Algoritmica

Note su Complessità

G. Prencipe
giuseppe.prencipe@unipi.it

Tempo di esecuzione

- Il ***tempo di esecuzione*** di un programma dipende da:
  - Hardware
  - Compilatore
  - Input
  - Soluzione
  - ….

Computer potente o
Soluzione efficiente?

Partiamo da un esempio

Problema P	Tempo di Esecuzione	Ordine del Tempo di Esecuzione
Algoritmo$_C$	N^3	N^3
Algoritmo$_Q$	N^2	N^2
Algoritmo$_L$	N	N

Solita assunzione che ogni passo costa 1 (modello RAM)

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

- Immaginiamo di avere una scadenza per la soluzione di P
  - Entro il tempo **T**, la soluzione scelta deve terminare

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

- Immaginiamo di avere una scadenza per la soluzione di P
  - Entro il tempo **T**, la soluzione scelta deve terminare

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

Partiamo da un esempio

$T_C(N) = N^3$
$T_Q(N) = N^2$
$T_L(N) = N$

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

- ????

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

- $N^3 = T \quad \Rightarrow \quad N = \sqrt[3]{T}$
- $N^2 = T \quad \Rightarrow \quad N = \sqrt{T}$
- $N = T \quad \Rightarrow \quad N = T$

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

- $N^3 = T \quad \Rightarrow \quad N_C = \sqrt[3]{T}$
- $N^2 = T \quad \Rightarrow \quad N_Q = \sqrt{T}$
- $N = T \quad \Rightarrow \quad N_L = T$

Partiamo da un esempio

$T_C(N) = N^3$
$T_Q(N) = N^2$
$T_L(N) = N$

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

- $N^3 = T \quad \Rightarrow \quad N_C = \sqrt[3]{T}$
- $N^2 = T \quad \Rightarrow \quad N_Q = \sqrt{T}$
- $N = T \quad \Rightarrow \quad N_L = T$

Aggiungiamo pedici per distinguere le tre "N"

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

$N_C = \sqrt[3]{T}$ $N_Q = \sqrt{T}$ $N_L = T$

Cosa possiamo dedurre?

….al crescere di T….

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

Quanti elementi ognuna delle tre soluzioni è in grado di processare nel tempo **T**?

$N_C = \sqrt[3]{T}$ $N_Q = \sqrt{T}$ $N_L = T$

Cosa possiamo dedurre?

….al crescere di T, le tre soluzioni permettono di elaborare più dati, ma con una **diversa velocità polinomiale**

Partiamo da un esempio

$N_C = \sqrt[3]{T}$ $N_Q = \sqrt{T}$ $N_L = T$

$T_C(N) = N^3$
$T_Q(N) = N^2$
$T_L(N) = N$

....al crescere di T, le tre soluzioni permettono di elaborare più dati, ma con una **diversa velocità polinomiale**

1 operazione atomica impiega 1**s**

	1 ora	10 ore	100 ore
N^3			
N^2			
N	????	????	????

Partiamo da un esempio

$N_C = \sqrt[3]{T}$ $N_Q = \sqrt{T}$ $N_L = T$

$T_C(N) = N^3$
$T_Q(N) = N^2$
$T_L(N) = N$

….al crescere di T, le tre soluzioni permettono di elaborare più dati, ma con una **diversa velocità polinomiale**

1 operazione atomica impiega 1**s**

	1 ora	10 ore	100 ore
N^3	15,32	33,01	71,13
N^2	60	189,73	600
N	3600	36000	360000

Partiamo da un esempio

$N_C = \sqrt[3]{T}$ $N_Q = \sqrt{T}$ $N_L = T$

$T_C(N) = N^3$
$T_Q(N) = N^2$
$T_L(N) = N$

Supponiamo ora di acquistare un calcolatore k volte più potente….(ad esempio k=10)

1 operazione atomica impiega 0.1**s**

	1 ora	10 ore	100 ore
N^3	33,01	71,13	153,26
N^2	189,73	600	1897,36
N	36000	360000	3600000

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

Supponiamo ora di acquistare un calcolatore k volte più potente….

- Questo è equivalente a usare k volte di più il vecchio calcolatore

$$N_C = \sqrt[3]{kT} \quad N_Q = \sqrt{kT} \quad N_L = kT$$

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

Supponiamo ora di acquistare un calcolatore k volte più potente….

- Questo è equivalente a usare k volte di più il vecchio calcolatore

$$N_C = \sqrt[3]{kT} \quad N_Q = \sqrt{kT} \quad N_L = kT$$

$$N_C = \sqrt[3]{k}\,\sqrt[3]{T} \quad N_Q = \sqrt{k}\,\sqrt{T} \quad N_L = kT$$

Partiamo da un esempio

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

$N_C = \sqrt[3]{k}\,\sqrt[3]{T}$ $N_Q = \sqrt{k}\,\sqrt{T}$ $N_L = kT$

Il vantaggio ottenuto impiegando il calcolatore più veloce è tanto maggiore quanto migliore è l'algoritmo impiegato!!!!

E quindi?

Legge di Moore (co-fondatore di Intel, 1965)

Prestazioni (ad es., #transistori per chip) **dei calcolatori raddoppiano ogni 18 mesi**

E quindi?

Legge di Moore (co-fondatore di Intel, 1965)

Prestazioni (ad es., #transistori per chip) **dei calcolatori raddoppiano ogni 18 mesi**

- Quindi, conviene concentrarsi da subito sulla progettazione di algoritmi efficienti per P piuttosto che attendere calcolatori più potenti!!!!

Ultima considerazione

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

$N_C = \sqrt[3]{k}\,\sqrt[3]{T}$ $N_Q = \sqrt{k}\,\sqrt{T}$ $N_L = kT$

- Quanto più potente deve essere il nuovo calcolatore per poter eseguire T_C nello stesso tempo impiegato dal vecchio per eseguire T_L?

Ultima considerazione

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

$N_C = \sqrt[3]{k}\ \sqrt[3]{T}$ $N_Q = \sqrt{k}\ \sqrt{T}$ $N_L = kT$

- Quanto più potente deve essere il nuovo calcolatore per poter eseguire T_C nello stesso tempo impiegato dal vecchio per eseguire T_L?
  - $\sqrt[3]{kT} = T$

Ultima considerazione

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

$N_C = \sqrt[3]{k}\,\sqrt[3]{T}$ $N_Q = \sqrt{k}\,\sqrt{T}$ $N_L = kT$

- Quanto più potente deve essere il nuovo calcolatore per poter eseguire T_C nello stesso tempo impiegato dal vecchio per eseguire T_L?
  - $\sqrt[3]{kT} = T \Rightarrow \sqrt[3]{k} = T / \sqrt[3]{T}$

Ultima considerazione

$$T_C(N) = N^3$$
$$T_Q(N) = N^2$$
$$T_L(N) = N$$

$N_C = \sqrt[3]{k}\,\sqrt[3]{T}$ $\quad N_Q = \sqrt{k}\,\sqrt{T}$ $\quad N_L = kT$

- Quanto più potente deve essere il nuovo calcolatore per poter eseguire T_C nello stesso tempo impiegato dal vecchio per eseguire T_L?
  - $\sqrt[3]{kT} = T \Rightarrow \sqrt[3]{k} = T / \sqrt[3]{T} \Rightarrow k = T^2$

Ultima considerazione

$k = T^2$

1 op al secondo, $T=N=3600 \Rightarrow k=3600^2 = 12960000$ volte più veloce!

Ultima considerazione

$k = T^2$

1 op al secondo, $T=N=3600 \Rightarrow k=3600^2 = 12960000$ volte più veloce!

Cioè, il nuovo calcolatore deve effettuare 1 op ogni $\sim 10^{-6}$ secondi, che, secondo Moore sono ~34anni!!!!

Ultima considerazione

$k = T^2$

1 op al secondo, $T=N=3600 \Rightarrow k=3600^2 = 12960000$ volte più veloce!

Cioè, il nuovo calcolatore deve effettuare 1 op ogni $\sim 10^{-6}$ secondi, che, secondo Moore sono ~34anni!!!!

Ultima considerazione

$k = T^2$

1 op al secondo, $T=N=3600 \Rightarrow k=3600^2 = 12960000$ volte più veloce!

Cioè, il nuovo calcolatore deve effettuare 1 op ogni $\sim 10^{-6}$ secondi, che, secondo Moore sono ~34anni!!!!

….e senza considerare che la legge di Moore **non può crescere all'infinito**!!!!

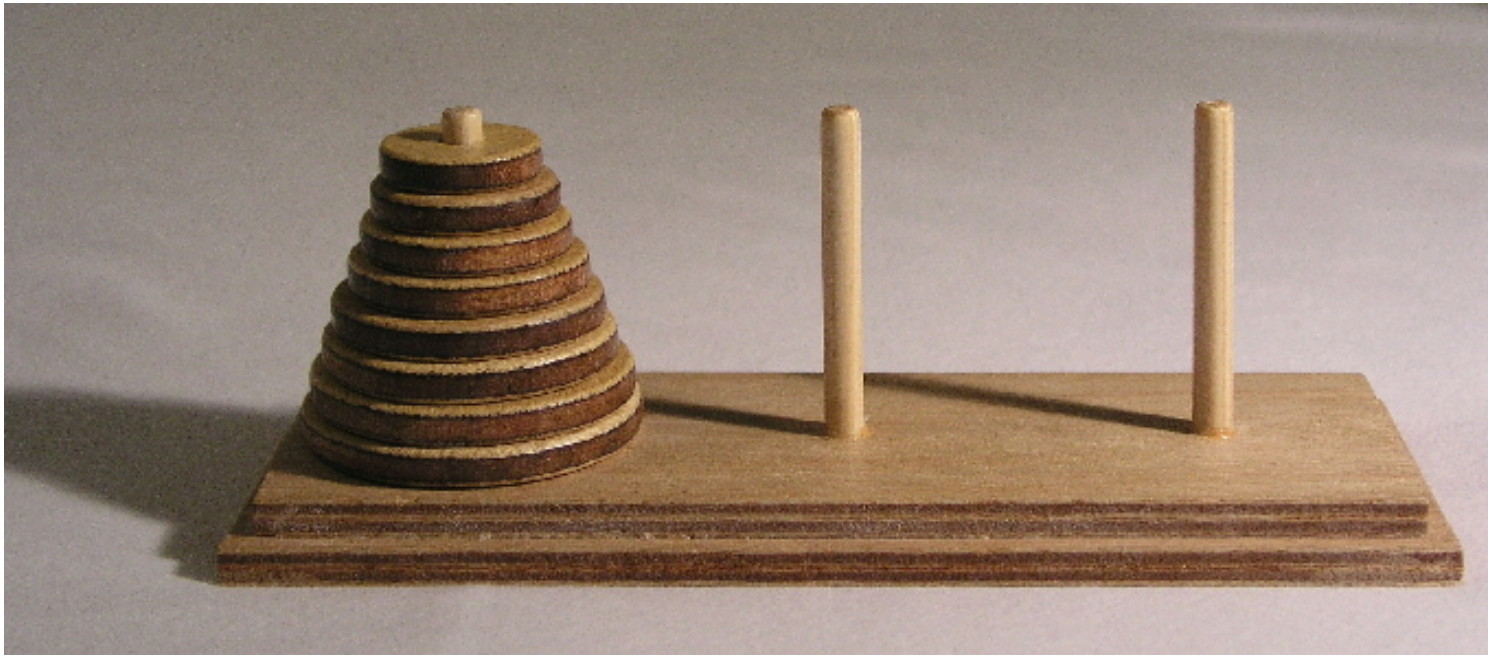
Algoritmica

Ricorsione

G. Prencipe
giuseppe.prencipe@unipi.it

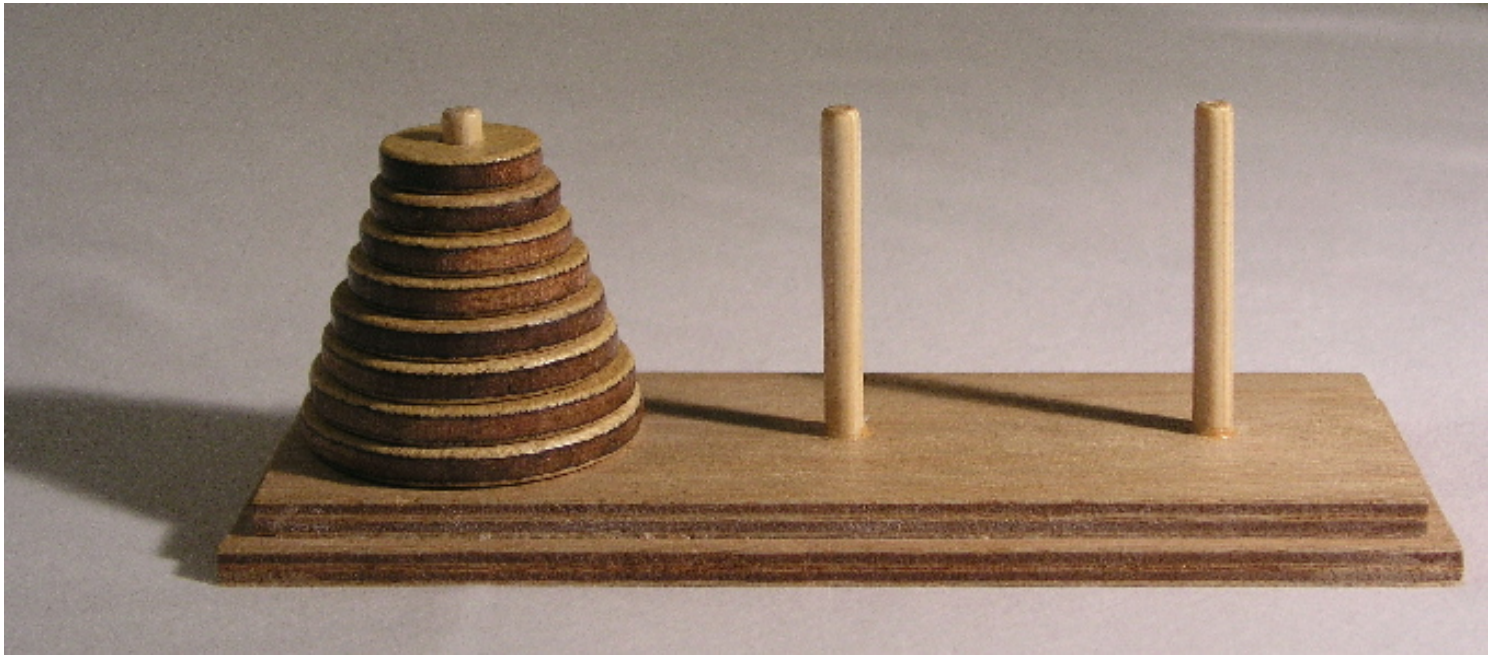
Esempio: Hanoi

- Lo scopo del gioco è di spostare la pila di dischi su un altro piolo, rispettando le seguenti regole:
  - Si può muovere solo un disco alla volta.
  - Una mossa consiste nel prendere un disco da una pila e piazzarlo su un’altra pila.
  - Un disco non può essere piazzato su un disco più piccolo.



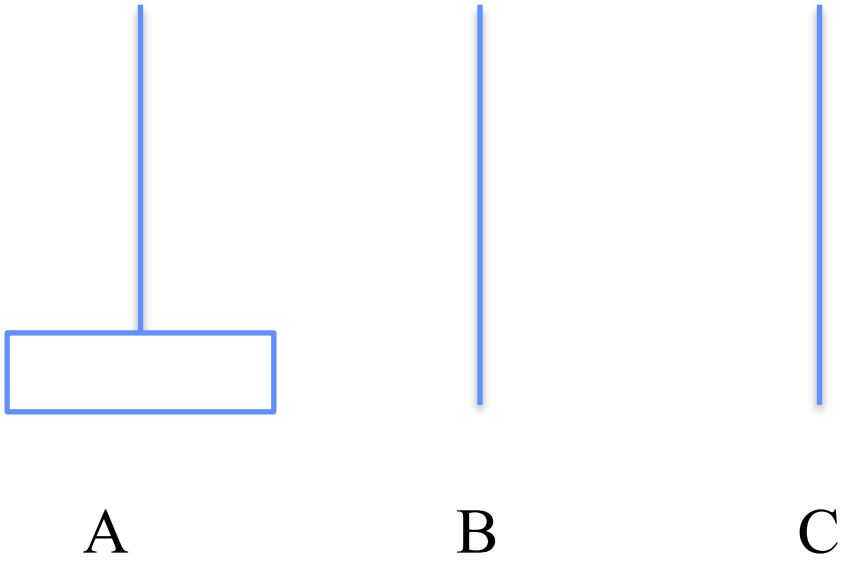
An example: Hanoi

- Il gioco fu inventato dal matematico francese Édouard Lucas nel 1883. La leggenda narra che in un tempio Indiano a Kashi Vishwanath c'è una grande stanza con tre colonne di diamante, e 64 dischi dorati. I sacerdoti *brahmani*, seguendo le direttive di un'antica profezia, spostano da allora questi dischi in accordo con le immutabili regole di *Brahma*. Il gioco è infatti anche noto come Il gioco della torre of Brahma. Secondo la leggenda, quando l'ultima mossa sarà effettuata, il mondo finirà.



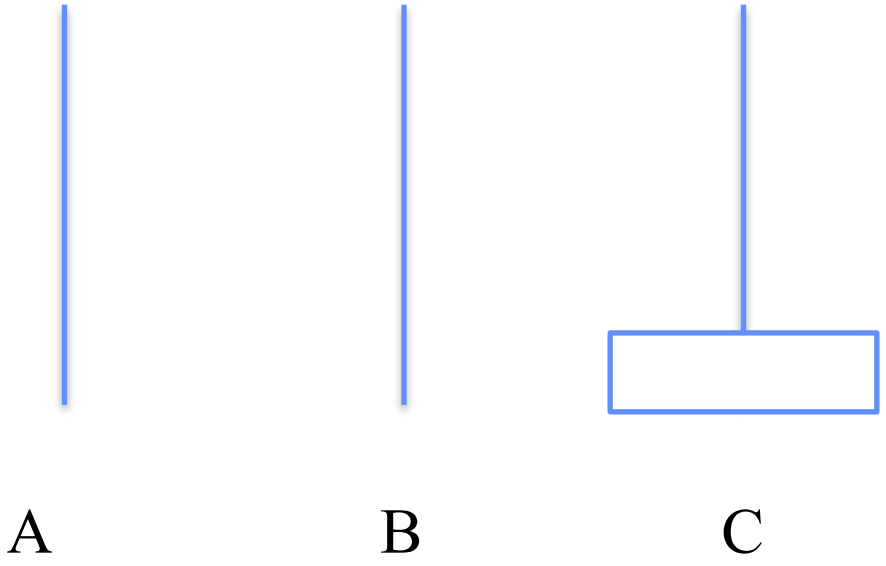
An example: Hanoi

- Per N=1, facile!



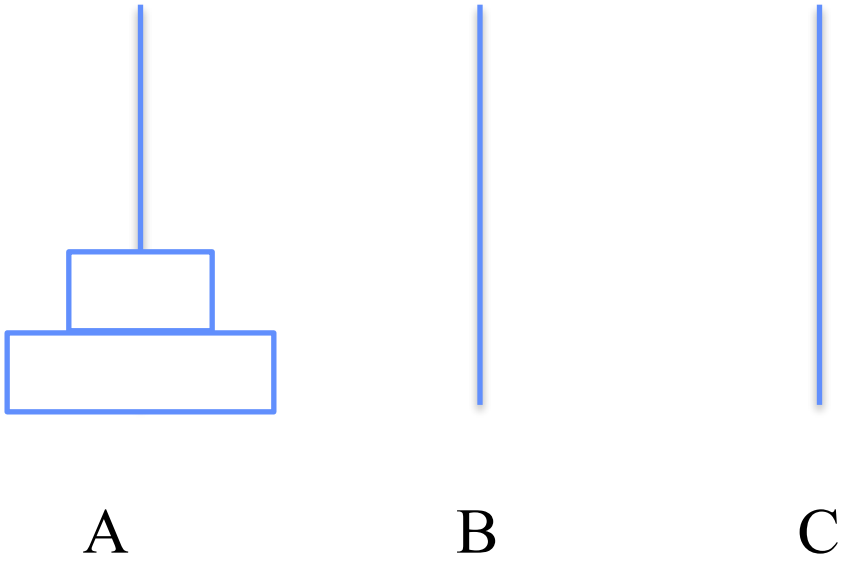
An example: Hanoi

- Per N=1, facile!
  - Sposta l'unico disco da A a C!



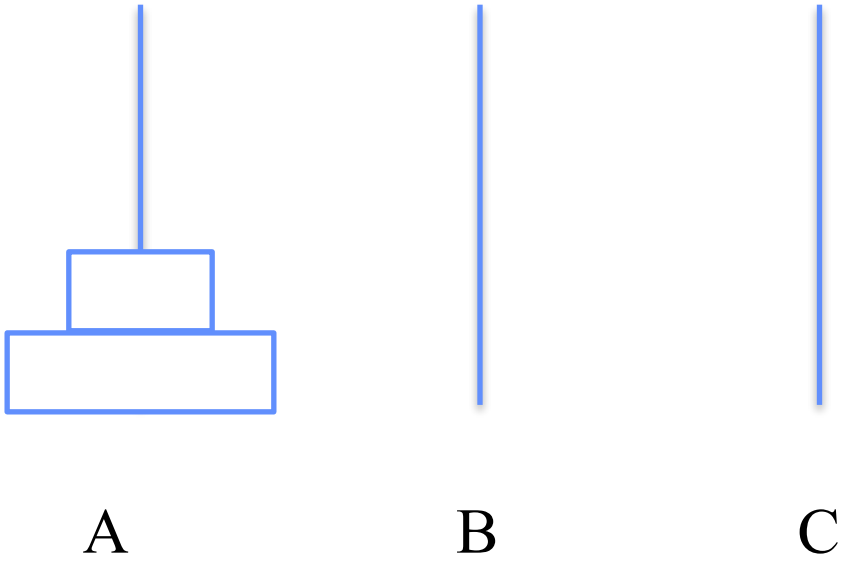
An example: Hanoi

- Conoscendo la soluzione per N=1, possiamo risolvere per N = 2?



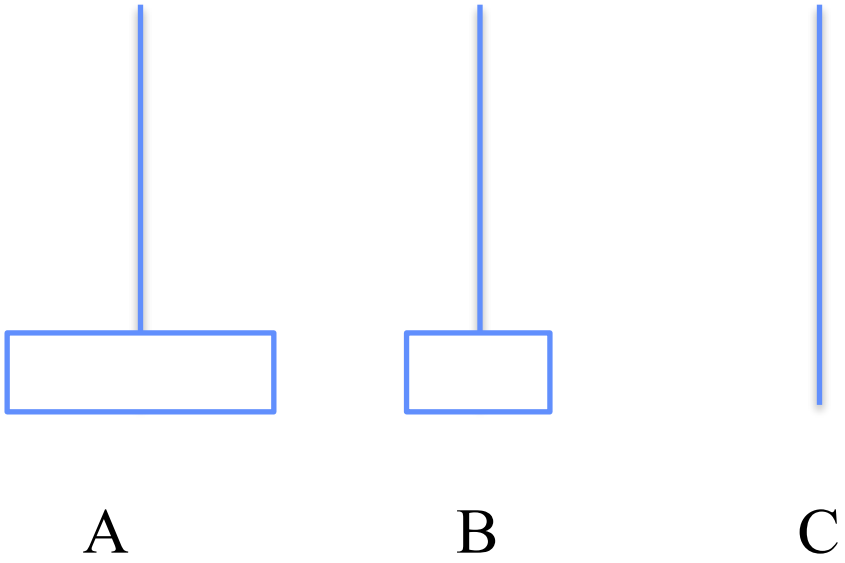
An example: Hanoi

- Conoscendo la soluzione per N=1, possiamo risolvere per N = 2?
  - Usa B come piolo di supporto



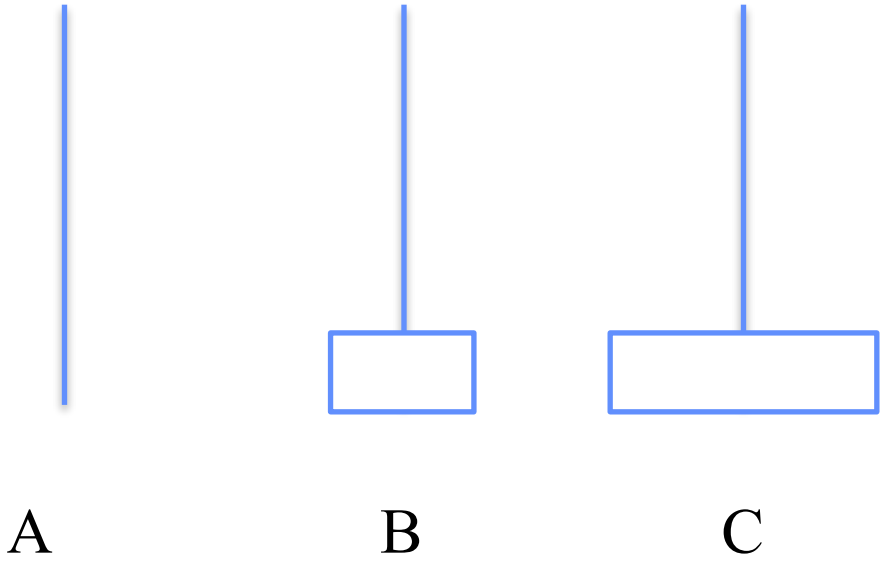
An example: Hanoi

- Conoscendo la soluzione per N=1, possiamo risolvere per N = 2?
  - Usa B come piolo di supporto



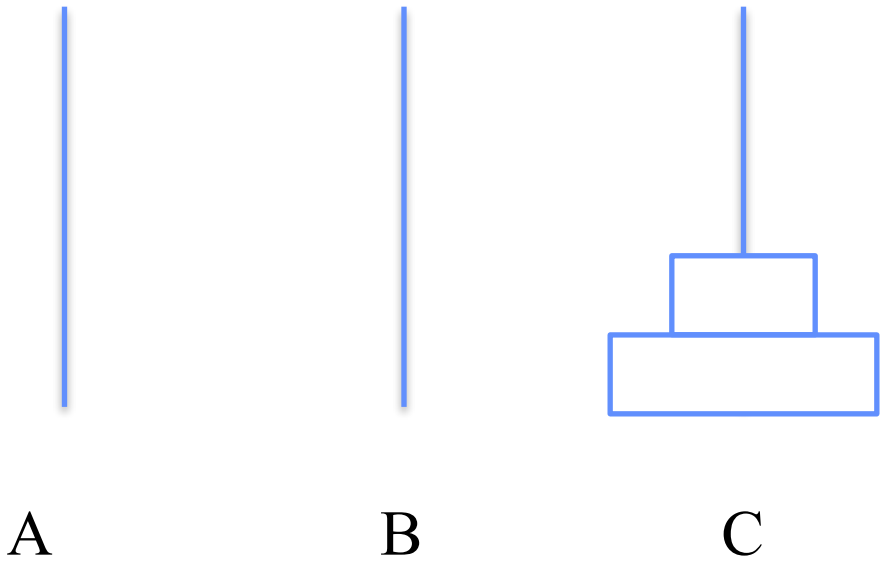
An example: Hanoi

- Conoscendo la soluzione per N=1, possiamo risolvere per N = 2?
  - Usa B come piolo di supporto



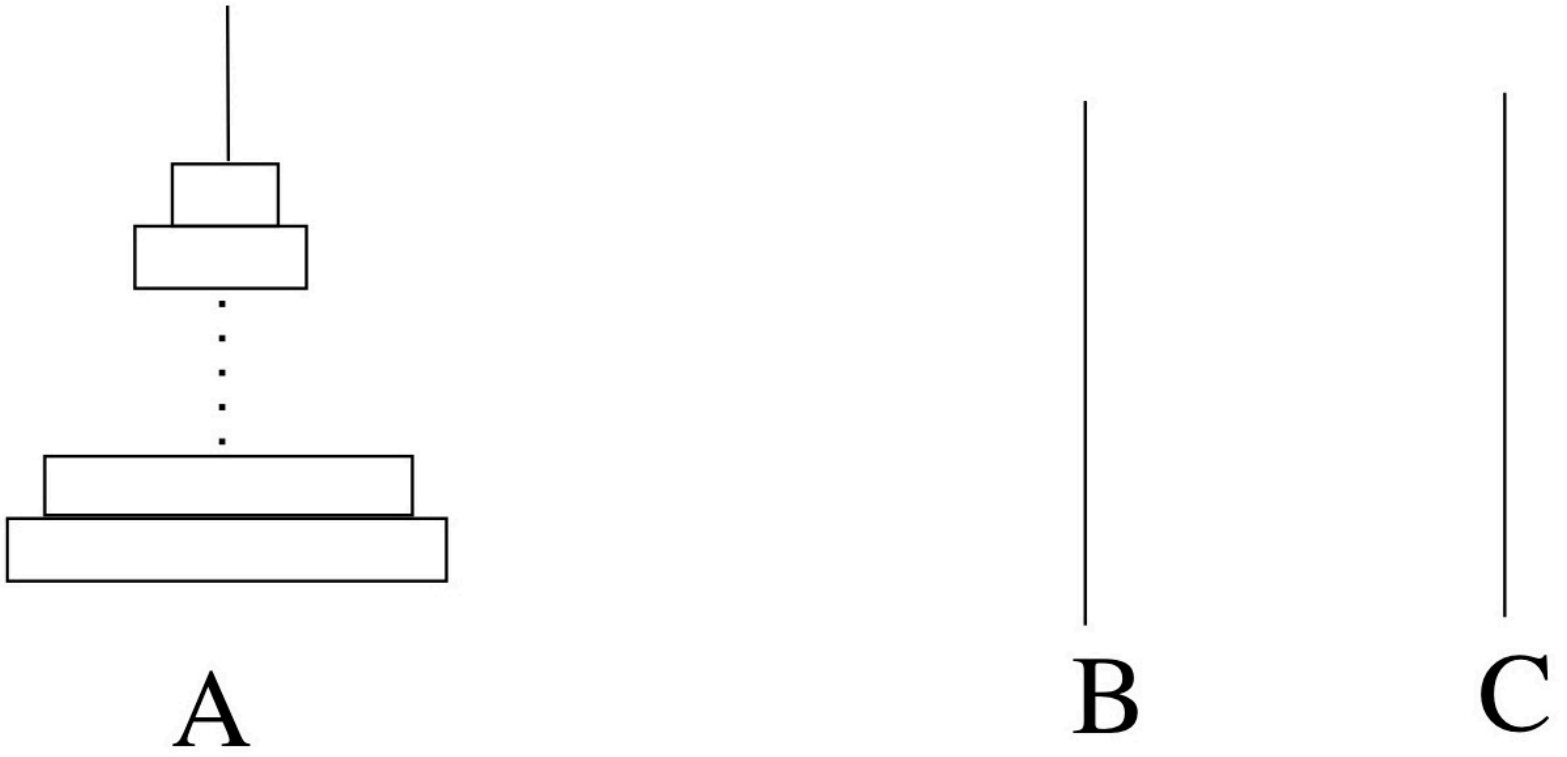
An example: Hanoi

- Conoscendo la soluzione per N=1, possiamo risolvere per N = 2?
  - Usa B come piolo di supporto



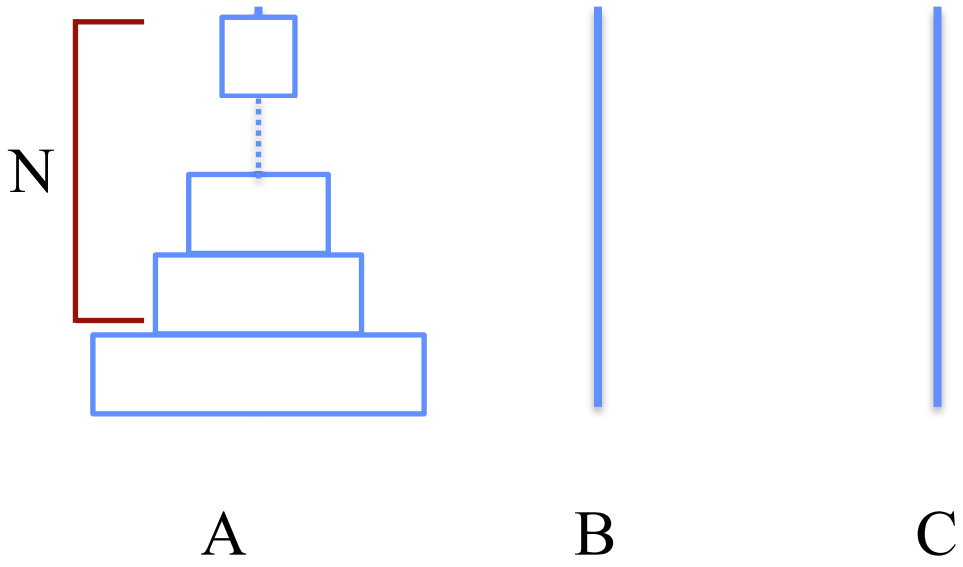
An example: Hanoi

- Come generalizzare per N > 2?
  - Suggerimento: argomento ricorsivo!



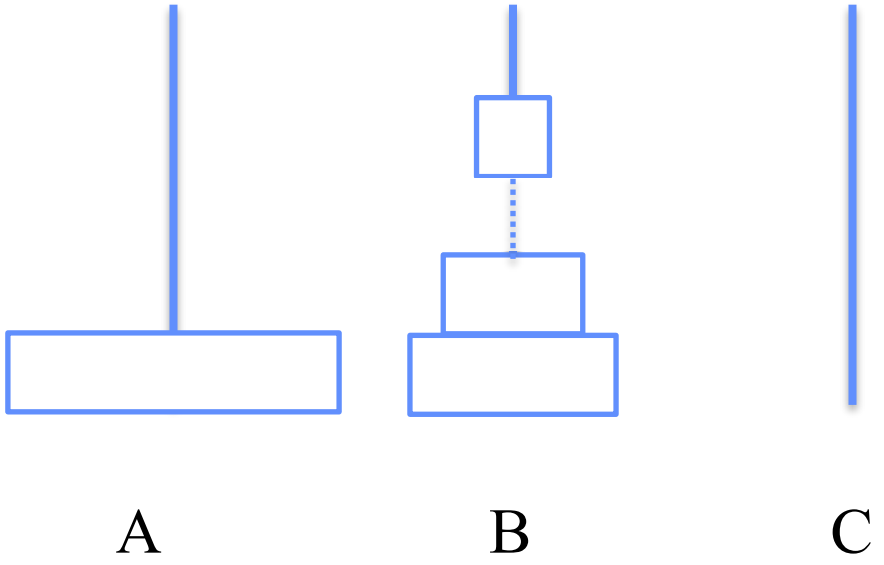
An example: Hanoi

- Come generalizzare per N > 2?
  - Se sapessimo risolvere per N, possiamo risolvere per N+1?



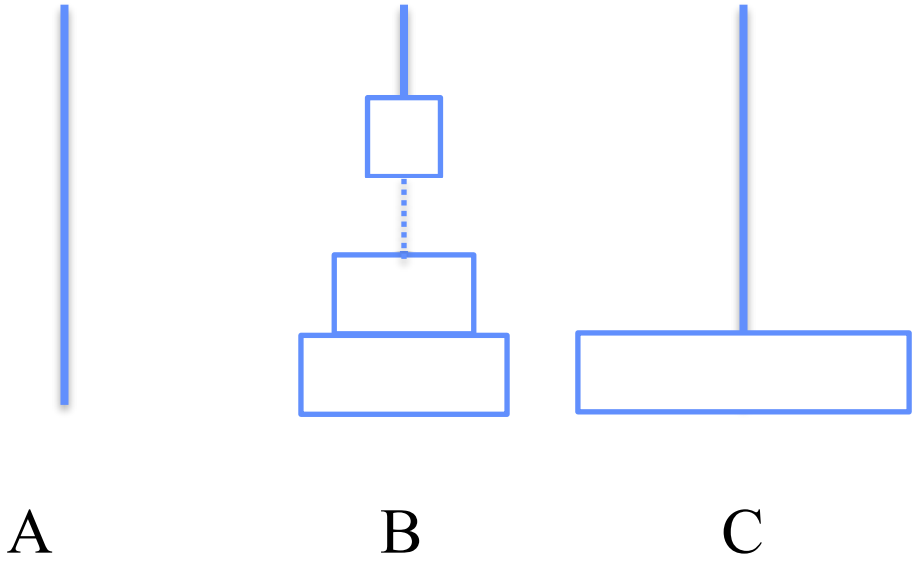
An example: Hanoi

- Come generalizzare per N > 2?
  - Se sapessimo risolvere per N, possiamo risolvere per N+1?



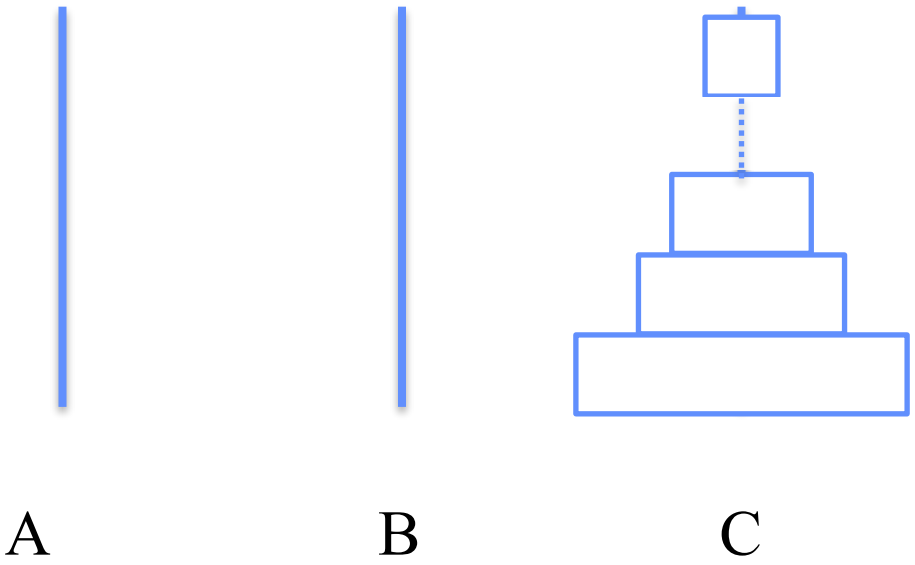
An example: Hanoi

- Come generalizzare per N > 2?
  - Se sapessimo risolvere per N, possiamo risolvere per N+1?



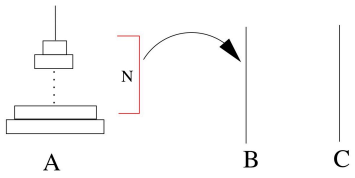
An example: Hanoi

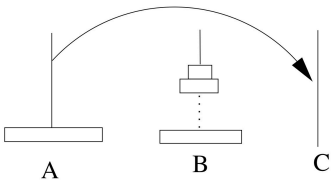
- Come generalizzare per N > 2?
  - Se sapessimo risolvere per N, possiamo risolvere per N+1?

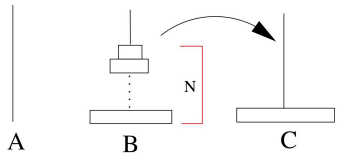


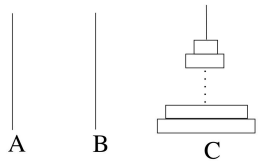
An example: Hanoi

sposta n dischi da
A a C, usando B







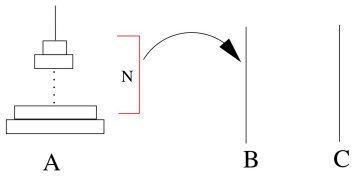


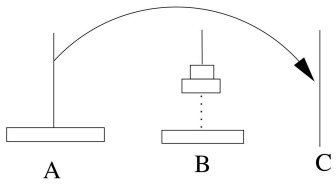

```
hanoi(n, A, B, C):
    if (n ==1)
        sposta da A a C;
    else{
       ????
        sposta da A a C;
       ????
    }
```

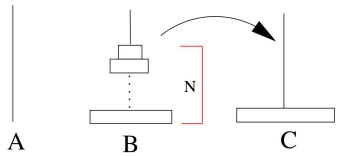
An example: Hanoi

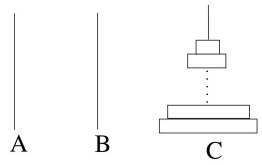
sposta n dischi da
A a C, usando B

```
hanoi(n, A, B, C):
    if (n ==1)
        sposta da A a C;
    else{
        hanoi(n-1, A, C, B);
        sposta da A a C;
        hanoi(n-1, B, A, C);
    }
```

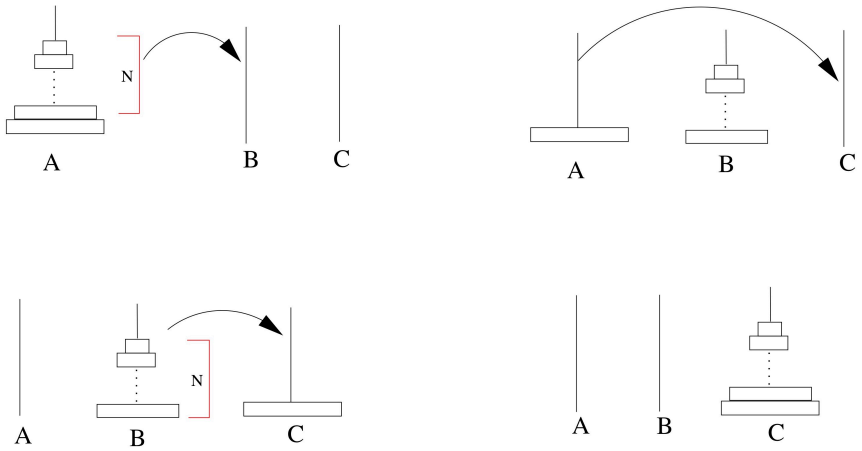








An example: Hanoi

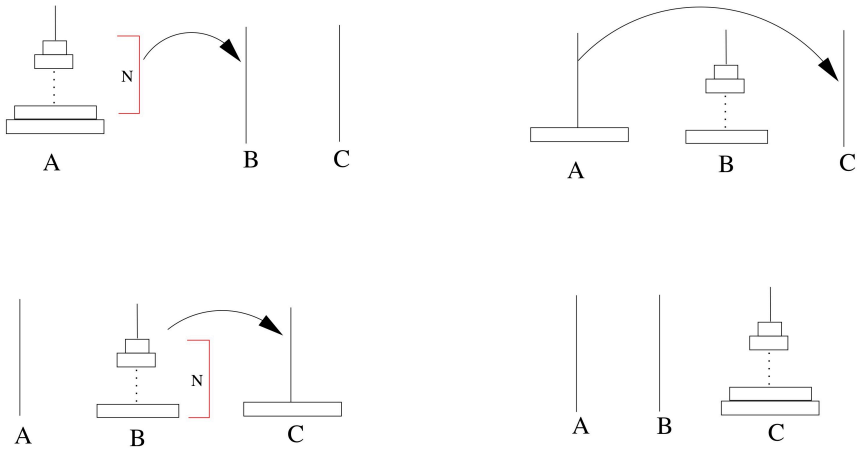



```
hanoi(n, A, B, C):
    if (n ==1)
        sposta da A a C;
    else{
        hanoi(n-1, A, C, B);
        sposta da A a C;
        hanoi(n-1, B, A, C);
    }
```

- se N == 1 —> 1 mossa
- se N > 1 —> ????

relazione di ricorrenza….

An example: Hanoi




```
hanoi(n, A, B, C):
    if (n ==1)
        sposta da A a C;
    else{
        hanoi(n-1, A, C, B);
        sposta da A a C;
        hanoi(n-1, B, A, C);
    }
```

- se N == 1 —> 1 mossa
- se N > 1 —> 2h(N-1) + 1

An example: Hanoi

- $h(1) = 1$
- $h(N) = 2h(N-1) + 1,\ N > 1$

An example: Hanoi

- $h(1) = 1$
- $h(N) = 2h(N-1) + 1,\ N > 1$

Provate a risolvere per
sostituzione

An example: Hanoi

- h(1) = 1
- h(N) = 2h(N-1) + 1, N > 1

$$h(N) = 2h(N-1) + 1 = 2(2h(N-2) + 1) + 1$$

An example: Hanoi

- h(1) = 1
- h(N) = 2h(N-1) + 1, N > 1

$$h(N) = 2h(N\text{-}1) + 1 = 2(2h(N\text{-}2) + 1) + 1$$
$$= 4h(N\text{-}2) + 3 = 4(2h(N\text{-}3) + 1\ ) + 3$$

An example: Hanoi

- $h(1) = 1$
- $h(N) = 2h(N-1) + 1,\ N > 1$

$$h(N) = 2h(N-1) + 1 = 2(2h(N-2) + 1) + 1$$
$$= 4h(N-2) + 3 = 4(2h(N-3) + 1\ ) + 3$$
$$= 8h(N-3) + 7 = \ldots.$$

An example: Hanoi

- $h(1) = 1$
- $h(N) = 2h(N-1) + 1,\ N > 1$

$$h(N) = 2h(N-1) + 1 = 2(2h(N-2) + 1) + 1$$
$$= 4h(N-2) + 3 = 4(2h(N-3) + 1\ ) + 3$$
$$= 8h(N-3) + 7 = \ldots.$$
$$= 2^{N-1}h(1) + 2^{N-1} - 1$$

An example: Hanoi

- $h(1) = 1$
- $h(N) = 2h(N-1) + 1,\ N > 1$

$$
\begin{aligned}
h(N) &= 2h(N-1) + 1 = 2(2h(N-2) + 1) + 1 \\
&= 4h(N-2) + 3 = 4(2h(N-3) + 1\ ) + 3 \\
&= 8h(N-3) + 7 = \ldots. \\
&= 2^{N-1}h(1) + 2^{N-1} - 1 \\
&= 2^{N} - 1
\end{aligned}
$$

An example: Hanoi

- $h(1) = 1$
- $h(N) = 2h(N-1)$

$$
\begin{aligned}
h(N) &= 2h(N-1) + 1 = \\
&= 4h(N-2) + 3 = \\
&= 8h(N-3) + 7 = \\
&= 2^{N-1}h(1) + 2^{N-1} - 1 \\
&= 2^{N} - 1
\end{aligned}
$$

Esponenziale!

580 miliardi di anni per 64 disks!

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$T_{CF}(1) = ????$

$T_{CF}(N) = ????$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$T_{CF}(1) = 1$

$T_{CF}(N) = ????$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$$T_{CF}(1) = 1$$
$$T_{CF}(N) = 2T_{CF}(N/2) + 1$$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$$T_{CF}(1) = 1$$
$$T_{CF}(N) = 2T_{CF}(N/2) + 1$$

$$T_{CF}(N) = 2T_{CF}(N/2) + 1 = 2(2T_{CF}(N/4) + 1) + 1$$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$$T_{CF}(1) = 1$$
$$T_{CF}(N) = 2T_{CF}(N/2) + 1$$

$$T_{CF}(N) = 2T_{CF}(N/2) + 1 = 2(2T_{CF}(N/4) + 1) + 1$$
$$= 4T_{CF}(N/4) + 3 = 2(4T_{CF}(N/8) + 3) + 1$$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$$T_{CF}(1) = 1$$
$$T_{CF}(N) = 2T_{CF}(N/2) + 1$$

$$T_{CF}(N) = 2T_{CF}(N/2) + 1 = 2(2T_{CF}(N/4) + 1) + 1$$
$$= 4T_{CF}(N/4) + 3 = 2(4T_{CF}(N/8) + 3) + 1$$
$$= 8T_{CF}(N/8) + 7 = \ldots.$$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$$T_{CF}(1) = 1$$
$$T_{CF}(N) = 2T_{CF}(N/2) + 1$$

$$\begin{aligned} T_{CF}(N) &= 2T_{CF}(N/2) + 1 = 2(2T_{CF}(N/4) + 1) + 1 \\ &= 4T_{CF}(N/4) + 3 = 2(4T_{CF}(N/8) + 3) + 1 \\ &= 8T_{CF}(N/8) + 7 = \ldots. \\ &= 2^k T_{CF}(N/2^k) + 2^k - 1 = [2^k == N \Leftrightarrow k == \log_2 N] = \end{aligned}$$

Esercizio

Cosa fa la procedura seguente? Qual è la sua complessità?

```
CosaFa(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = CosaFa(a, sx, cx);
   m2 = CosaFa(a, cx+1, dx);
   return m1 + m2;
}
```

$$T_{CF}(1) = 1$$
$$T_{CF}(N) = 2T_{CF}(N/2) + 1$$

$$\begin{aligned} T_{CF}(N) &= 2T_{CF}(N/2) + 1 = 2(2T_{CF}(N/4) + 1) + 1 \\ &= 4T_{CF}(N/4) + 3 = 2(4T_{CF}(N/8) + 3) + 1 \\ &= 8T_{CF}(N/8) + 7 = \ldots. \\ &= 2^k T_{CF}(N/2^k) + 2^k - 1 = [2^k == N \Leftrightarrow k == \log_2 N] = \\ &= NT_{CF}(1) + N - 1 = 2N - 1 \end{aligned}$$

Esercizio 2

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

Provate a risolvere per **sostituzione**

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

Esercizio 2

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

Provate a risolvere per **sostituzione**

$$T(N) = T(N/2) + 1 = T(N/4) + 1 + 1$$

Esercizio 2

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

Provate a risolvere per **sostituzione**

$$T(N) = T(N/2) + 1 = T(N/4) + 1 + 1$$
$$= T(N/4) + 2 =$$

Esercizio 2

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

Provate a risolvere per **sostituzione**

$$T(N) = T(N/2) + 1 = T(N/4) + 1 + 1$$
$$= T(N/4) + 2 =$$
$$= T(N/8) + 3 = \ldots.$$

Esercizio 2

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

Provate a risolvere per **sostituzione**

$$T(N) = T(N/2) + 1 = T(N/4) + 1 + 1$$
$$= T(N/4) + 2 =$$
$$= T(N/8) + 3 = \ldots.$$
$$= T(N/2^k) + k = [2^k == N \Leftrightarrow k == \log_2 N] =$$

Esercizio 2

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

Provate a risolvere per **sostituzione**

$$
\begin{aligned}
T(N) &= T(N/2) + 1 = T(N/4) + 1 + 1 \\
&= T(N/4) + 2 = \\
&= T(N/8) + 3 = \ldots. \\
&= T(N/2^k) + k = [2^k == N \Leftrightarrow k == \log_2 N] = \\
&= T(1) + \log_2 N
\end{aligned}
$$

Esercizio 2

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

Provate a risolvere per **sostituzione**

$$T(N) = T(N/2) + 1 = T(N/4) + 1 + 1$$
$$= T(N/4) + 2 =$$
$$= T(N/8) + 3 = \ldots.$$
$$= T(N/2^k) + k = [2^k == N \Leftrightarrow k == \log_2 N] =$$
$$= T(1) + \log_2 N$$

Algoritmo con **questa complessità**?

Esercizio 3

Algoritmo con **questa complessità**?

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

$= T(1) + \log_2 N$

Esercizio 3

Algoritmo con **questa complessità**?

$T(1) = 1$
$T(N) = T(N/2) + 1$ $= T(1) + \log_2 N$

```
AlgoLog(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = AlgoLog(a, sx, cx);
   return m1;
}
```

Esercizio 3

Algoritmo con **questa complessità**?

$T(1) = 1$
$T(N) = T(N/2) + 1$ $= T(1) + \log_2 N$

```
AlgoLog(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = AlgoLog(a, sx, cx);
   return m1;
}
```

Cosa fa????