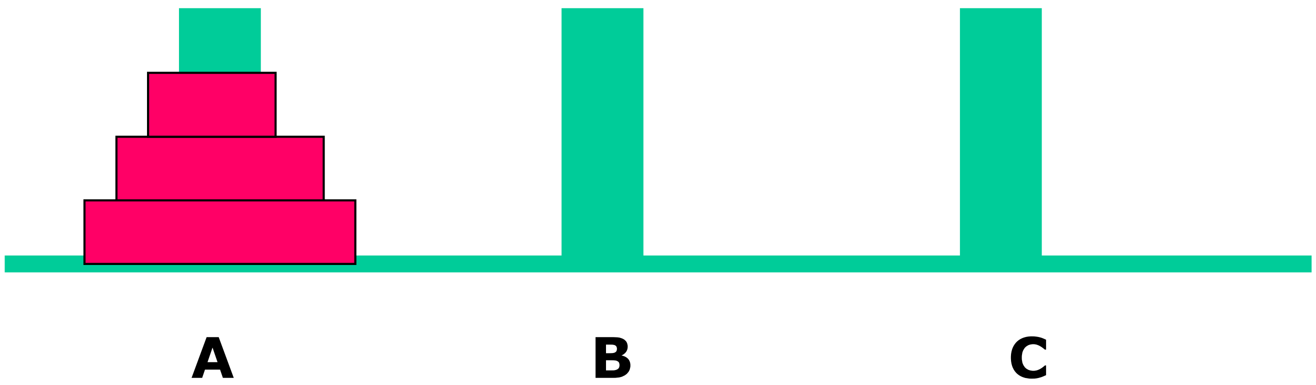
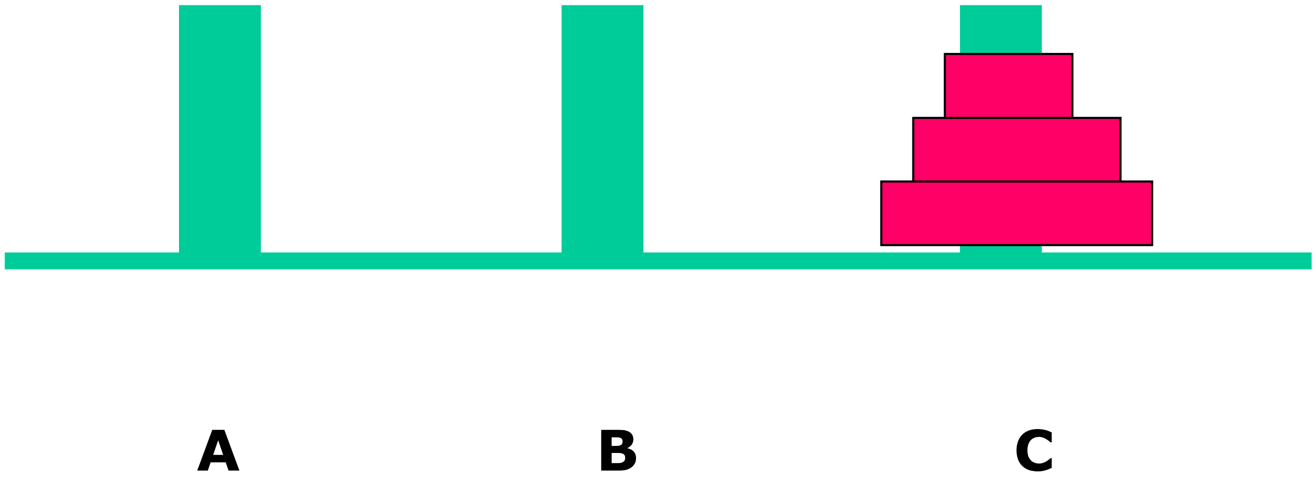
# Torre di Hanoi

- **3 paletti, 1 torre di n cerchi**
- **spostare la torre dal paletto sorgente A a quello destinatario C usando un paletto ausiliario B**
- **Un cerchio alla volta**
- **Mai un cerchio sopra uno più piccolo**



# Torre di Hanoi



# Torre di Hanoi

```
void trasferisci una torre di n cerchi da A a C
{

        Se n=1
                sposta il cerchio dal A a C;

        altrimenti

        {
                trasferisci la torre degli n-1 cerchi più piccoli da A a B
                   usando C come paletto ausiliario;
                sposta il cerchio più grande dal A a C;
                trasferisci la torre degli n-1 cerchi più piccoli da B a C
                   usando A come paletto ausiliario;
        }
}
```

# Torre di Hanoi

```
void hanoi(int n, pal A, pal B, pal C)
{
    if (n == 1)
        sposta(A, C);
    else {
        hanoi(n - 1, A, C, B);
        sposta(A, C);
        hanoi(n - 1, B, A, C);
    }
}
```

$$T(1) = a$$

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

```
hanoi(3, A, B, C)
    hanoi(2, A, C, B)
            hanoi(1, A, B, C)
                        sposta(A, C);
            sposta(A, B);
            hanoi(1, C, A, B)
                        sposta(C, B);
    sposta(A,C);
    hanoi(2, B, A, C)
            hanoi(1, B, C, A)
                        sposta(B, A);
            sposta(B, C);
            hanoi(1, A, B,C)
                        sposta(A,C);
```

hanoi(3, A, B, C)

hanoi(2, A, C, B)

hanoi(1, A, B, C)
sposta(A, C);

sposta(A, B);

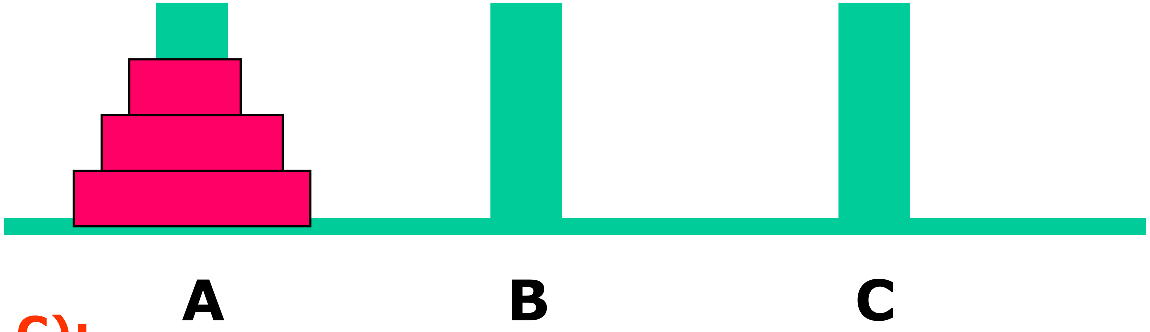
hanoi(1, C, A, B)
sposta(C, B);

sposta(A,C);

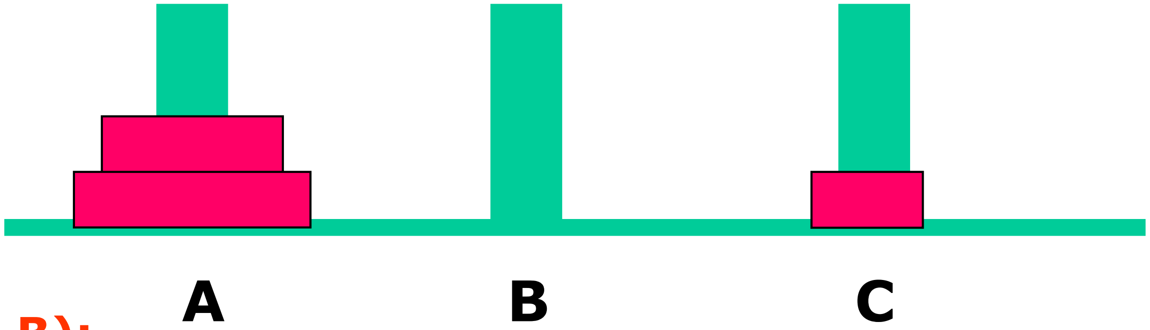
hanoi(2, B, A, C)
hanoi(1, B, C, A)
sposta(B, A);

sposta(B, C);

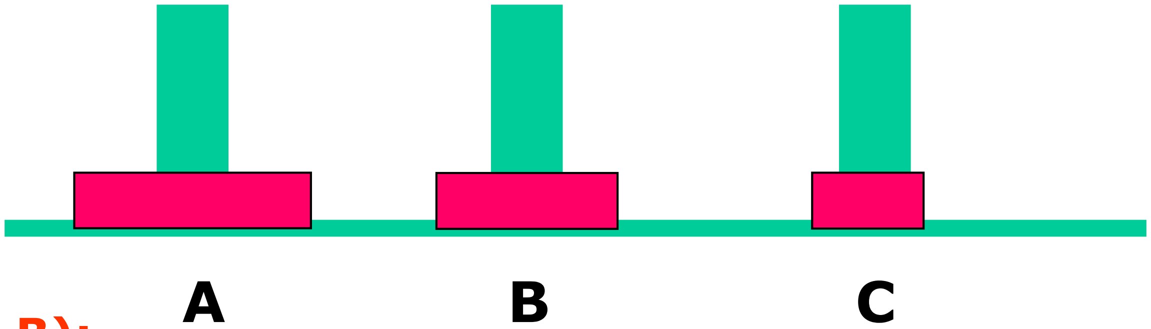
hanoi(1, A, B,C)
sposta(A,C);



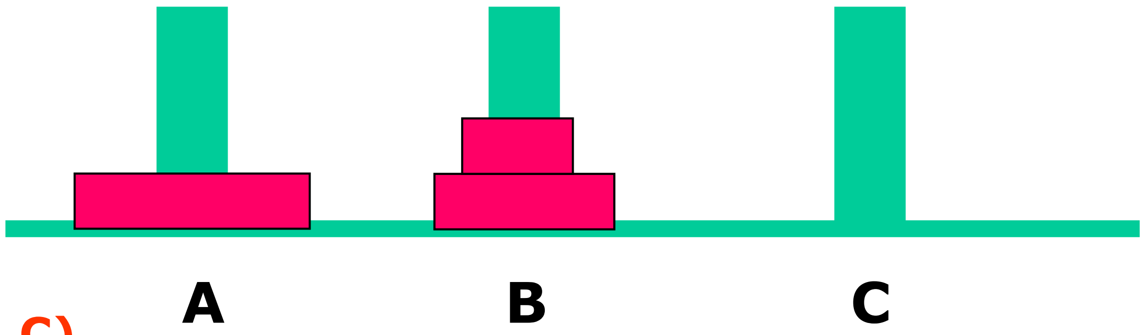

sposta(A, C);



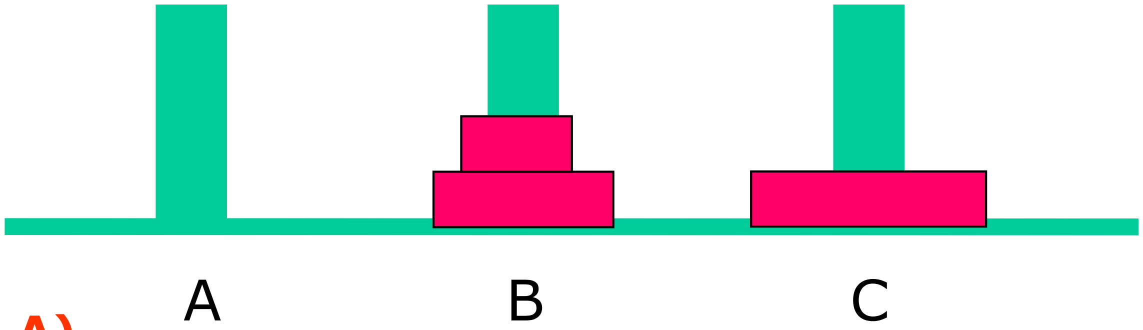

sposta(A, B);



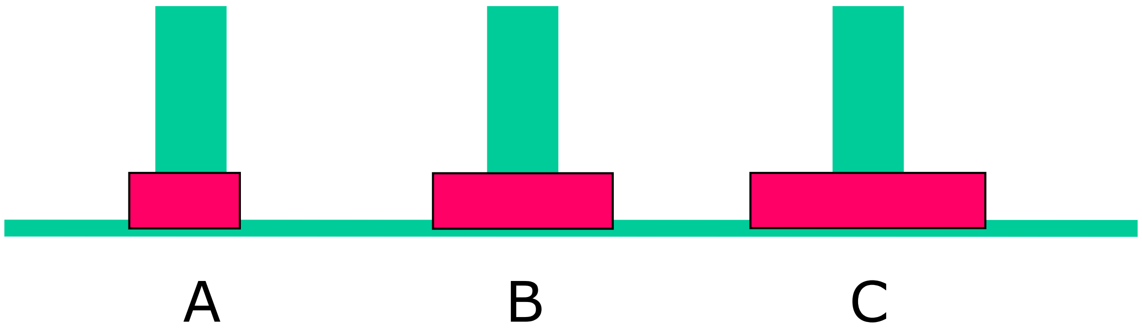

sposta(C, B);




**sposta(A,C)**



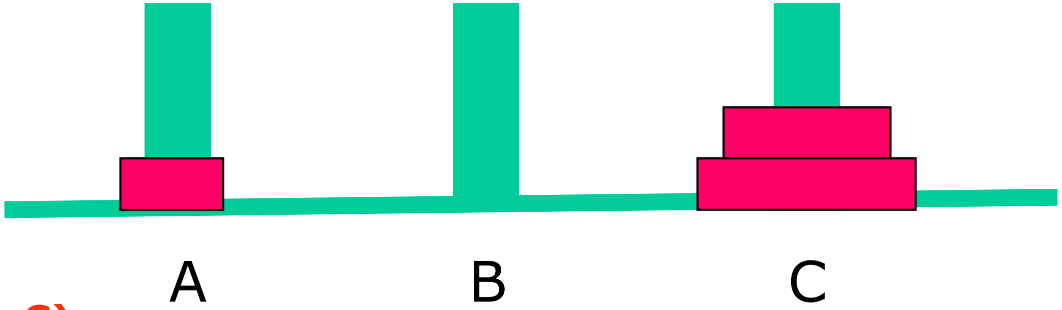

**sposta(B,A)**



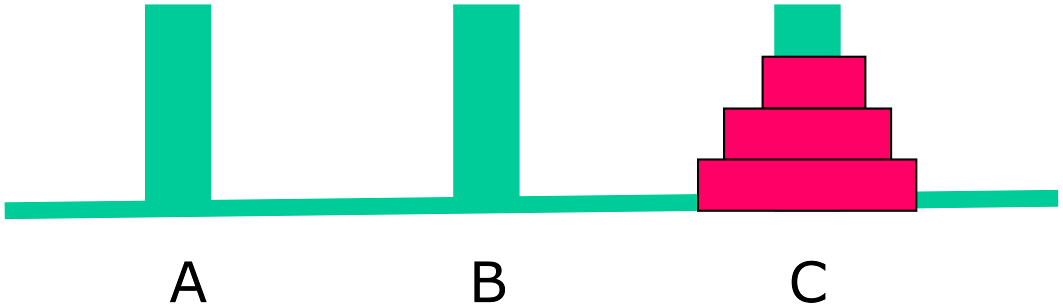

**sposta(B,C)**

```
hanoi(3, A, B, C)
    hanoi(2, A, C, B)
        hanoi(1, A, B, C)
            sposta(A, C);
        sposta(A, B);
        hanoi(1, C, A, B)
            sposta(C, B);
    sposta(A,C);
    hanoi(2, B, A, C)
        hanoi(1, B, C, A)
            sposta(B, A);
        sposta(B, C);
        hanoi(1, A, B,C)
            sposta(A,C);
```

```
hanoi(3, A, B, C)
    hanoi(2, A, C, B)
            hanoi(1, A, B, C)
                    sposta(A, C);
            sposta(A, B);
            hanoi(1, C, A, B)
                    sposta(C, B);
    sposta(A,C);
    hanoi(2, B, A, C)
            hanoi(1, B, C, A)
                    sposta(B, A);
            sposta(B, C);
            hanoi(1, A, B,C)
                    sposta(A,C);
```




**sposta(A,C)**



## soluzione

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

$$T(1) = a$$

$$T(2) = b + 2a$$

$$T(3) = b + 2b + 4a = 3b + 4a$$

$$T(4) = 7b + 8a$$

.

.

$$T(n) = (2^{(n-1)} - 1)b + 2^{(n-1)} a$$

**$T(n)$ è $O(2^n)$**

## Ricerca in un insieme

```
int RlinearSearch (int A [], int x, int m, int
i=0) {
    if (i == m) return 0;
    if (A[i] == x)  return 1;
    return RlinearSearch(A, x, m, i+1);
}
```

**O(n)**

$T(0) = a$

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

# Ricerca in un insieme

```
int binSearch (int A [],int x, int i=0, int j=m-1)
{
    if (i > j) return 0;
    int k=(i+j)/2;
    if (x == A[k]) return 1;
    if (x < A[k])
      return binSearch(A, x, i, k-1);
    else
      return binSearch(A, x, k+1, j);
 }
```

$$T(0) = a$$

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

## soluzione

$T(0) = a$
$T(n) = b + T(n/2)$

$T(0) = a$

$T(1) = b + a$

$T(2) = b + b + a$

$T(4) = b + b + b + a$

$T(8) = b + b + b + b + a$

.

$T(n) = (\log n + 1)\, b + a$

**T(n) è O(logn)**

# Ricerca in un insieme

```
int Search (int A [],int x, int i=0, int j=n-1) {
    if (i > j) return 0;
    int k=(i+j)/2;
    if (x == A[k])
      return 1;
    return Search(A, x, i, k-1) || Search(A, x, k+1, j);
  }
```

$$T(0) = a$$

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

# soluzione

$$T(0) = a$$
$$T(n) = b + 2T(n/2)$$

$$T(0) = a$$

$$T(1) = b + 2a$$

$$T(2) = b + 2b + 4a = 3b + 4a$$

$$T(4) = b + 6b + 8a = 7b + 8a$$

.

.

$$T(n) = (2n-1)b + 2n\,a$$

**T(n) è O(n)**

# Classificazione di alcune relazioni di ricorrenza

# Metodo divide et impera

```
void dividetimpera( S )
{
        if ( |S|<= m )
                <risolvi direttamente il problema>;
        else  {
                <dividi S in b sottoinsiemi S1.. Sb>;
                dividetimpera(S i1 );
                ...
                dividetimpera(S ia );
                <combina i risultati ottenuti>;
        }
}
```

# Metodo divide et impera

$T(0) = d$

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

**O ( log n )**

$T(0) = d$

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

**O ( n )**

$T(0) = d$

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

**O ( n log n )**

# Metodo divide et impera

$T(n) = d$ se $n = 1$

$T(n) = c + aT(\, n/b\, )$ se $n > 1$

$\mathbf{T(n) \in O(\, logn\, )}$ **se** $\mathbf{a = 1}$

$\mathbf{T(n) \in O(\, n^{\log_b a}\, )}$ **se** $\mathbf{a > 1}$

# Metodo divide et impera

$$T(n) = d \qquad \text{se } n \le m$$

$$T(n) = hn^k + aT(n/b) \qquad \text{se } n > m$$

$h > 0$

$T(n) \in O(n^k)$ se $a < b^k$

$T(n) \in O(n^k \log n)$ se $a = b^k$

$T(n) \in O(n^{\log_b a})$ se $a > b^k$

La complessità è calcolata prendendo come misura il numero di cifre che compongono il numero

**Ad esempio:**

- **L'addizione ha complessità $O(n)$**
- **la moltiplicazione che studiamo alle elementari ha complessità $O(n^2)$**

# Moltiplicazione veloce fra interi non negativi

n

| A |
|---|

n/2 n/2

| $A_s$ | $A_d$ |
|---|---|

$A = A_s 10^{n/2} + A_d$

$A=1325 = 13*10^2 + 25 \quad n=4$

| 13 | 25 |
|---|---|
| $A_s$ | $A_d$ |

$B = B_s 10^{n/2} + B_d$

$$AB= A_sB_s10^n + (A_sB_d+A_dB_s)\, 10^{n/2} + A_dB_d$$

$$(A_s+A_d)(B_s+B_d)= A_sB_d + A_dB_s + A_sB_s+A_dB_d$$

$$A_sB_d+A_dB_s =(A_s+A_d)(B_s+B_d)-A_sB_s- A_dB_d$$

$$AB=A_sB_s10^n + ((A_s+A_d)(B_s+B_d)-A_sB_s-A_dB_d)10^{n/2}+ A_dB_d$$

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

$$AB = A_sB_s10^n + ((A_s+A_d)(B_s+B_d) - A_sB_s - A_dB_d)10^{n/2} + A_dB_d$$

```
numero mult ( numero A, numero B, int n ) {
        if ( n=1 )  return A * B;           O(1)
        else  {
                As = parte sinistra di A ; Ad = parte destra di A ;  O(n/2)
                Bs = parte sinistra di B ; Bd = parte destra di B ;  O(n/2)
                int x1= As+Ad ; int x2= Bs+Bd ; O(n/2)  (somma di due numeri)
                int y1= mult (x1, x2, n/2);
                int y2= mult (As, Bs , n/2);
                int y3= mult (Ad, Bd , n/2);
                int z1= left_shift(y2,n);                  O(n)
                int z2= left_shift (y1-y2-y3, n/2);        O(n/2)
                return z1+z2+y3;
        }
}
```

**left_shift(h,n) scorre h di n posti a sinistra facendo entrare n 0 ($*10^n$)**

# Moltiplicazione veloce

$$T(1) = d$$

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

| | |
|---|---|
| $T(n) = d$ | se $n <= m$ |
| $T(n) = hn^k + aT(n/b)$ | se $n > m$ |

| | |
|---|---|
| **$T(n) \in O(n^k)$** | **se $a < b^k$** |
| **$T(n) \in O(n^k \log n)$** | **se $a = b^k$** |
| **$T(n) \in O(n^{\log_b a})$** | **se $a > b^k$** |

$$T(n) \in O(n^{\log_2 3})$$

$$T(n) \in O(n^{1.59})$$

## Relazioni lineari

$T(0) = d$
$T(n) = b + T(n-1)$

**$O(n)$**

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

**$O(n^2)$**

$T(0) = d$
$T(n) = b + 2T(n-1)$

**$O(2^n)$**

## Relazioni lineari

$T(0) = d$

$T(n) = bn^k + a_1T(n-1) + a_2T(n-2) + a_rT(n-r)$

**Polinomiale solo se**

- **esiste al più un solo $a_i = 1$ e**
- **gli altri $a_i$ sono tutti 0 (c'è una sola chiamata ricorsiva).**

**Negli altri casi sempre esponenziale.**

## Soluzione di una classe di relazioni lineari

$$T(0) = d$$

$$T(n) = bn^{k} + T(n-1)$$

$b > 0$

$$T(n) \in O(n^{k+1})$$