &= 4T(n/2)+n \\
&\leq 4c\ (n/2)^2 + n \\
&= c\ n^2 + n \\
&\not\leq cn^2
\end{aligned}
$$

$T(n) = 4T(n/2) + n,$
$T(1) = 1$

ipotizzo: $T(n)=O(n^2)$

proviamo a dimostrare che: $T(n) \leq c_1 n^2 - c_2 n$

assumiamo: $T(k) \leq c_1 k^2 - c_2 k$ per ogni $k < n$

$$
\begin{aligned}
T(n) &= 4T(n/2)+n \\
&\leq 4\,(c_1 (n/2)^2 - c_2 (n/2)) + n \\
&= c_1 n^2 - 2 c_2 n + n \\
&= \underbrace{c_1 n^2 - c_2 n}_{\text{goal}} - \underbrace{(c_2 n - n)}_{\text{residuo}} \\
&\leq c_1 n^2 - c_2 n
\end{aligned}
$$

se $c_2 n - n \geq 0$

per esempio: $c_2 \geq 1$

caso base:

$T(1)=1 \leq c_1 - c_2$

per esempio: $c_1=2$ $c_2=1$

$T(n) \leq 2n^2 - n$

$T(n)=O(n^2)$

Tecnica del divide et impera

Algoritmi basati sulla tecnica del *divide et impera*:
- dividi il problema (di dimensione n) in a sottoproblemi di dimensione n/b
- risolvi i sottoproblemi ricorsivamente
- ricombina le soluzioni

Sia $f(n)$ il tempo per dividere e ricombinare istanze di dimensione n. La relazione di ricorrenza è data da:

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & \text{se } n>1 \\ \Theta(1) & \text{se } n=1 \end{cases}$$

Algoritmo Fibonacci6

algoritmo `fibonacci6`$(intero\ n) \rightarrow intero$

$A \leftarrow \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

$M \leftarrow$ `potenzaDiMatrice`$(A, n-1)$

return $M[0][0]$

funzione `potenzaDiMatrice`$(matrice\ A,\ intero\ k) \rightarrow matrice$

if $(k \leq 1)$ **then** $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

else $M \leftarrow$ `potenzaDiMatrice`$(A, \lfloor k/2 \rfloor)$

$\quad M \leftarrow M \cdot M$

if (k è dispari) **then** $M \leftarrow M \cdot A$

return M

a=1, b=2, f(n)=O(1)

Algoritmo ottimo di pesatura

Alg4 (X)

1. **if** (|X|=1) **then** return unica moneta in X
2. dividi X in tre gruppi X_1, X_2, X_3 di dimensione bilanciata siano X_1 e X_2 i gruppi che hanno la stessa dimensione (ci sono sempre)
3. **if** peso(X_1) = peso(X_2) **then return** Alg4(X_3)
4. **if** peso(X_1) > peso(X_2) **then return** Alg4(X_1)
   else return Alg4(X_2)

a=1, b=3, $f(n)=O(1)$

Teorema Master: enunciato informale

$$n^{\log_b a} \quad \text{vs} \quad f(n)$$

quale va più velocemente a infinito?

Stesso ordine asintotico $\rightarrow$ $T(n) = \Theta(f(n) \log n)$

Se una delle due è "polinomialmente" più veloce
$\rightarrow$ $T(n)$ ha l'ordine asintotico della più veloce

Teorema Master

La relazione di ricorrenza:

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & \text{se } n>1 \\ \Theta(1) & \text{se } n=1 \end{cases}$$

ha soluzione:

1. $T(n) = \Theta(n^{\log_b a})$ se $f(n)=O(n^{\log_b a - \varepsilon})$ per $\varepsilon > 0$

2. $T(n) = \Theta(n^{\log_b a} \log n)$ se $f(n) = \Theta(n^{\log_b a})$

3. $T(n) = \Theta(f(n))$ se $f(n)=\Omega(n^{\log_b a + \varepsilon})$ per $\varepsilon > 0$ e $a\,f(n/b) \leq c\,f(n)$ per $c<1$ e n sufficientemente grande

Esempi

1) $T(n) = n + 2T(n/2)$

$a=2,\ b=2,\ f(n)=n=\Theta(n^{\log_2 2})$ ➡ $T(n)= \Theta(n \log n)$

(caso 2 del teorema master)

2) $T(n) = c + 3T(n/9)$

$a=3,\ b=9,\ f(n)=c=O(n^{\log_9 3 - \varepsilon})$ ➡ $T(n)= \Theta(\sqrt{n})$

(caso 1 del teorema master, es: $\varepsilon=0.1$)

3) $T(n) = n + 3T(n/9)$

$a=3,\ b=9,\ f(n)=n=\Omega(n^{\log_9 3 + \varepsilon})$

$3(n/9) \leq c\, n$ per $c=1/3$ ➡ $T(n)= \Theta(n)$

(caso 3 del teorema master, es: $\varepsilon=0.1$)

Esempi

4) $T(n) = n \log n + 2T(n/2)$

$a=2$, $b=2$, $f(n) = \omega(n^{\log_2 2})$

ma $f(n) \neq \Omega(n^{\log_2 2+\varepsilon})$, $\forall \varepsilon > 0$

non si può applicare
il teorema Master!

Cambiamento di variabile

Esempio: $T(n) = T(\sqrt{n}) + O(1)$,
$T(1) = 1$

$$T(n) = T(n^{1/2}) + O(1)$$

$n=2^x \quad \rightarrow \quad x = \log_2 n$

$T(2^x) = T(2^{x/2}) + O(1)$ $\qquad R(x):=T(2^x)$

$R(x) = R(x/2) + O(1) \quad \Rightarrow \quad R(x) = O(\log x)$

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

due problemi (per cui la ricorsione può aiutare)

Esercizio: progettare due algoritmi ricorsivi per i seguenti due problemi. Se ne studi la complessità temporale (nel caso peggiore).

problema della celebrità

ad una festa ci sono n persone
una di queste è una celebrità
la celebrità non conosce nessuno ma è conosciuta da tutti

obiettivo:
individuare la celebrità facendo (poche) domande a persone del tipo:
conosci questa persona?

problema della celebrità: un algoritmo ricorsivo

Celebrità (X)

X: insieme di persone fra le quali sto cercando la celebrità

1. **if** |X|=1 **then return** l'unica persona in X
   % che è la celebrità

2. siano A e B due persone qualsiasi in X:
   chiedi ad A se conose B

3. **if** (A conosce B)
   then
     %A non può essere la celebrità
     return Celebrità(X-{A})
   else
     %B non può essere la celebrità
     return Celebrità(X-{B})

quante domande fa l'algoritmo?

T(n): # domande che l'algoritmo fa nel caso peggiore prima di individuare la celebrità fra n persone

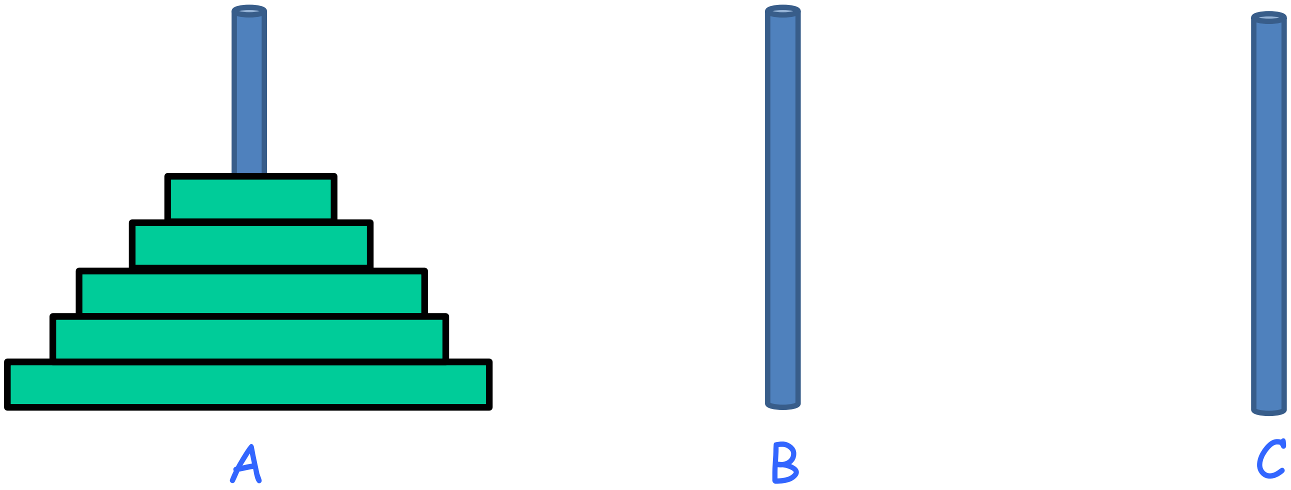
$T(n)=T(n-1)+1$ $\quad$ $T(1)=0$

$$\boxed{T(n)= n-1}$$

(srotolando)

$T(n)=T(n-1)+1=T(n-2)+2=T(n-3)+3= \dots T(n-i)+i \dots =T(1)+n-1=n-1$

La torre di Hanoi

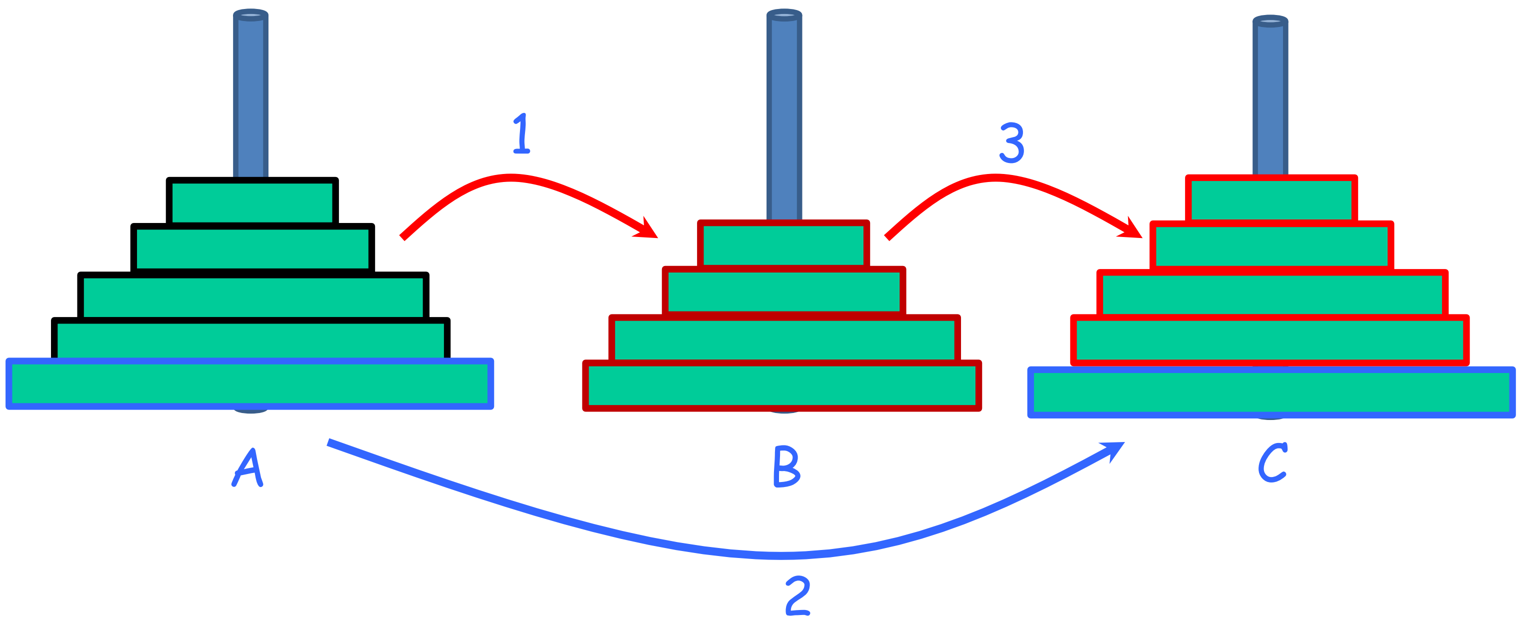


n dischi di diametro diverso, tre pali

regole: si può spostare un disco alla volta e non si può mettere un disco di diametro più grande sopra uno di diametro più piccolo

obiettivo: spostare i dischi dal palo A al palo C (facendo meno spostamenti possibile)

Un'elegante soluzione ricorsiva



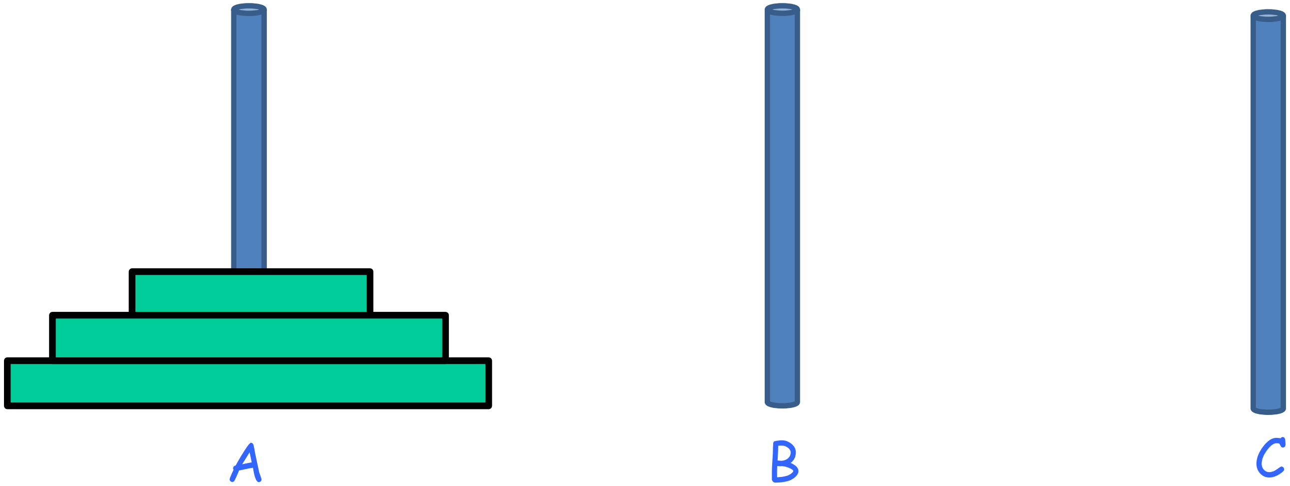
Hanoi(dischi, destinazione, palo ausiliario)

Hanoi ([1,2..,n], C, B)

1. **if** n=1 **then** sposta il disco su C
2. Hanoi([1,2,…,n-1], B, C)
3. sposta il disco n su C
4. Hanoi([1,2,…,n-1], C, A)

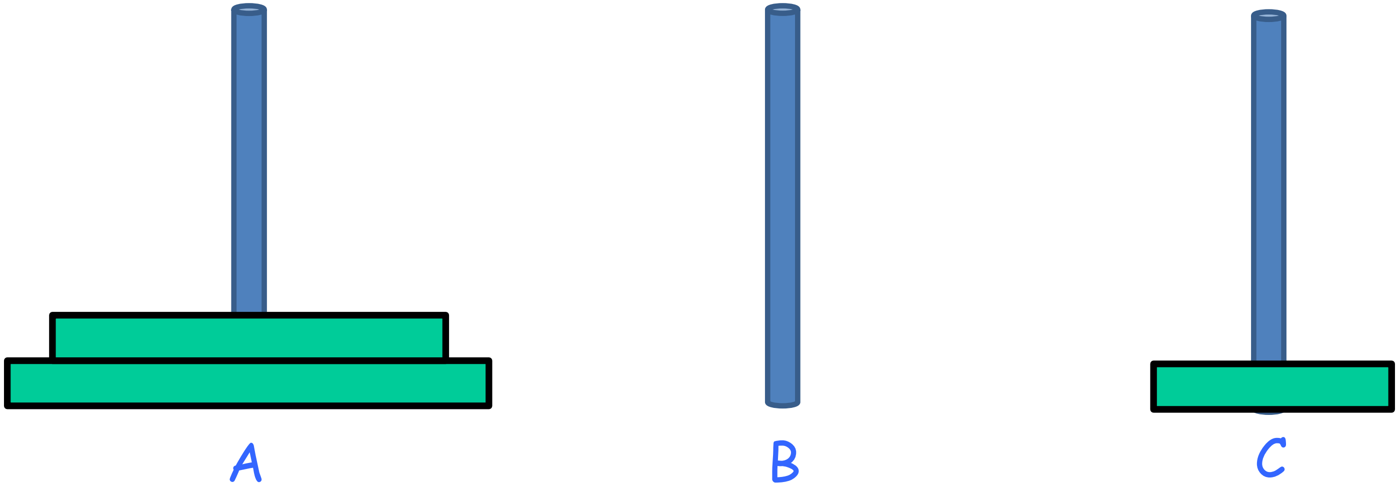
esecuzione dell'algoritmo

$n = 3$



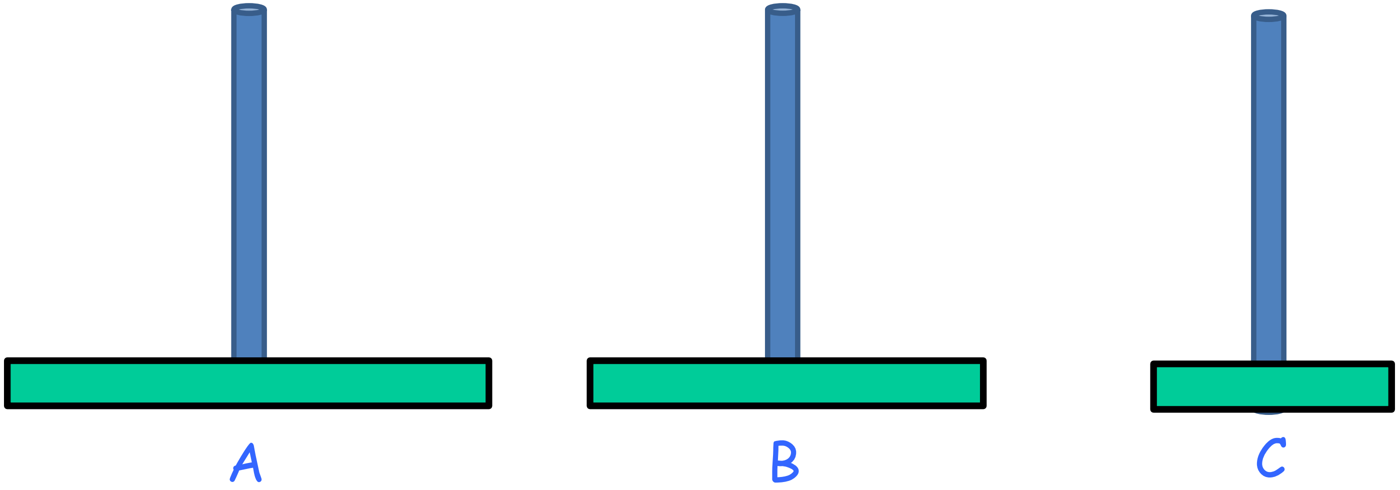
esecuzione dell'algoritmo

n = 3



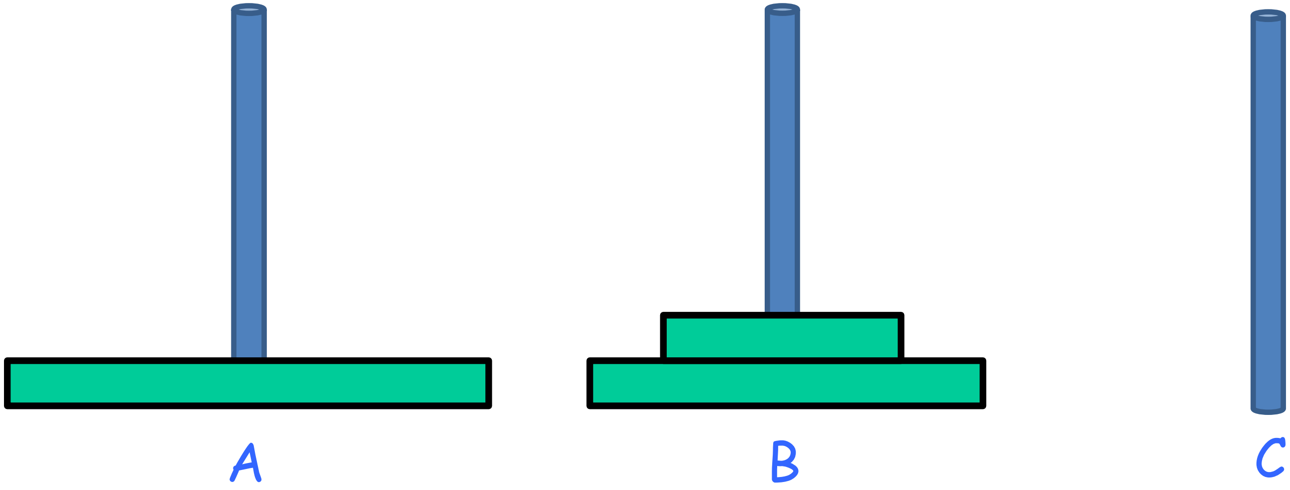
esecuzione dell'algoritmo

$n = 3$



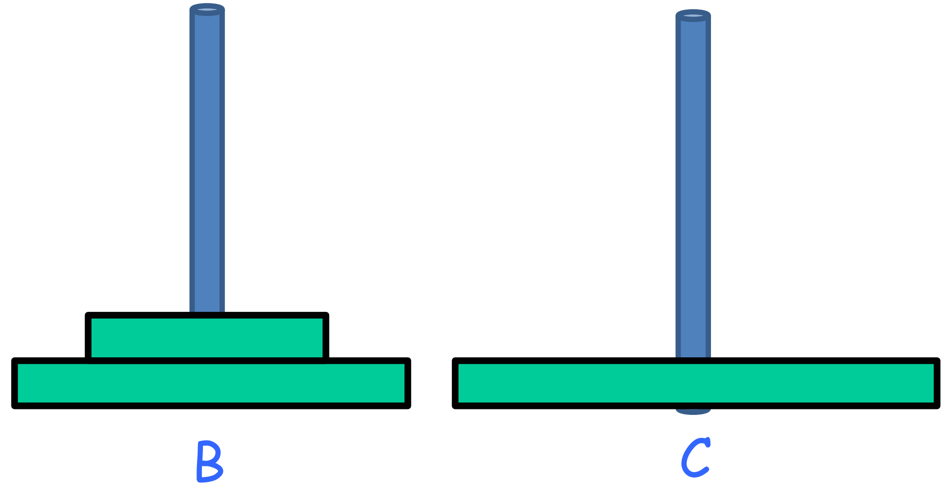
esecuzione dell'algoritmo

$n = 3$



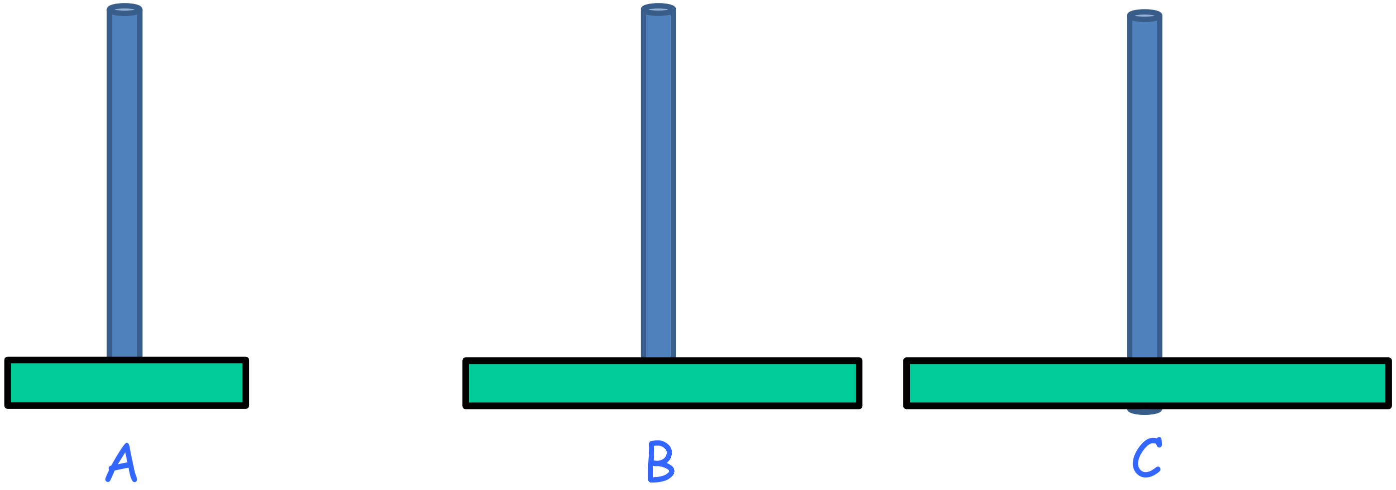
esecuzione dell'algoritmo

$n = 3$



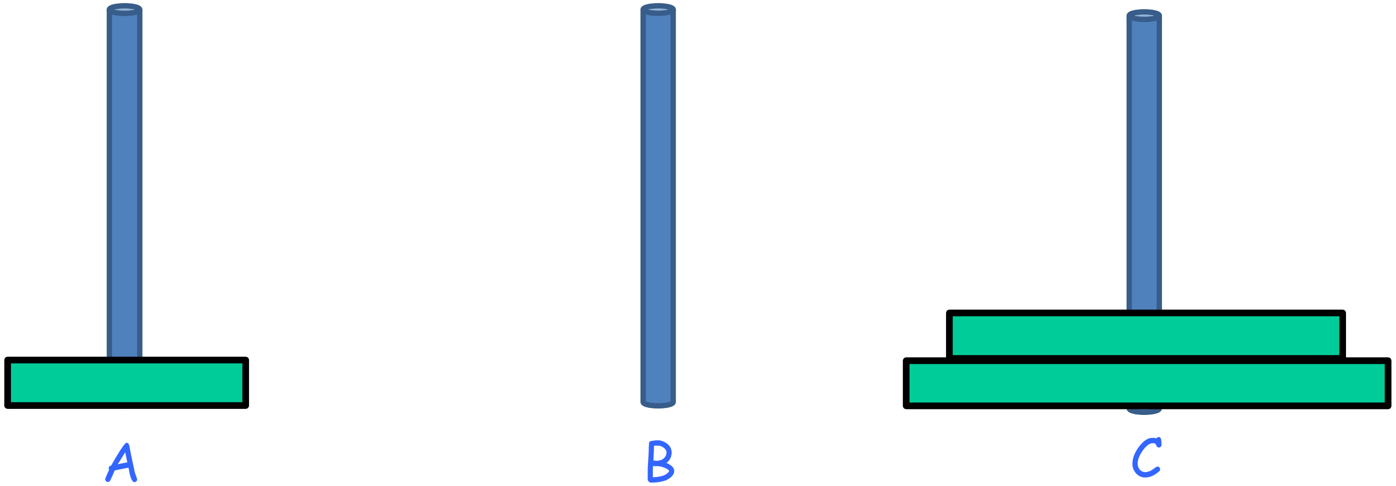
esecuzione dell'algoritmo

$n = 3$



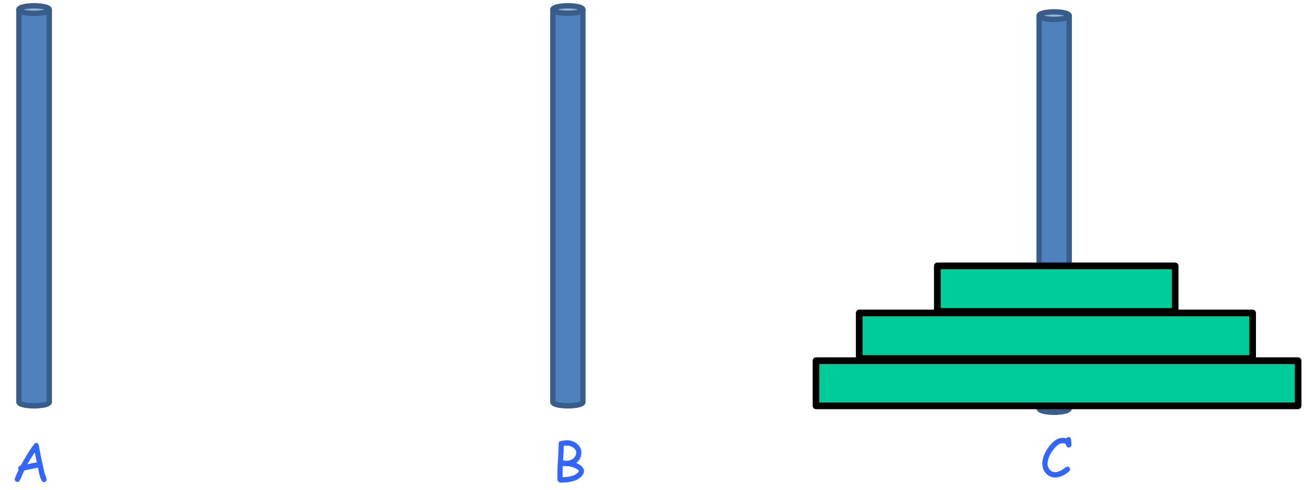
esecuzione dell'algoritmo

$n = 3$



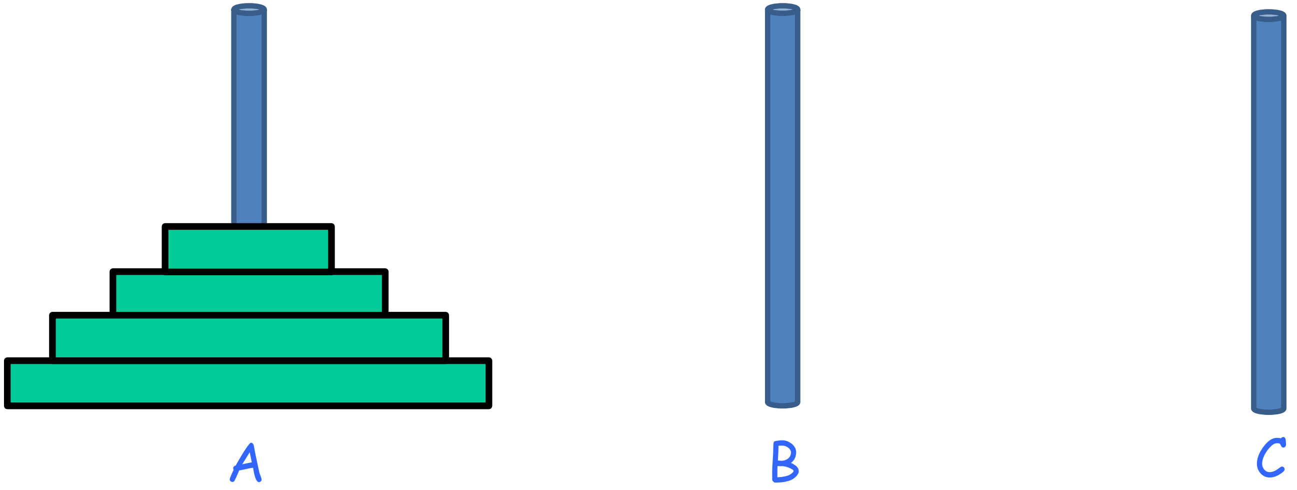
esecuzione dell'algoritmo

$n = 3$



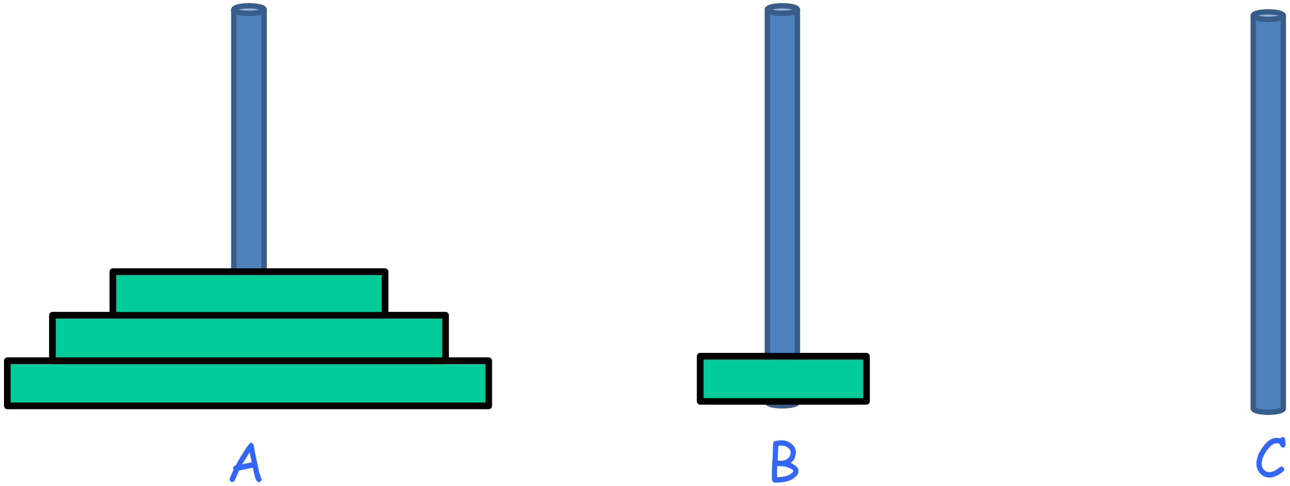
esecuzione dell'algoritmo

$n = 4$



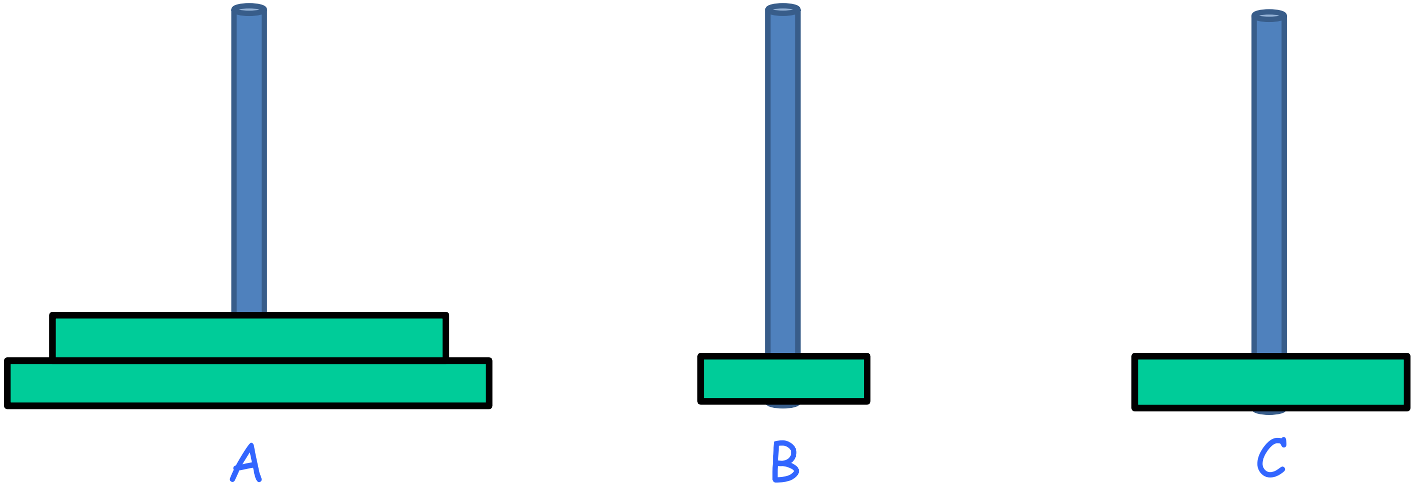
esecuzione dell'algoritmo

$n = 4$



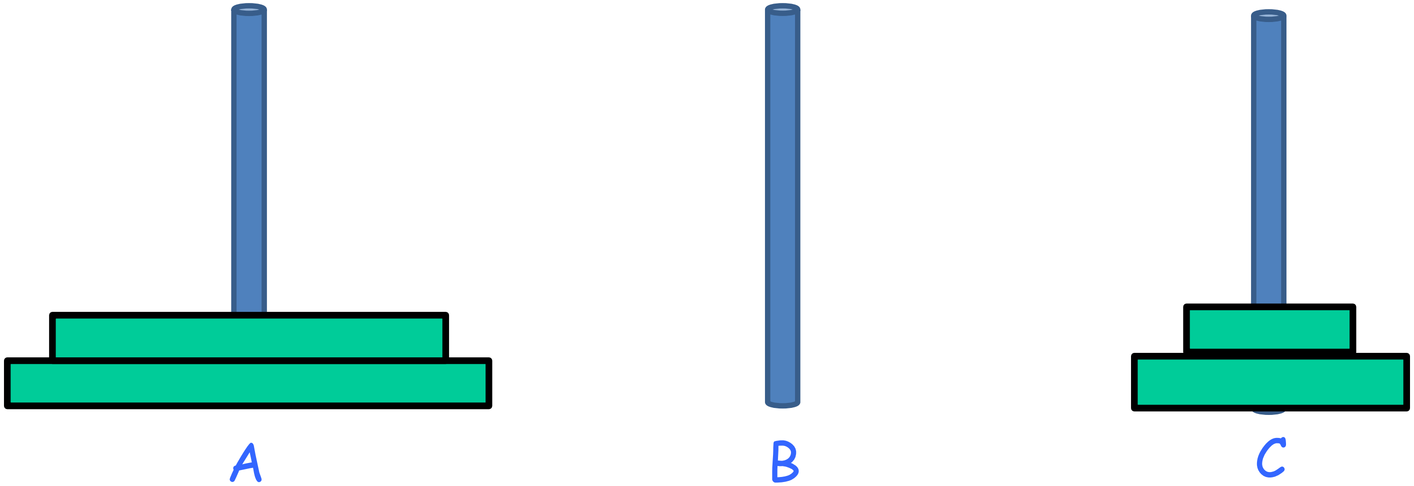
esecuzione dell'algoritmo

$n = 4$



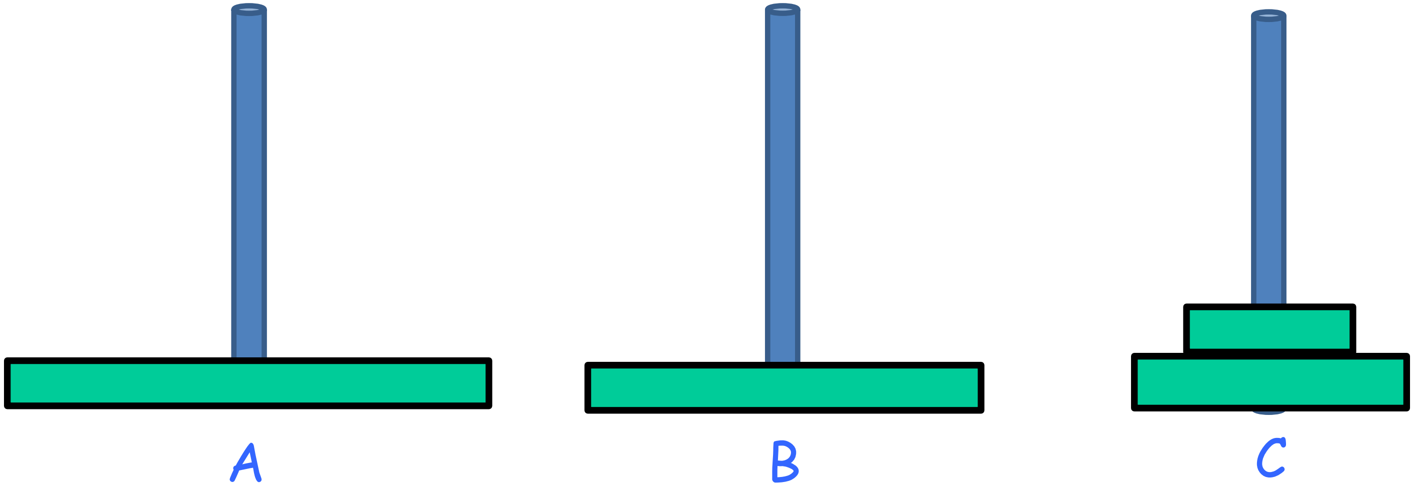
esecuzione dell'algoritmo

$n = 4$



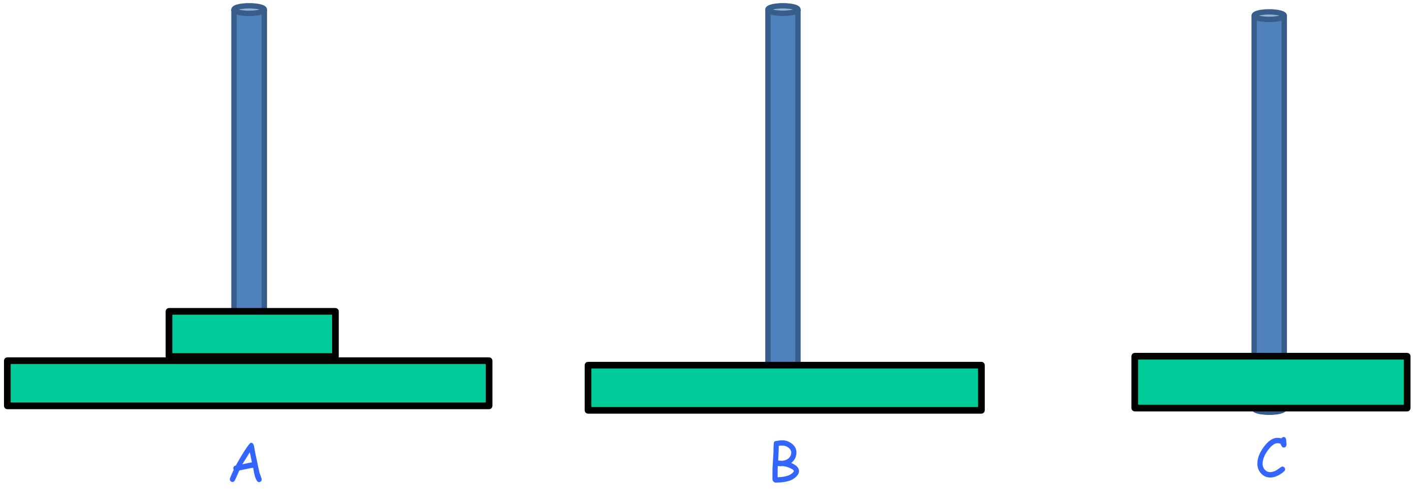
esecuzione dell'algoritmo

n = 4



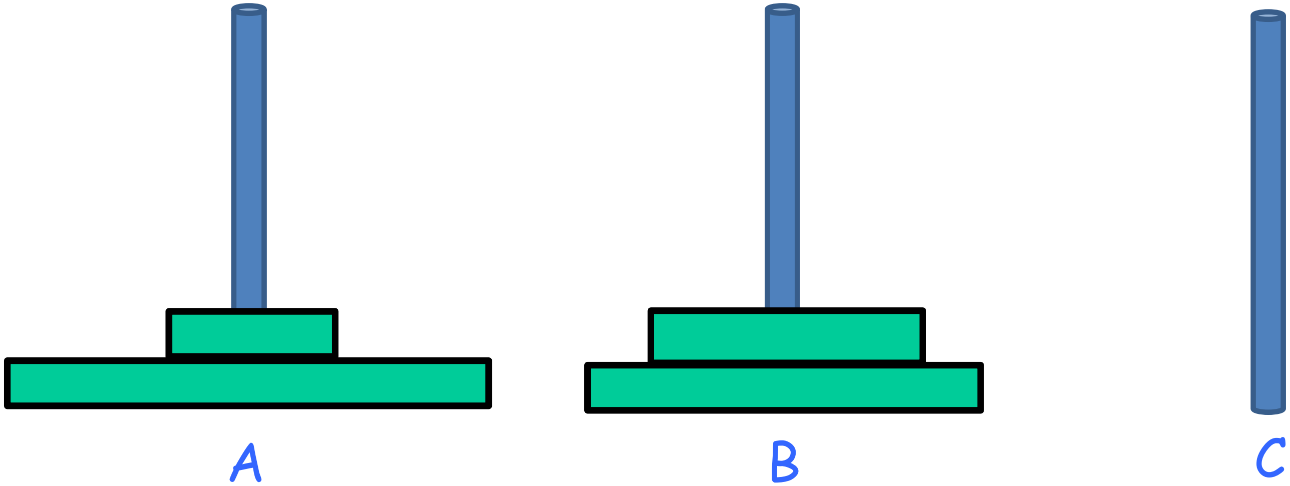
esecuzione dell'algoritmo

$n = 4$



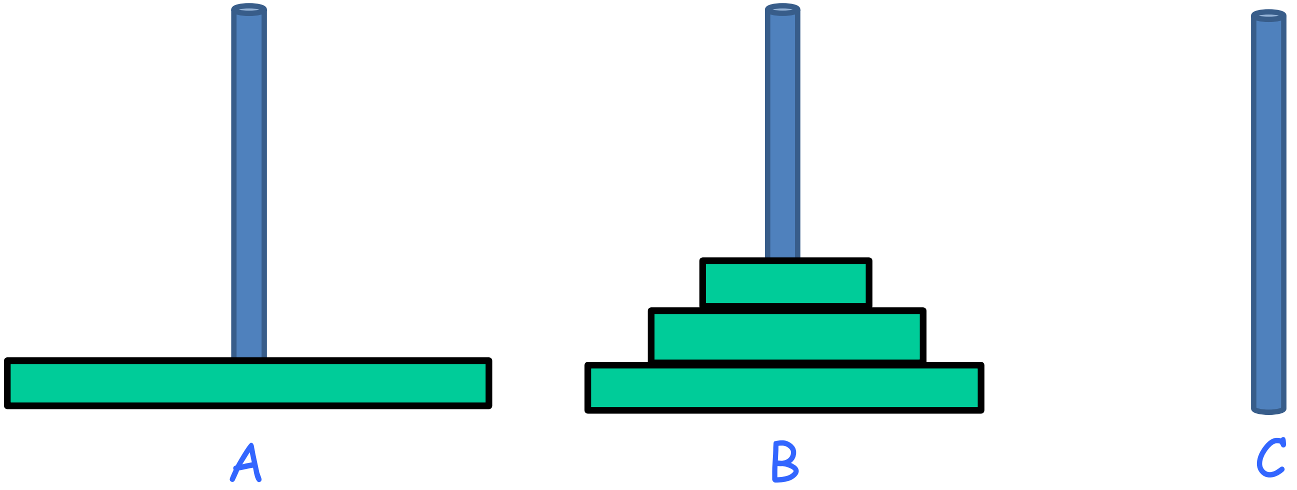
esecuzione dell'algoritmo

$n = 4$



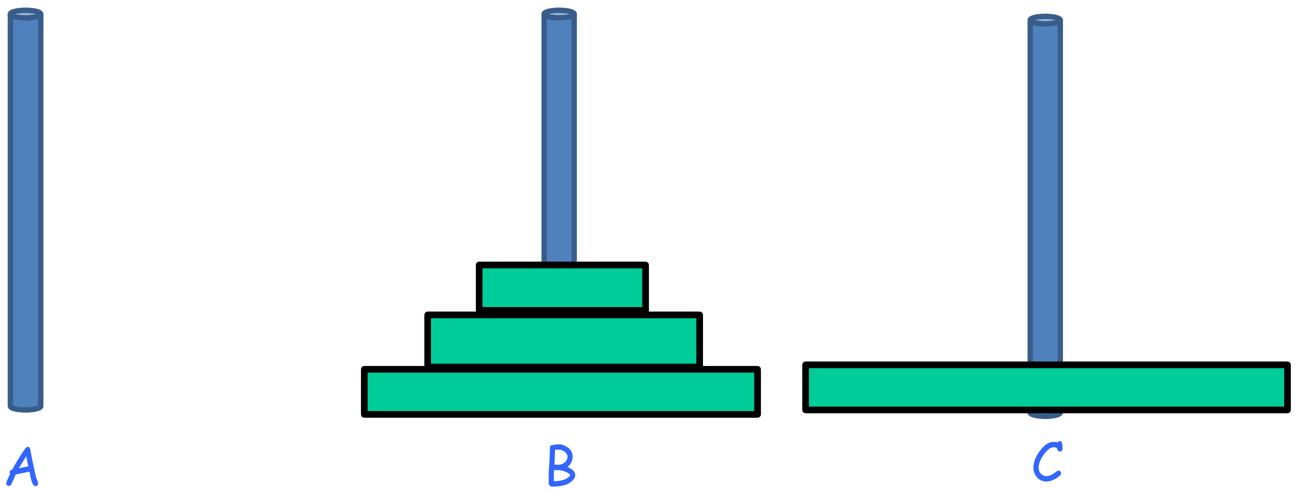
esecuzione dell'algoritmo

$n = 4$



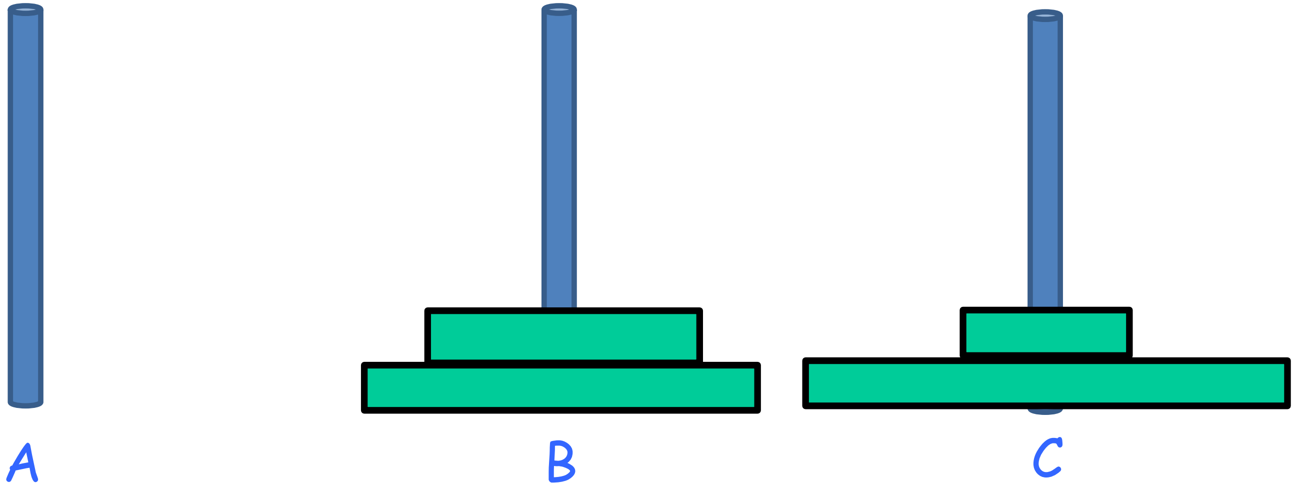
esecuzione dell'algoritmo

$n = 4$



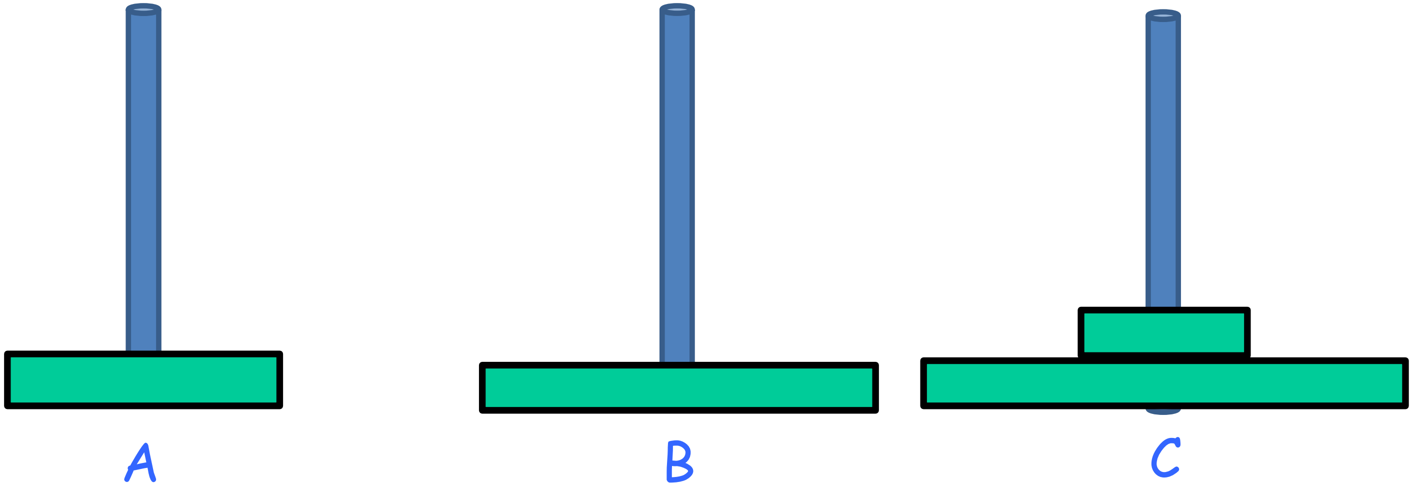
esecuzione dell'algoritmo

$n = 4$



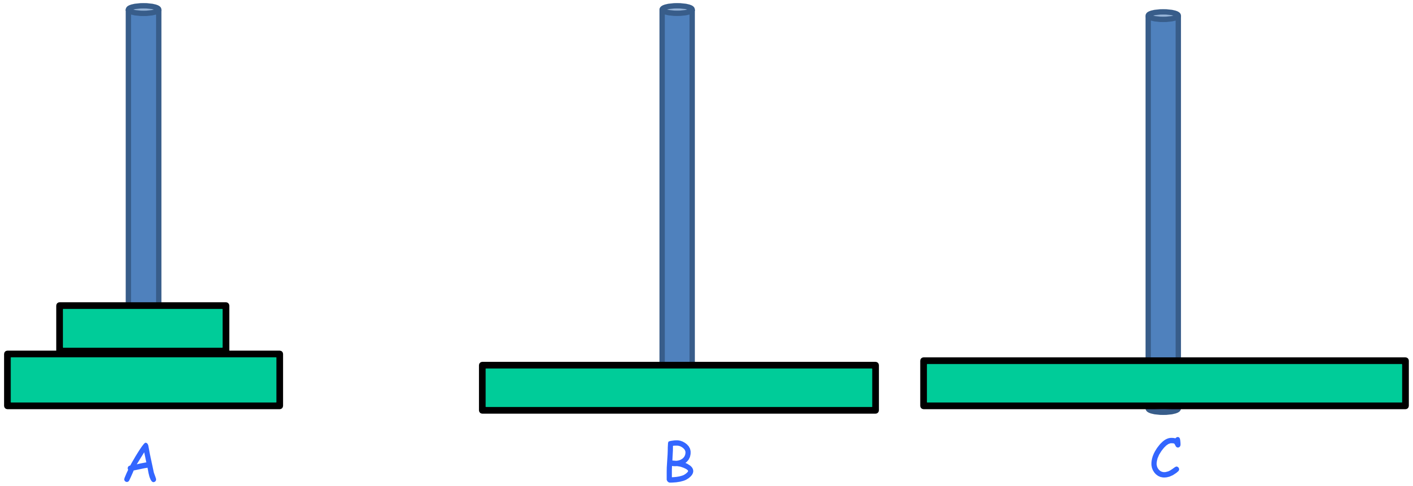
esecuzione dell'algoritmo

$n = 4$



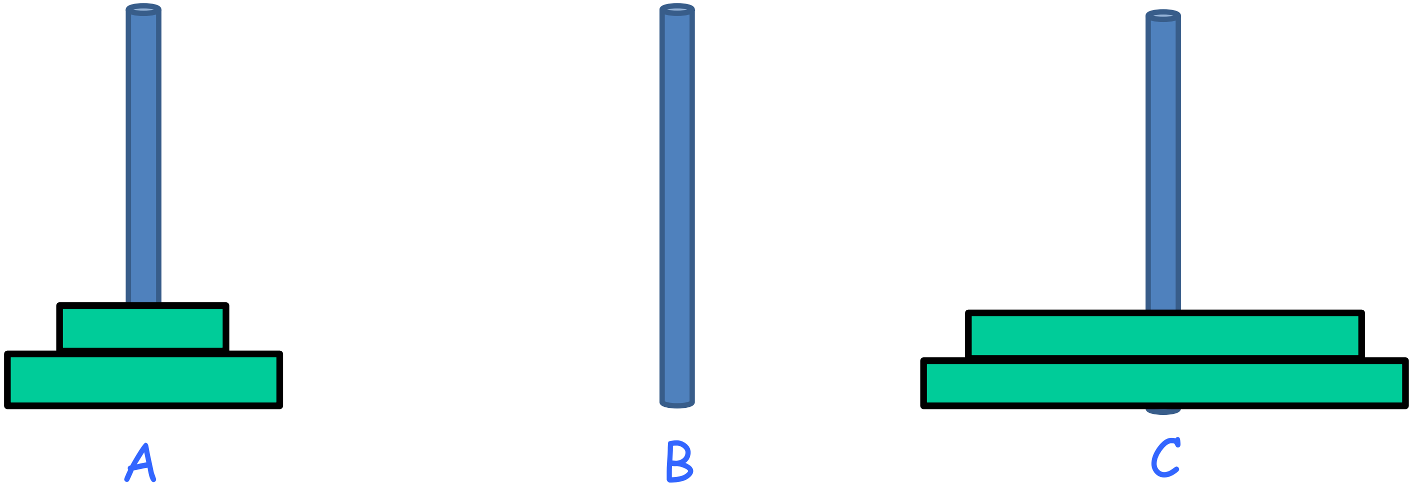
esecuzione dell'algoritmo

$n = 4$



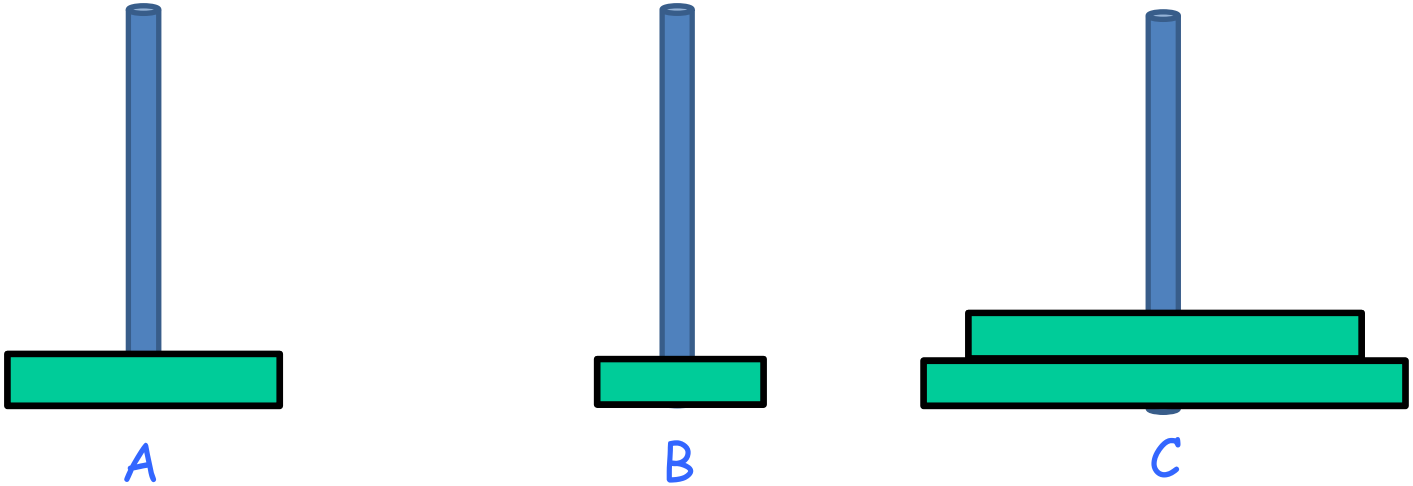
esecuzione dell'algoritmo

n = 4



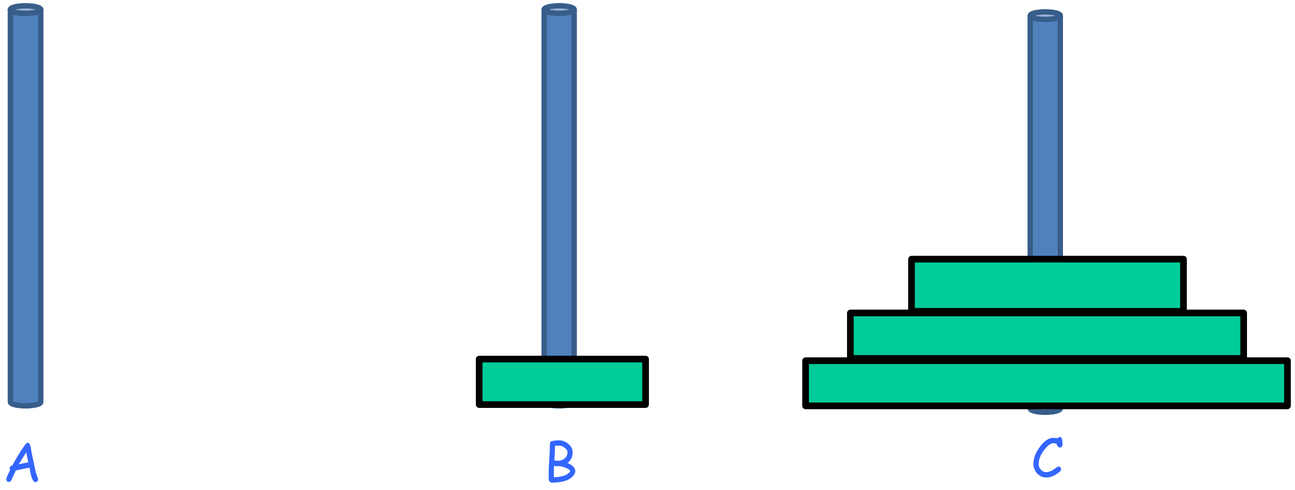
esecuzione dell'algoritmo

$n = 4$



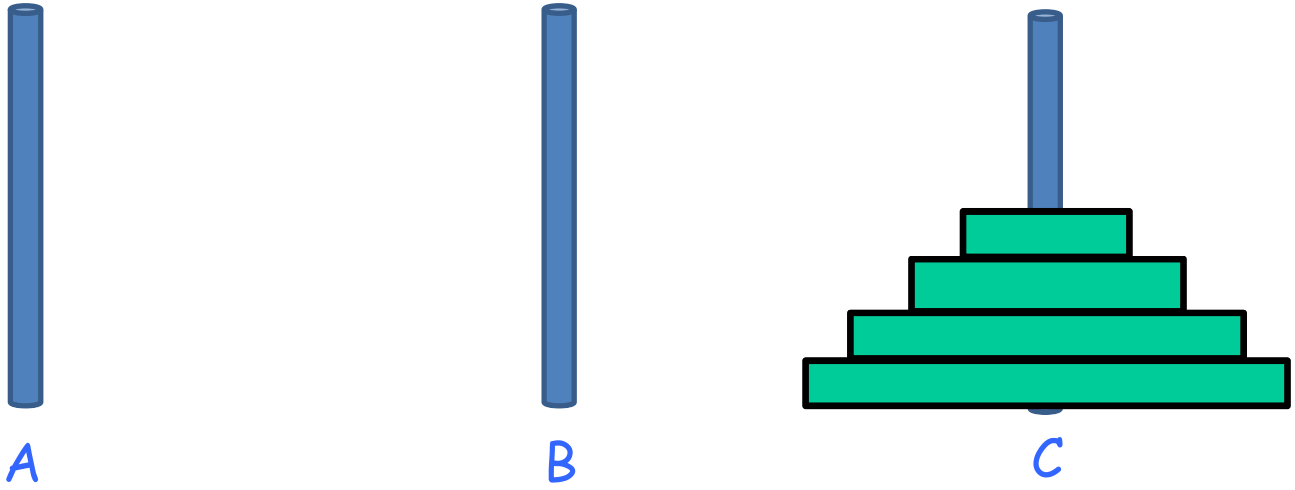
esecuzione dell'algoritmo

$n = 4$

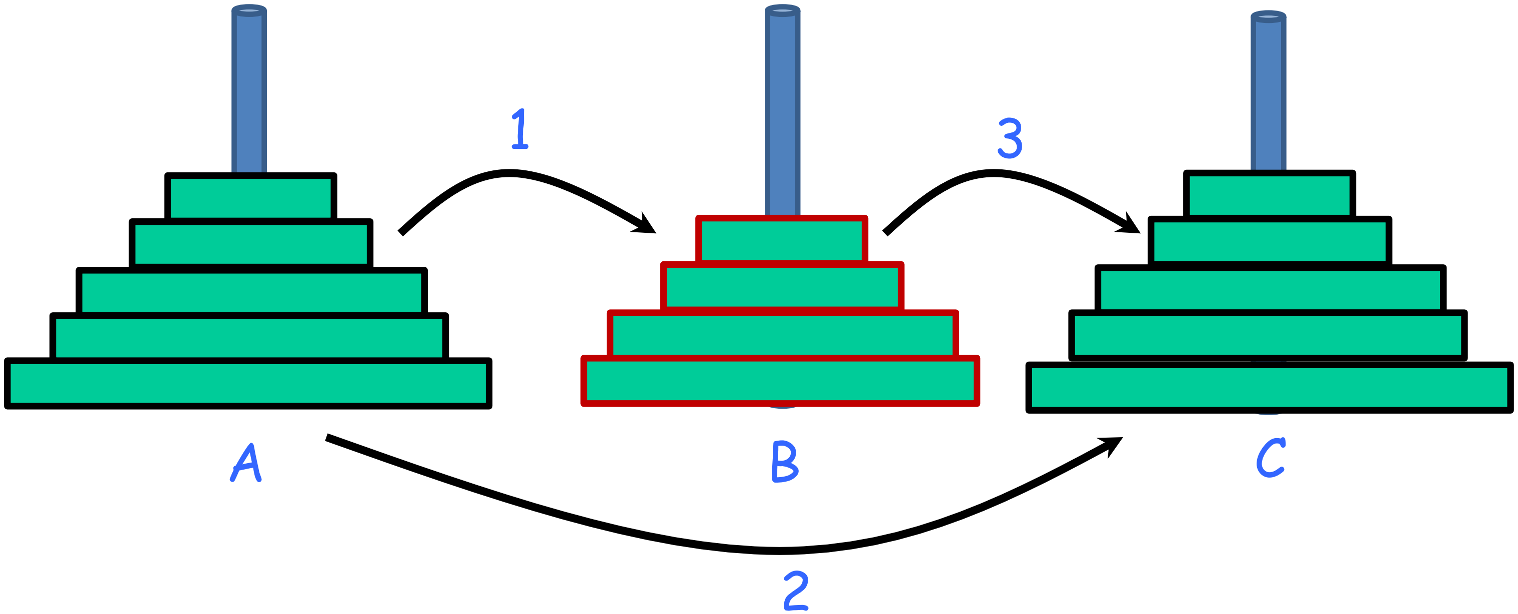


esecuzione dell'algoritmo

n = 4



quanti spostamenti fa l'algoritmo?



T(n): #spostamenti che l'alg fa nel caso peggiore (?) per spostare n dischi

Hanoi(dischi, destinazione, palo ausiliario)

Hanoi ([1,2..,n], C, B)

1. **if** n=1 **then** sposta il disco su C
2. Hanoi([1,2,…,n-1], B, C)
3. sposta il disco n su C
4. Hanoi([1,2,…,n-1], C, A)

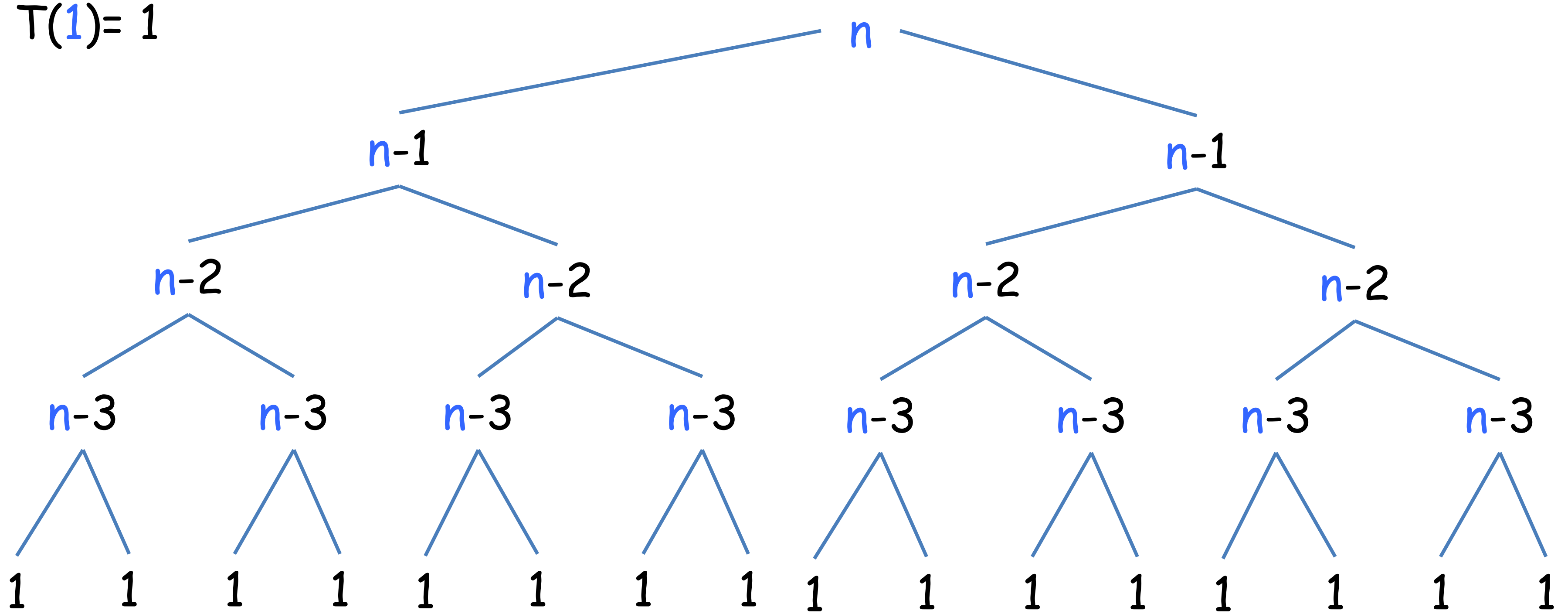
$T(n) = 2T(n-1) + 1$

$T(1) = 1$

analisi (tecnica albero della ricorsione)

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

$T(1) = 1$



albero binario completo!

quanti spostamenti fa ogni nodo? ...uno!

quanto è alto l'albero? ...n-1!

quanti nodi ha un albero binario completo di altezza h?

$$\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

$$\Rightarrow T(n) = 2^n - 1 = \Theta(2^n)$$