Calcul de complexité

Journées Irem 2015

Guillaume CONNAN

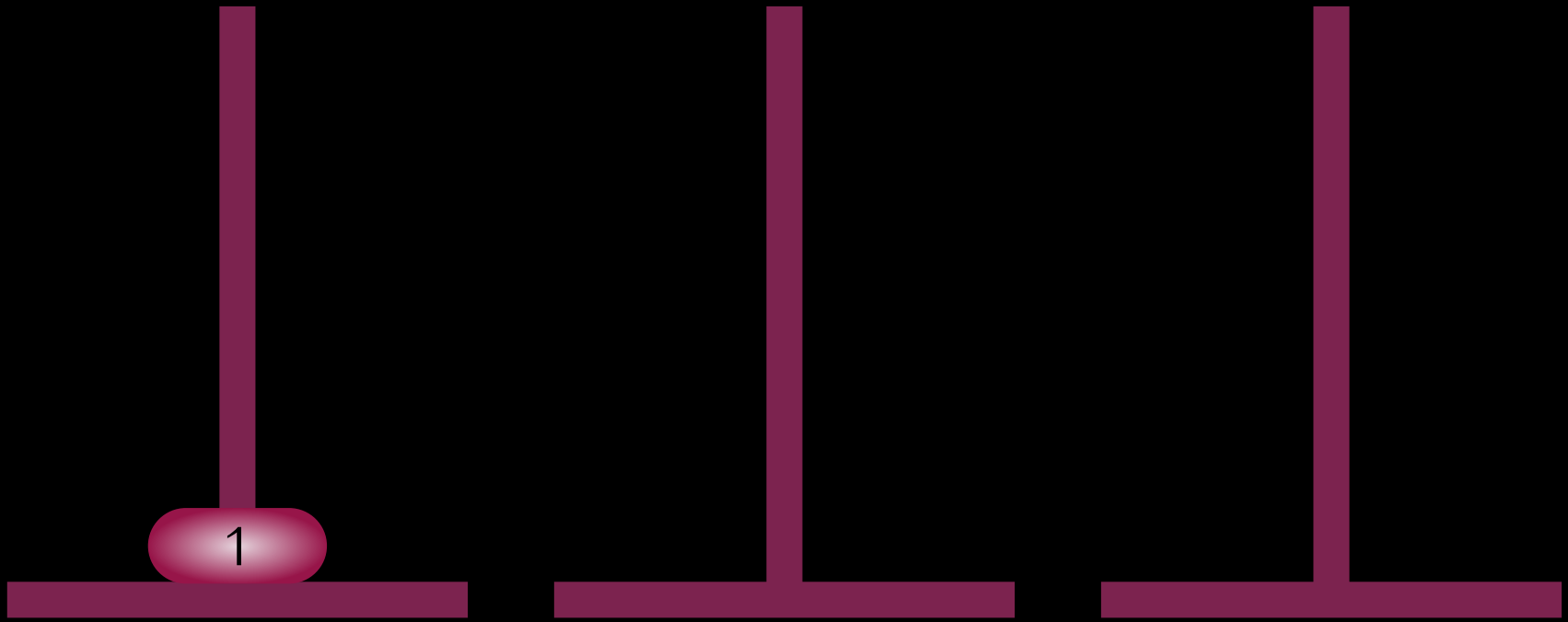
8 et 9 avril 2015

Irem de Nantes

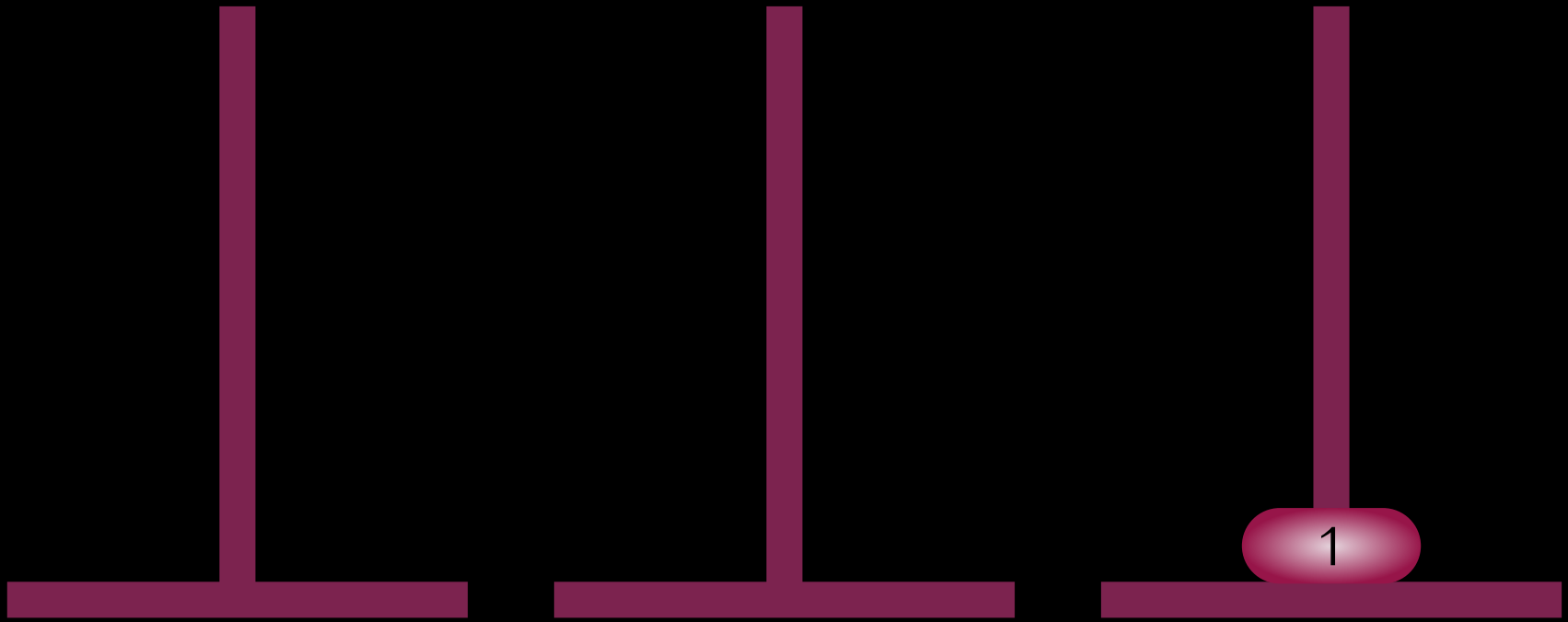
Sommaire

Sommaire

Tours de Hanoi – 1 Disque

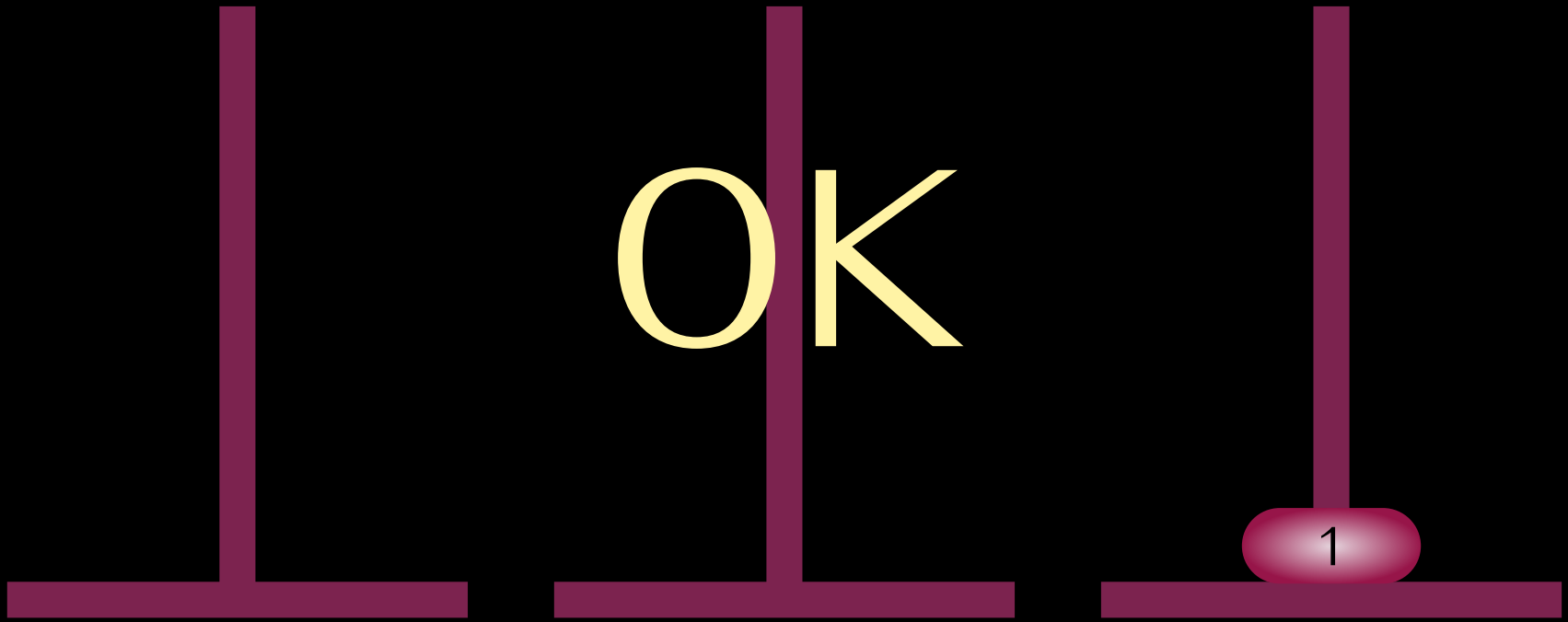


Tours de Hanoi – 1 Disque

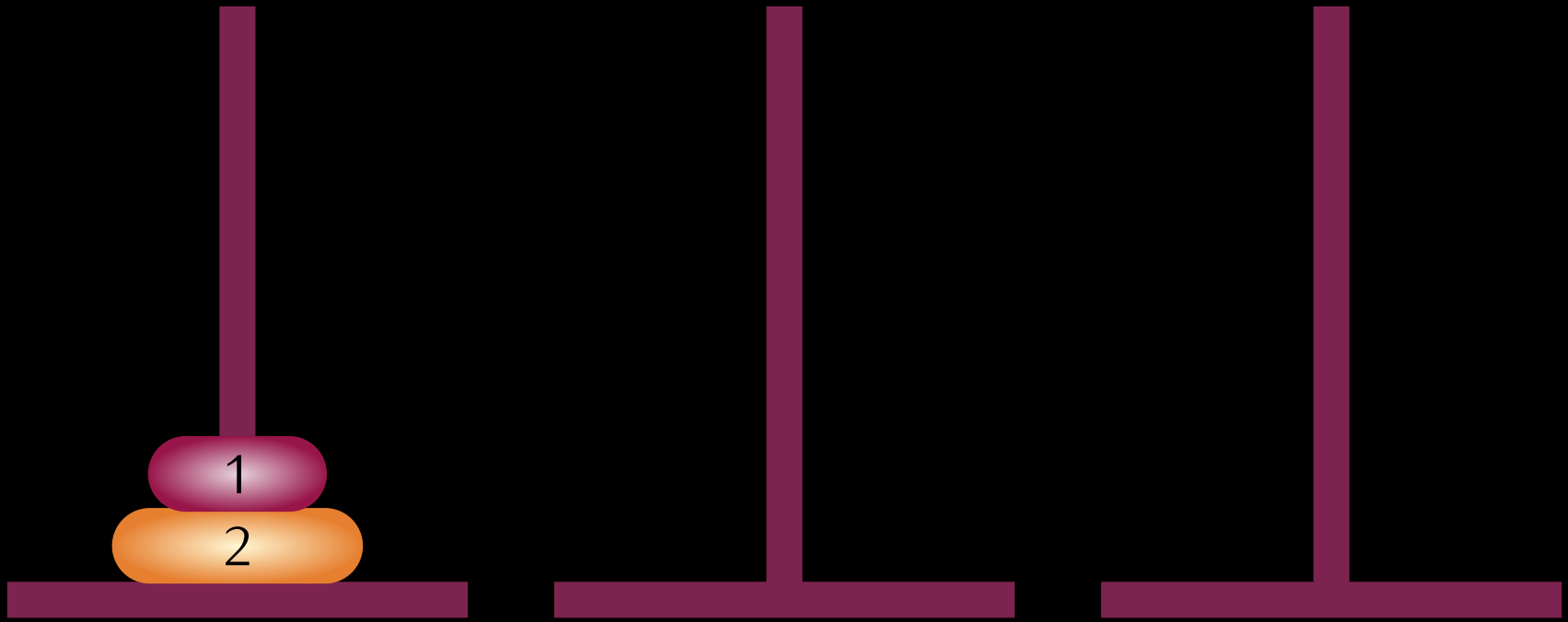


On déplace le disque de la tige 1 vers la tige 3.

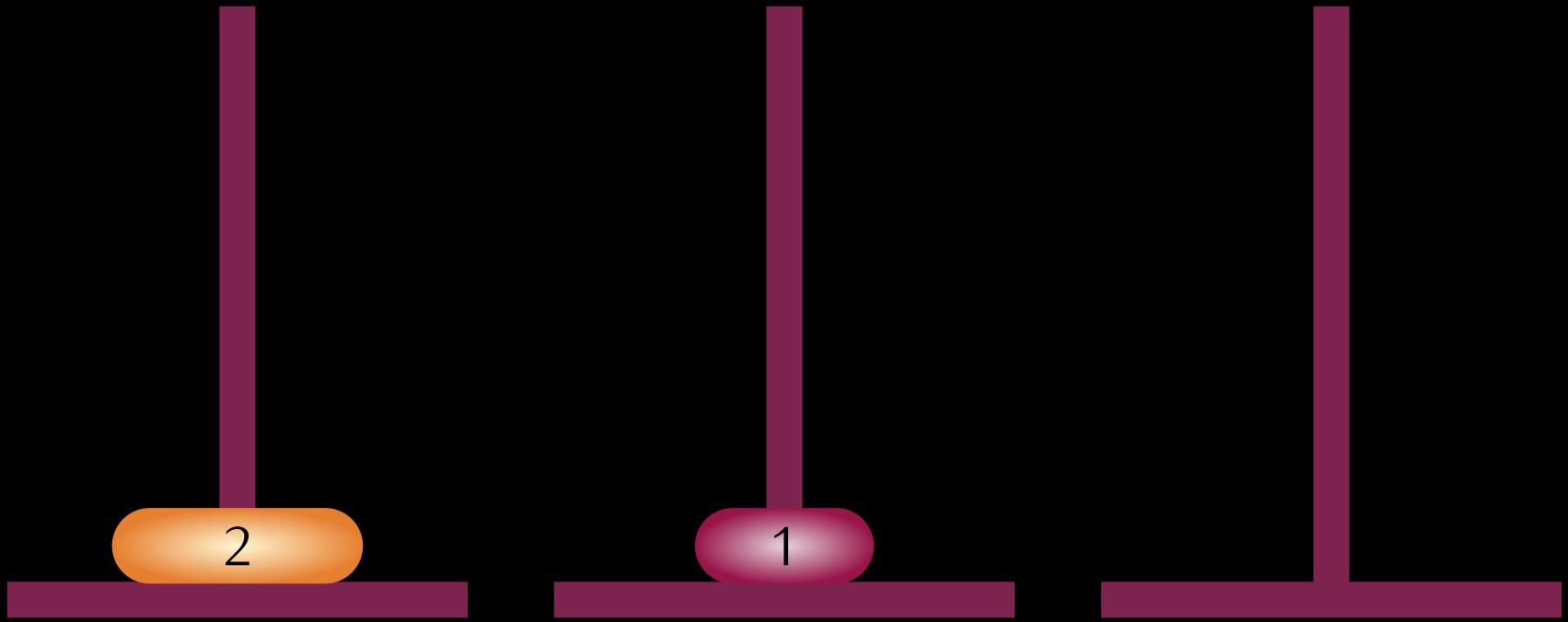
Tours de Hanoi – 1 Disque



Tours de Hanoi – 2 Disques

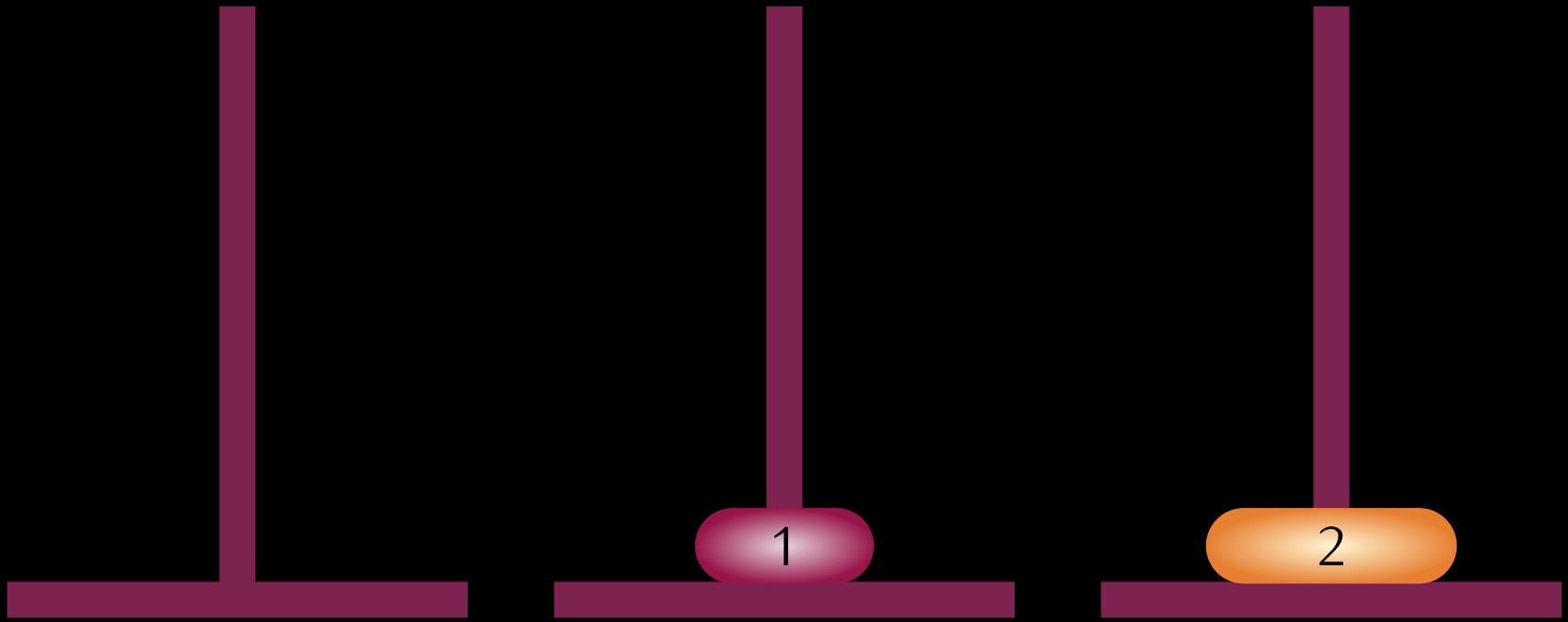


Tours de Hanoi – 2 Disques



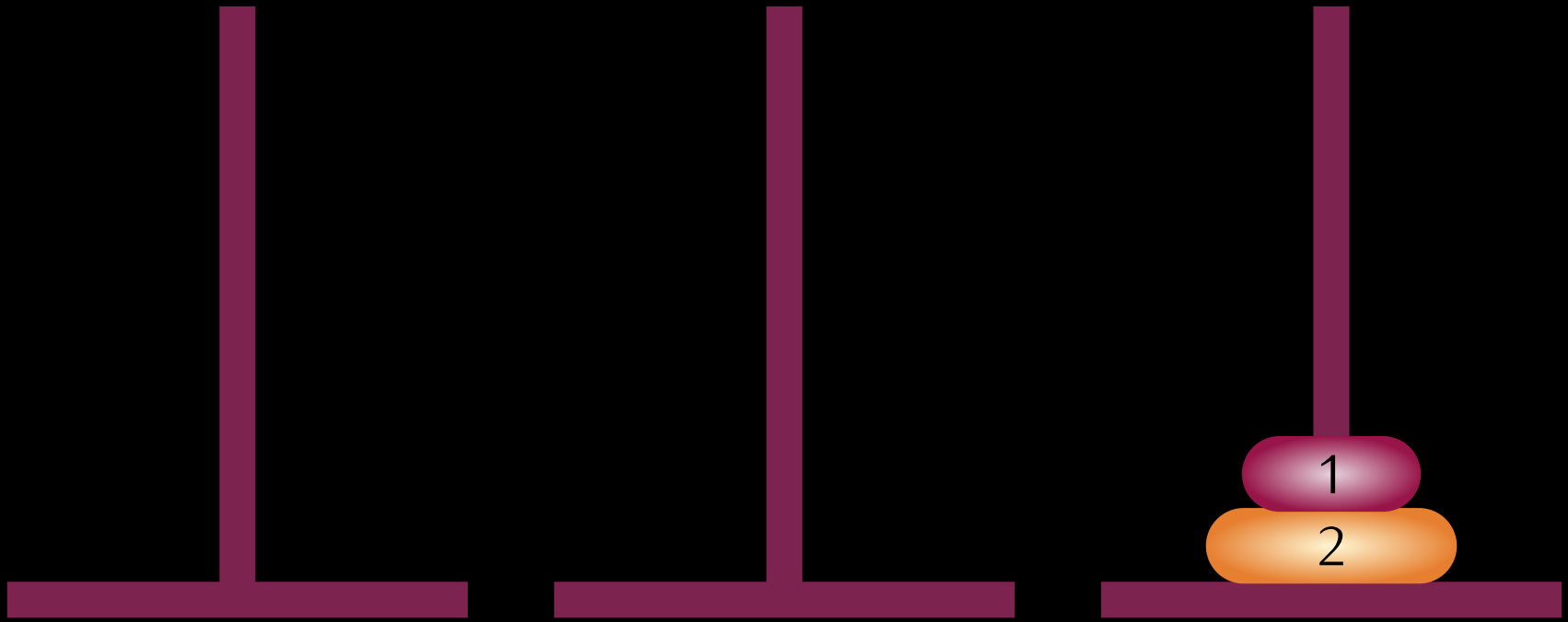
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 2 Disques



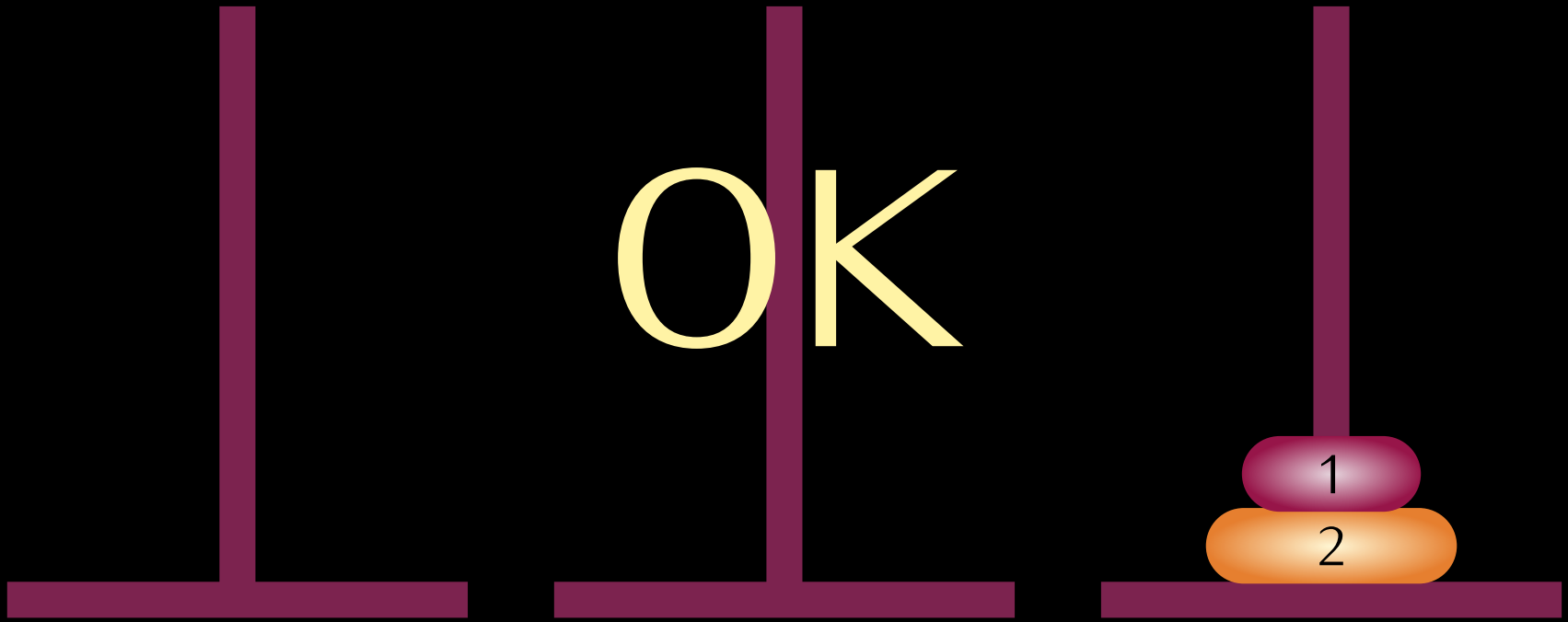
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 2 Disques

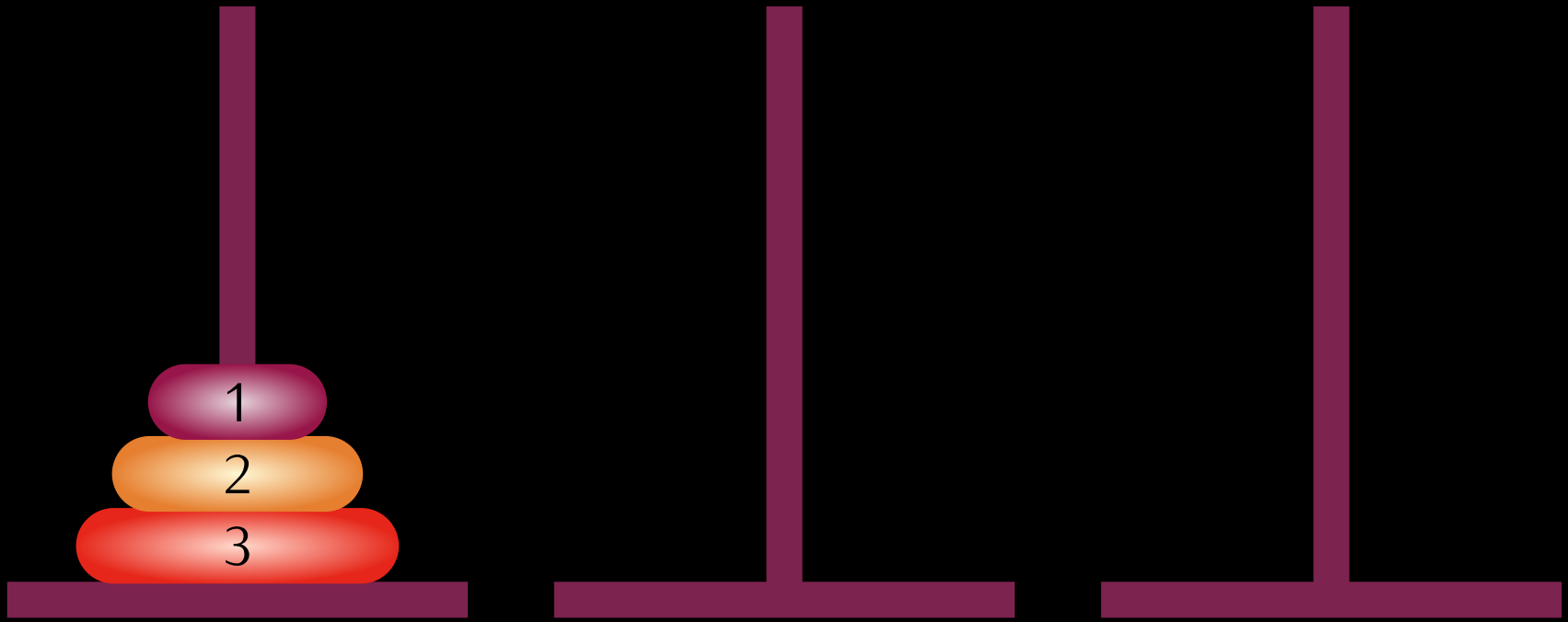


On déplace le disque de la tige 2 vers la tige 3.

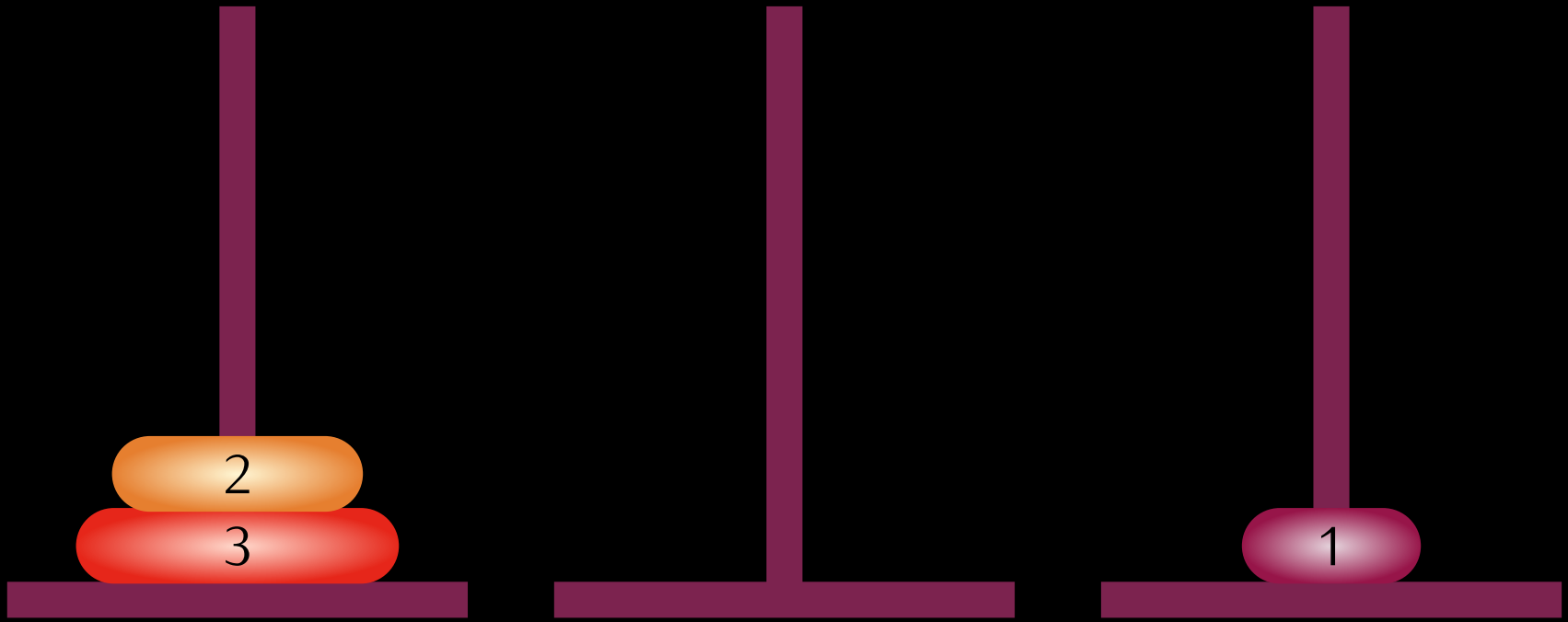
Tours de Hanoi – 2 Disques



Tours de Hanoi – 3 Disques

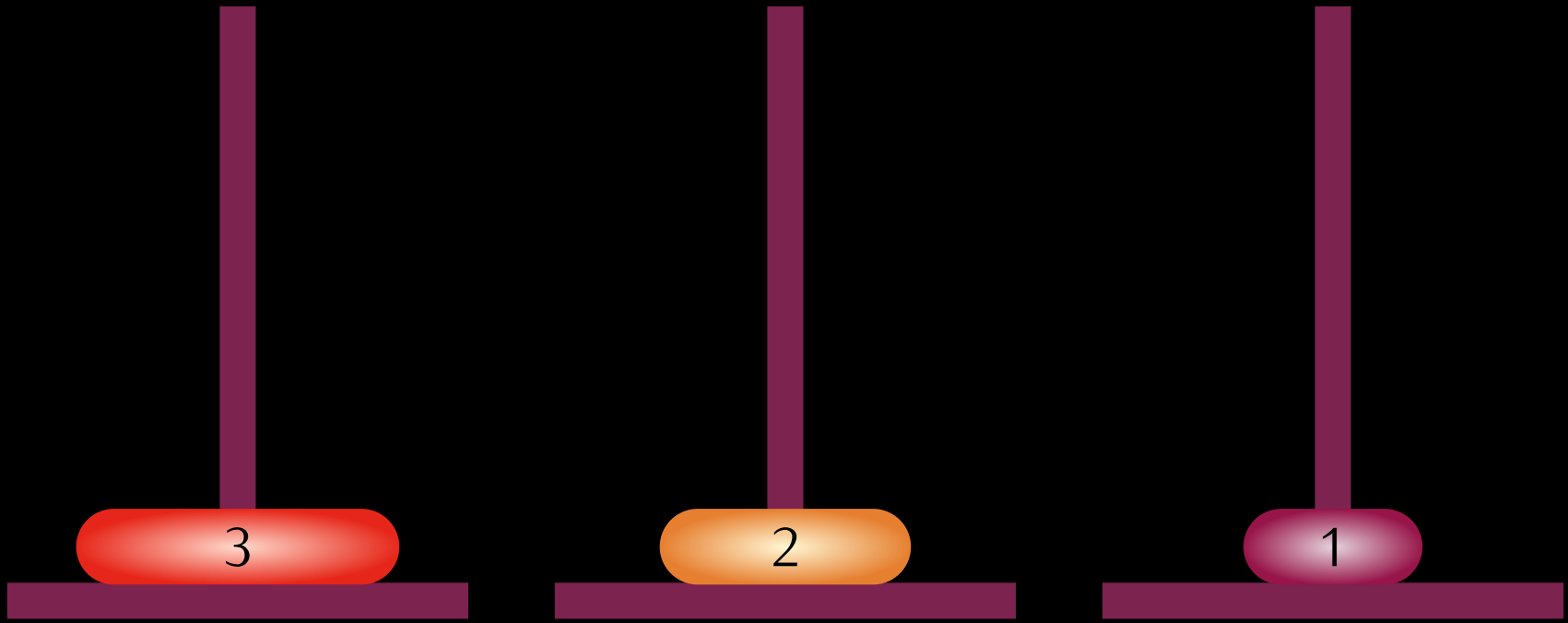


Tours de Hanoi – 3 Disques



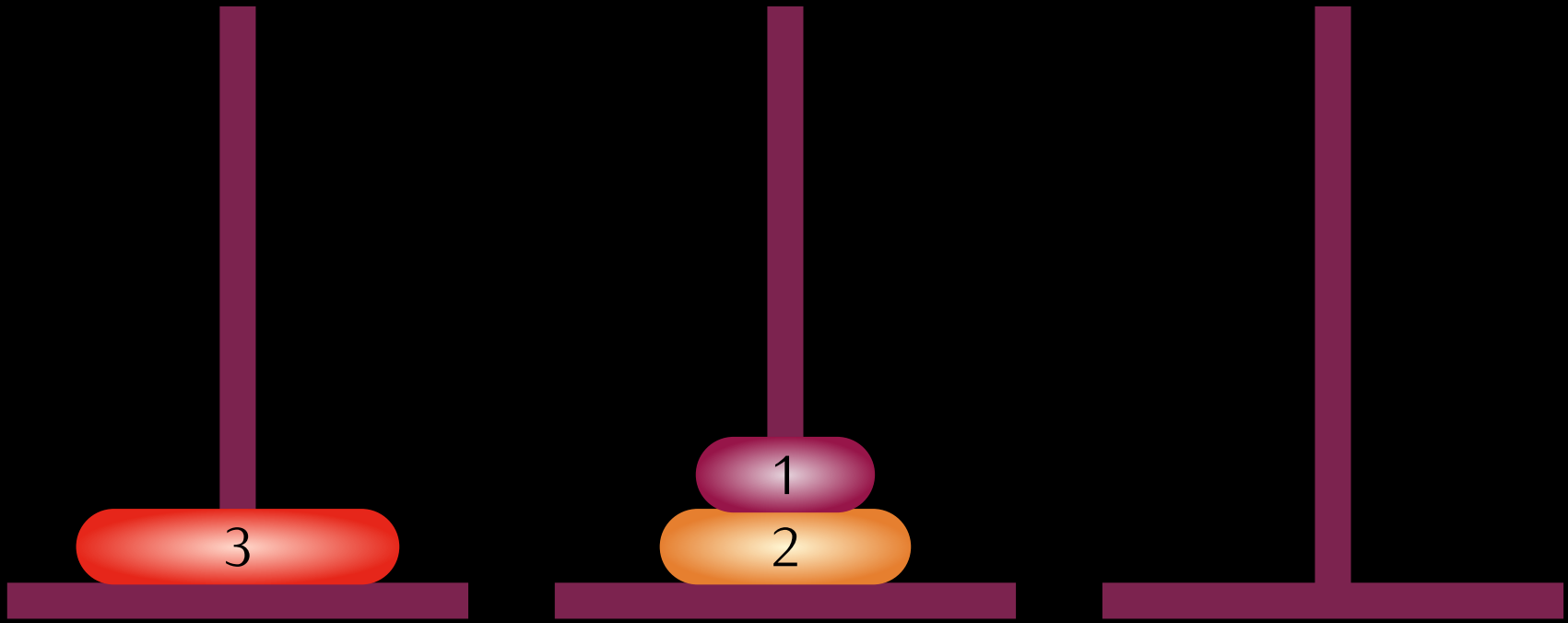
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 3 Disques



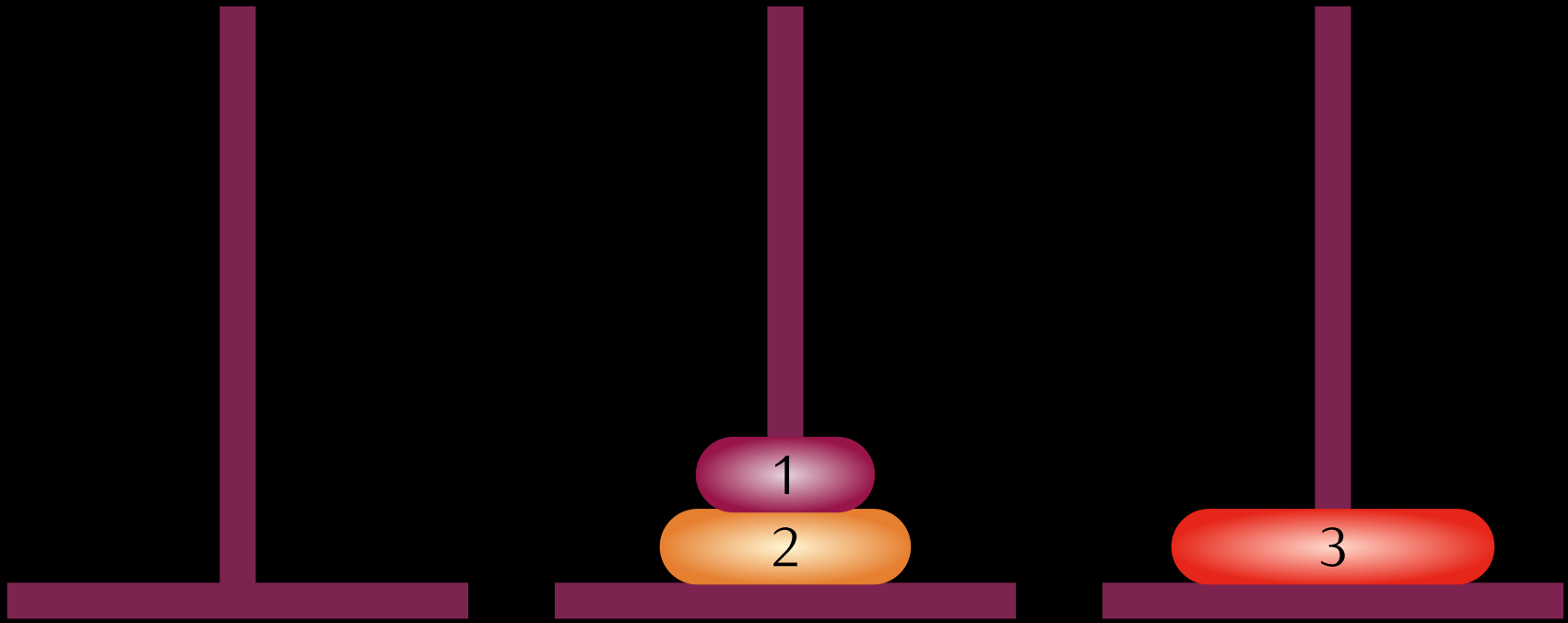
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 3 Disques



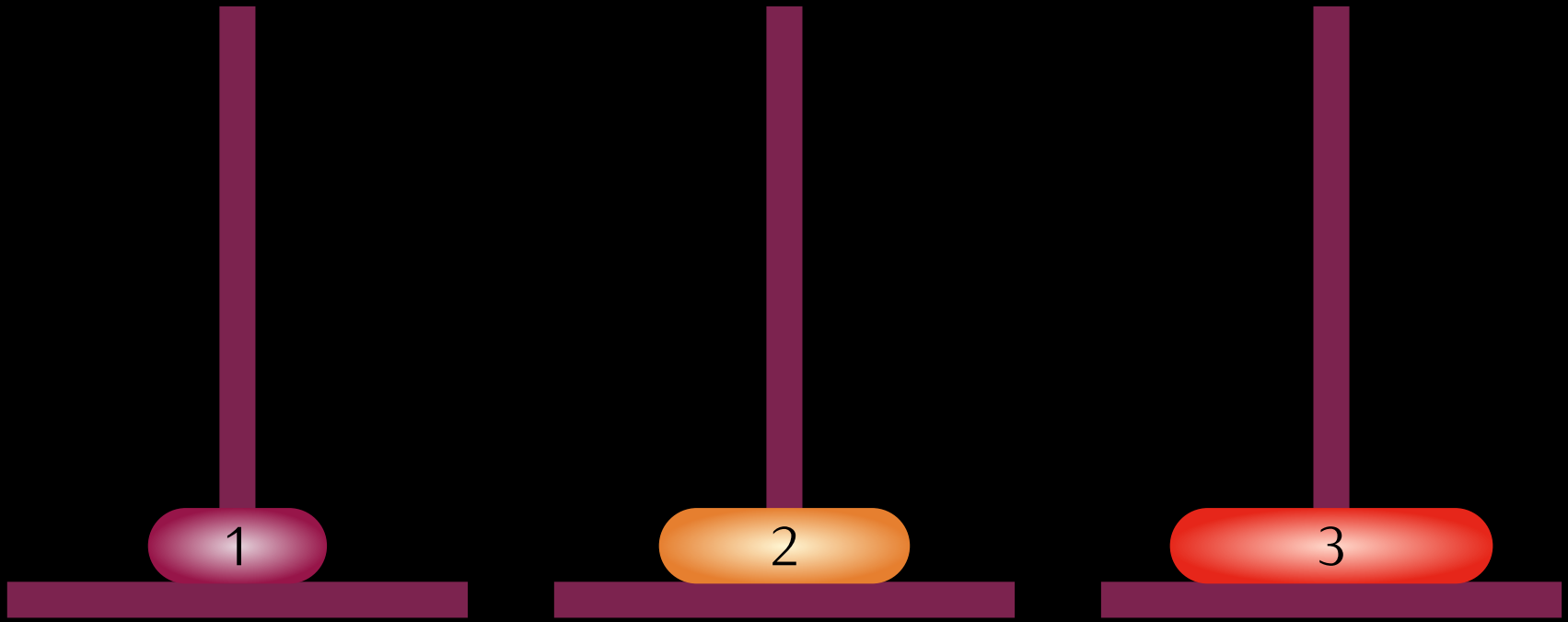
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 3 Disques



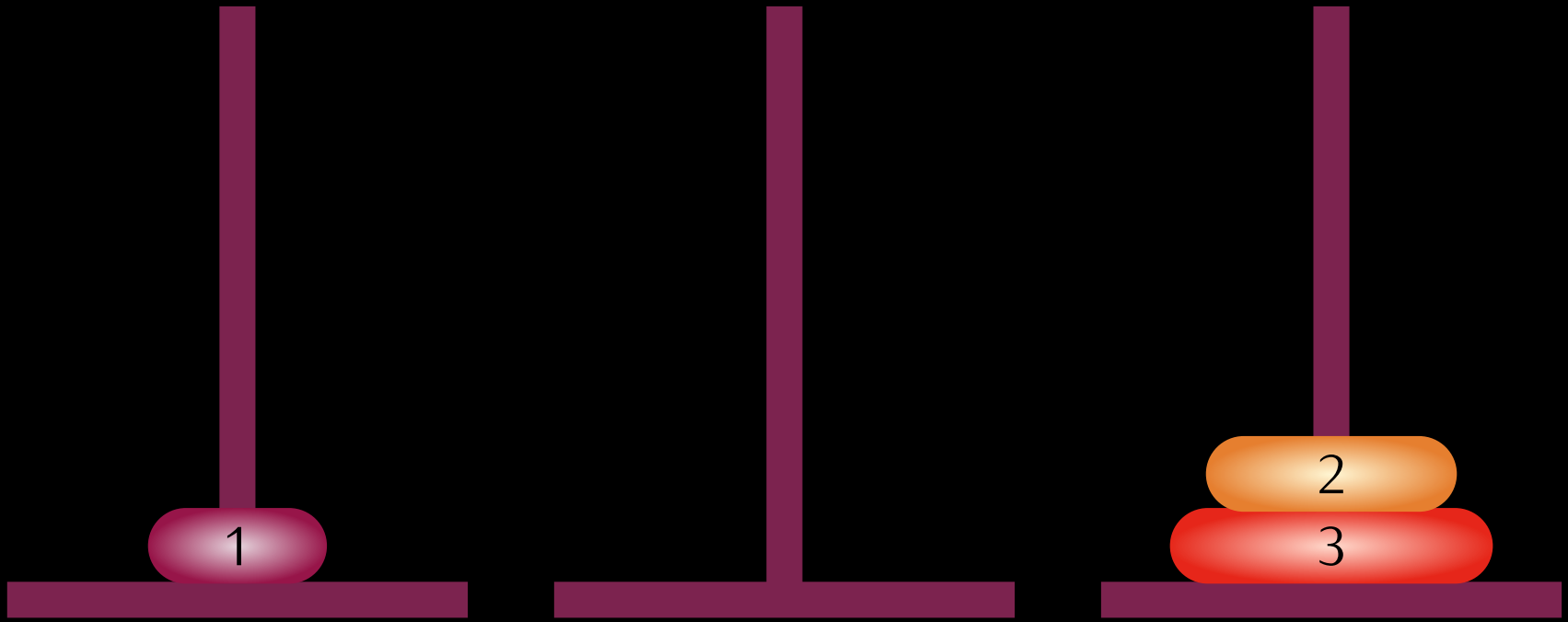
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 3 Disques



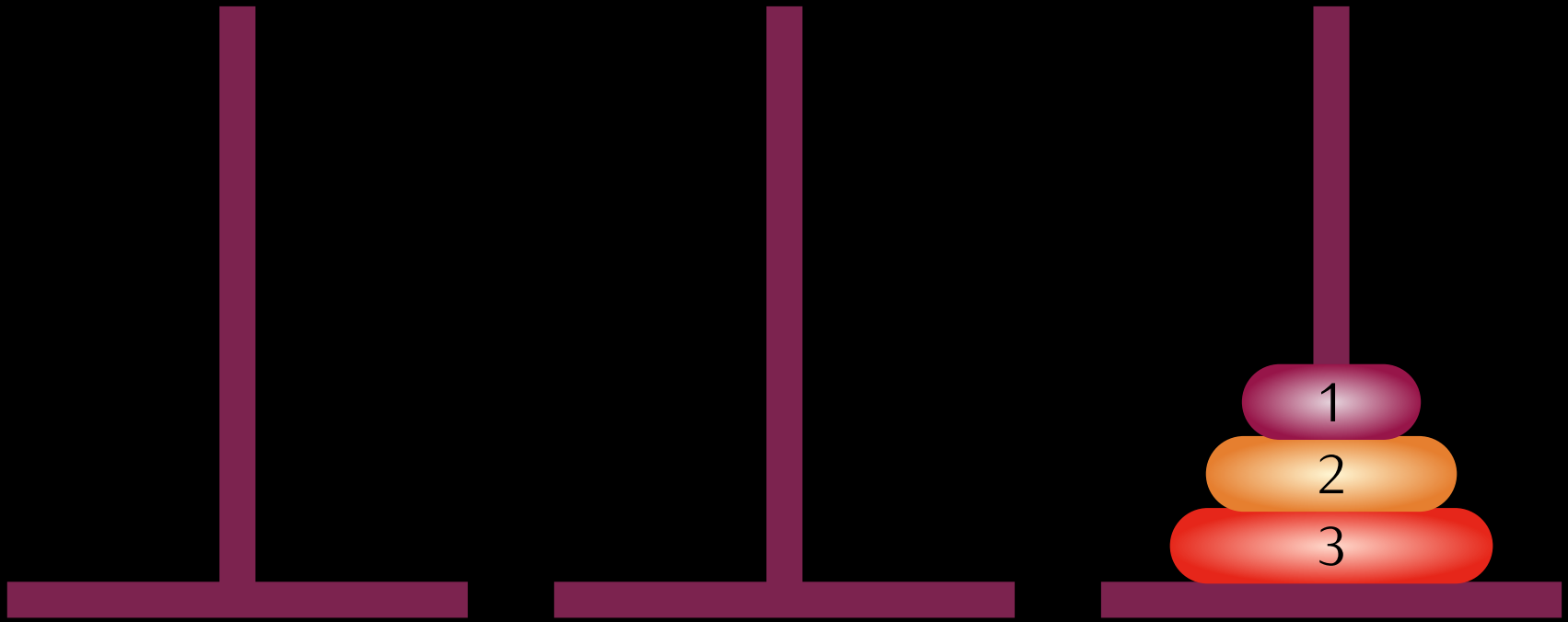
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 3 Disques



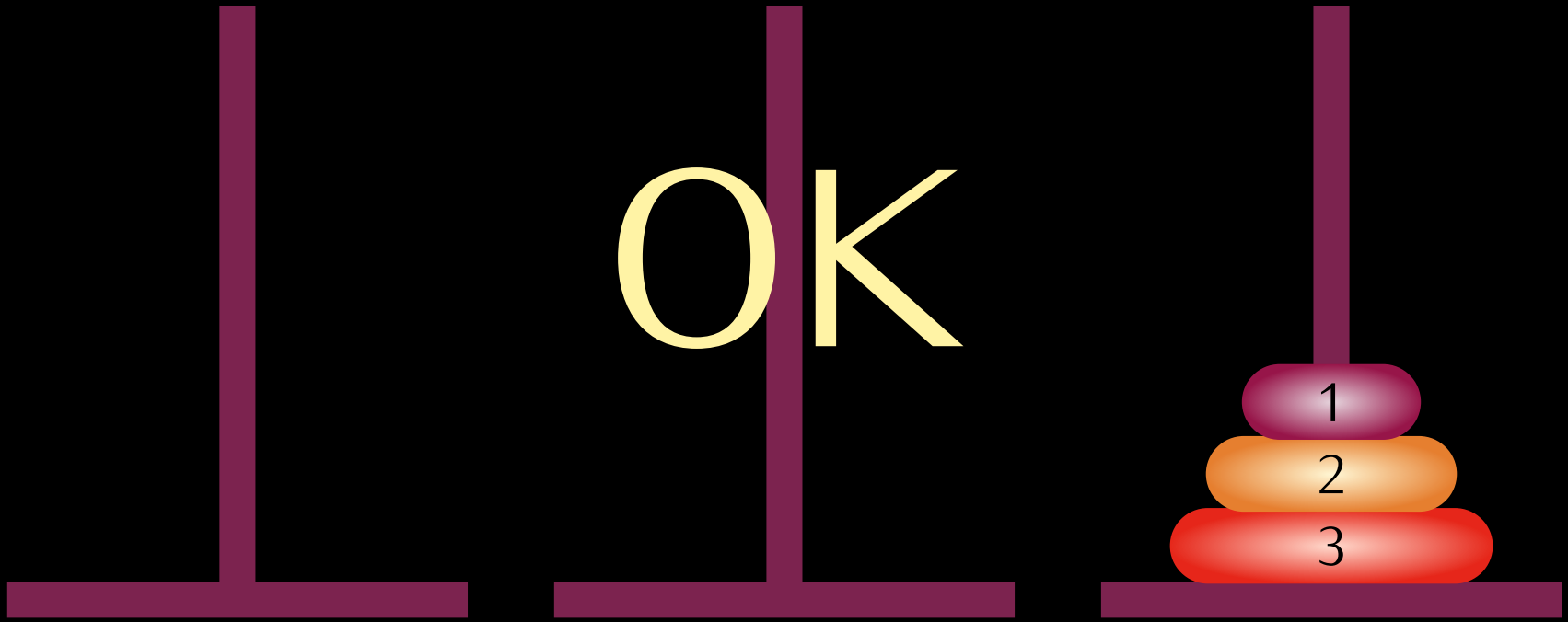
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 3 Disques

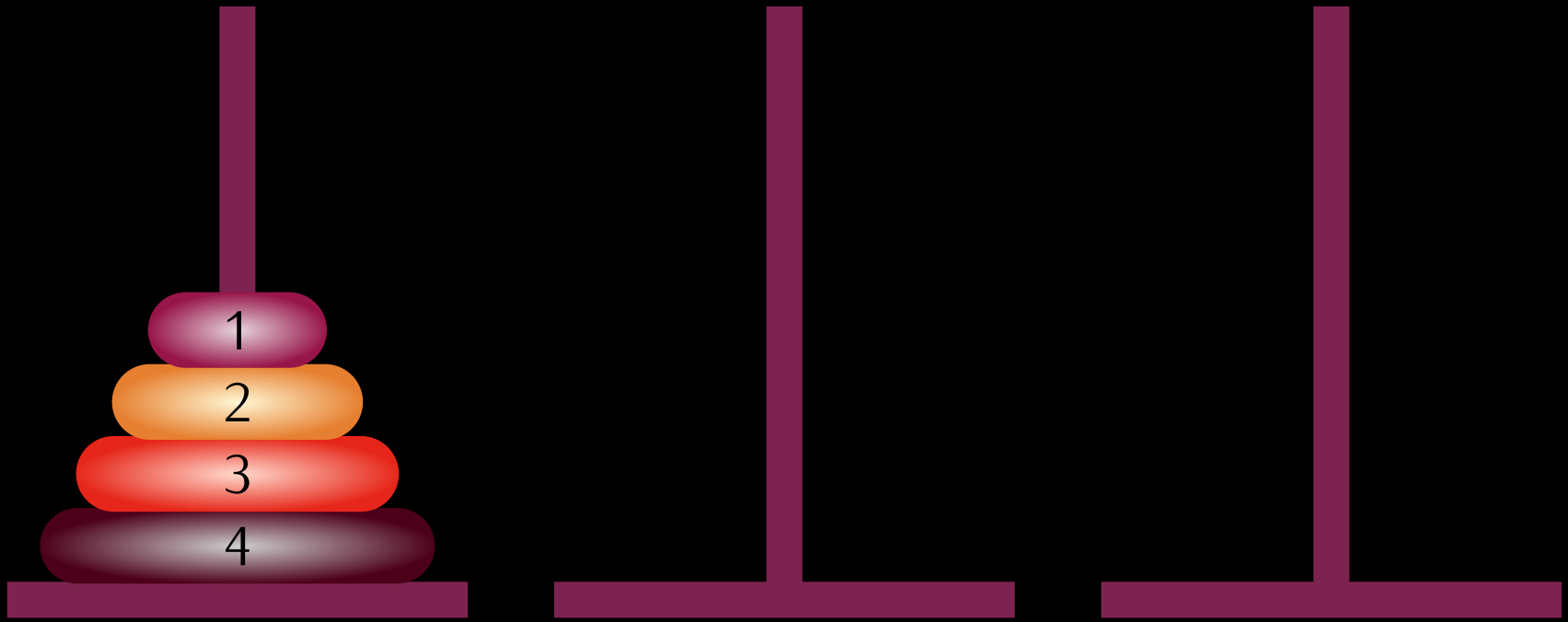


On déplace le disque de la tige 1 vers la tige 3.

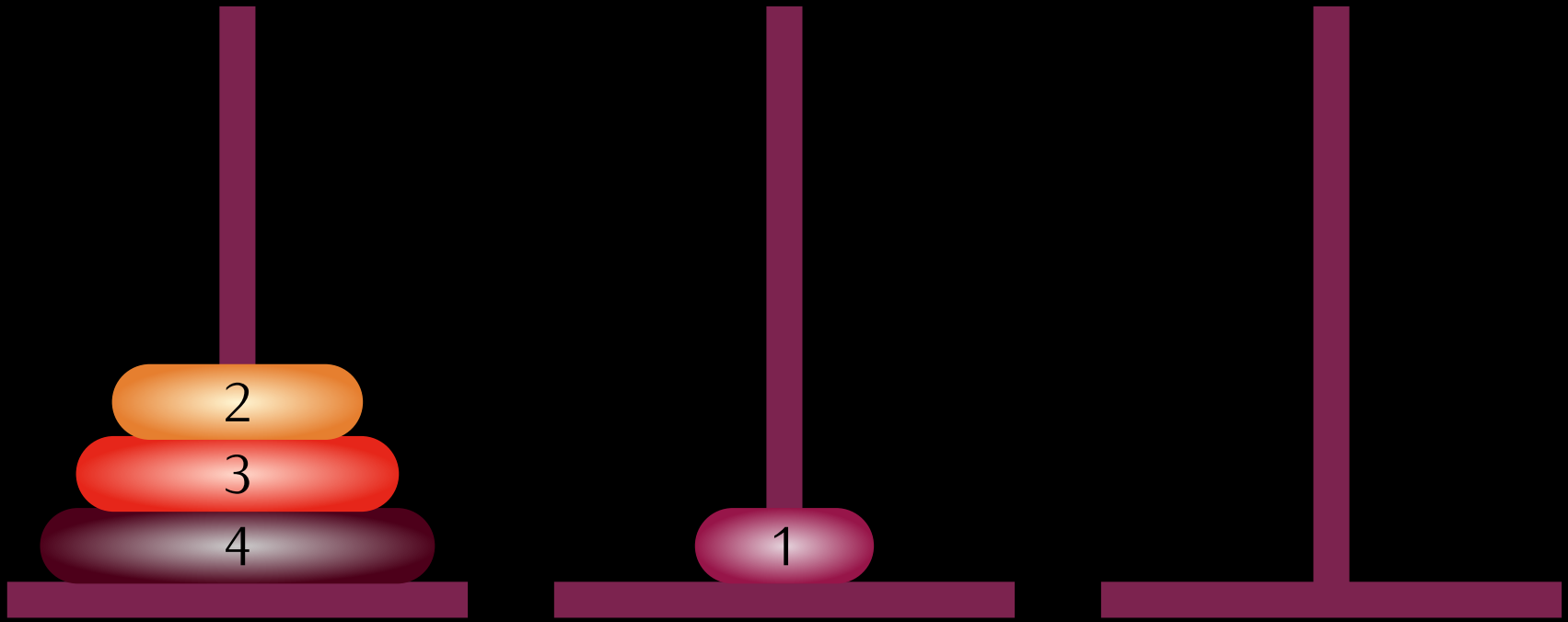
Tours de Hanoi – 3 Disques



Tours de Hanoi – 4 Disques

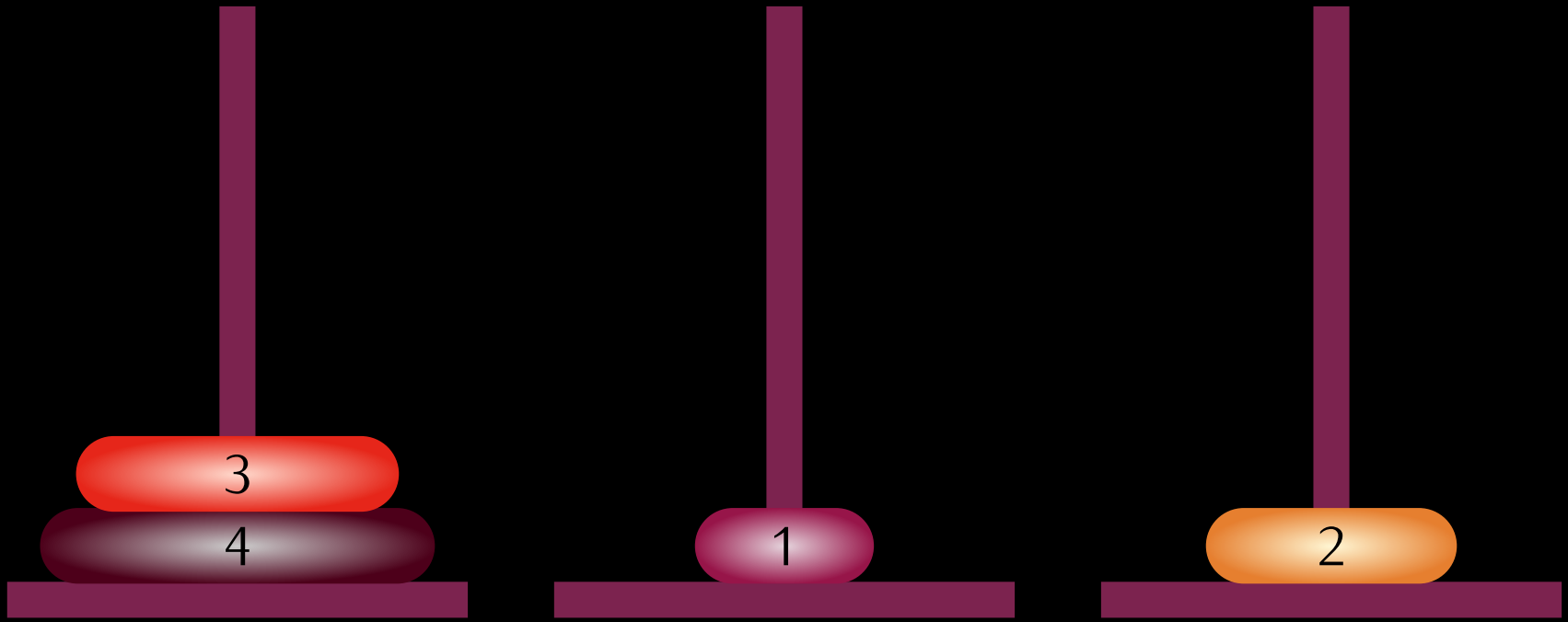


Tours de Hanoi – 4 Disques



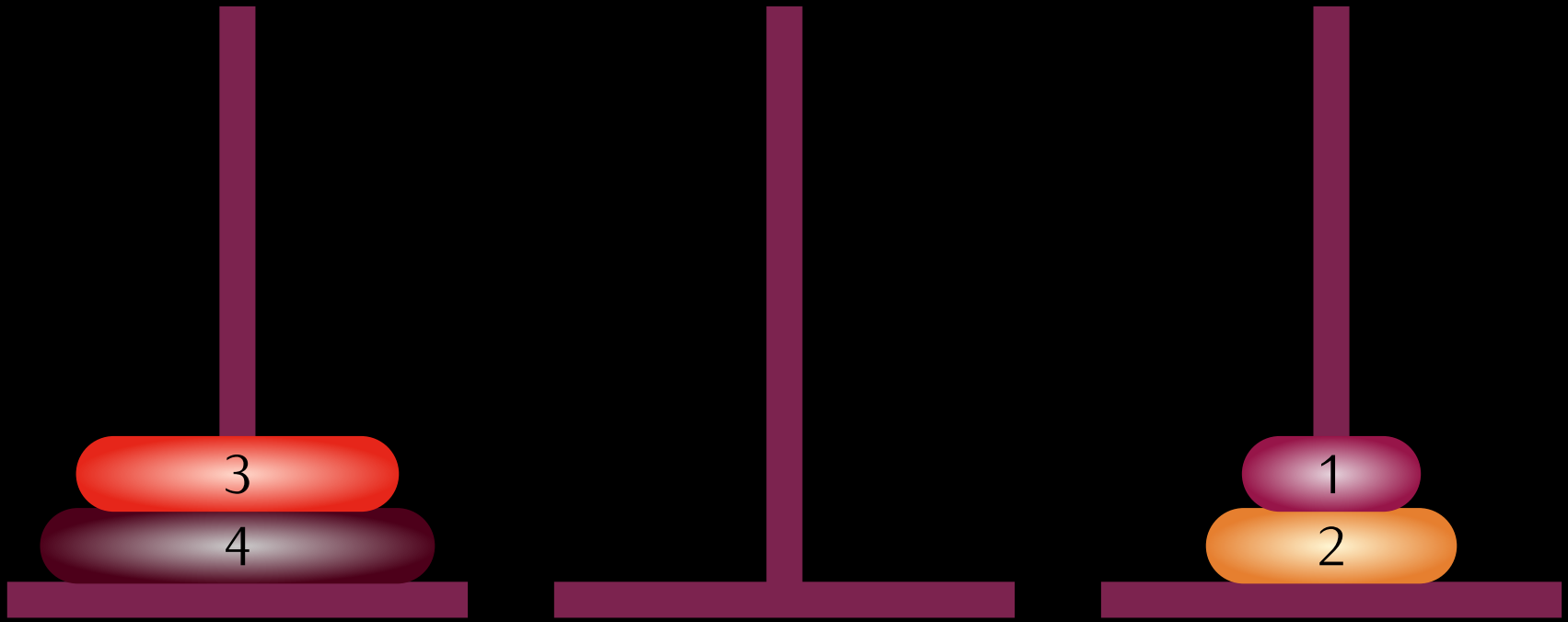
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



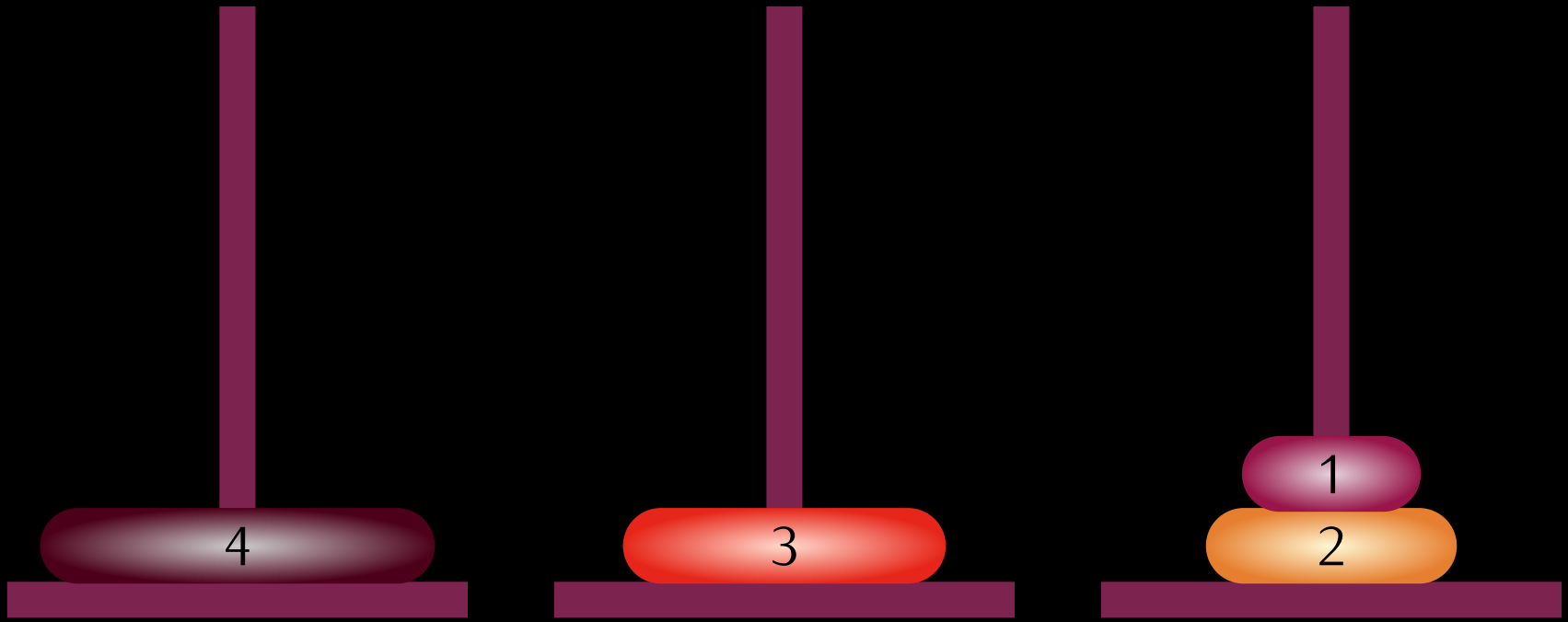
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



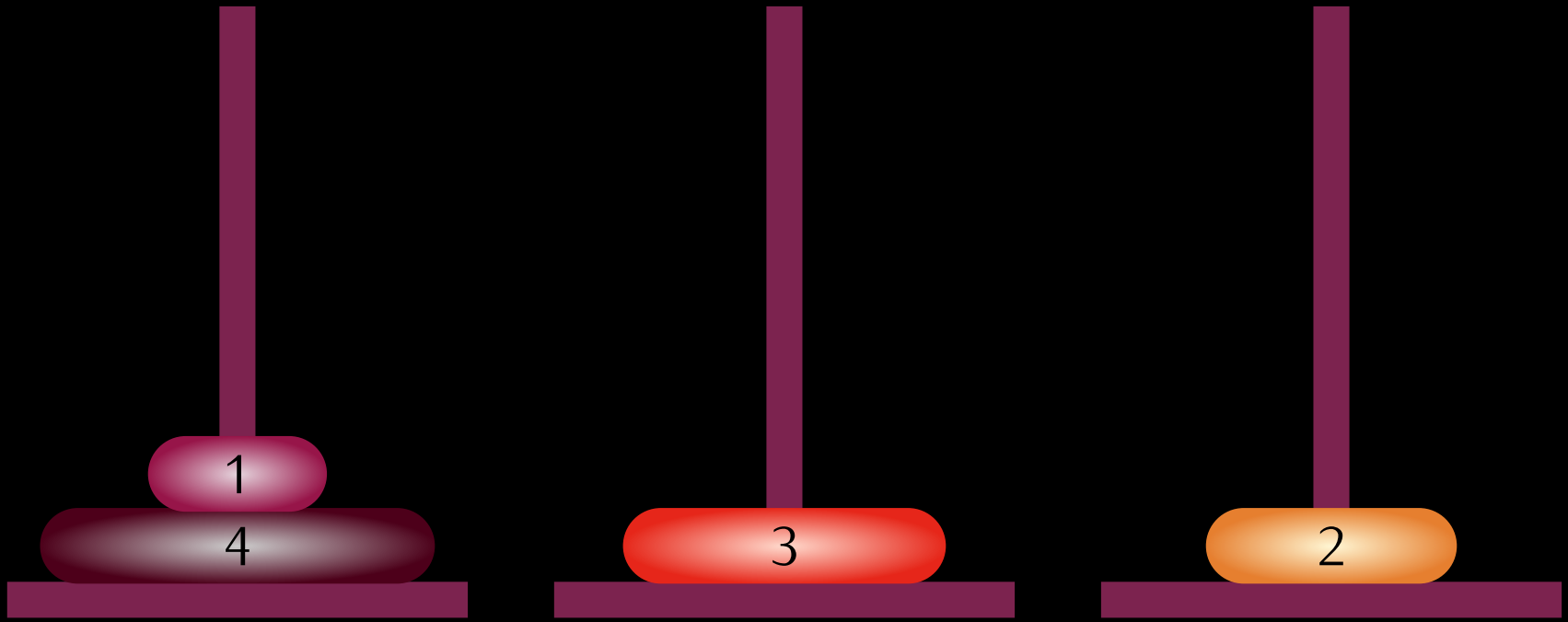
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



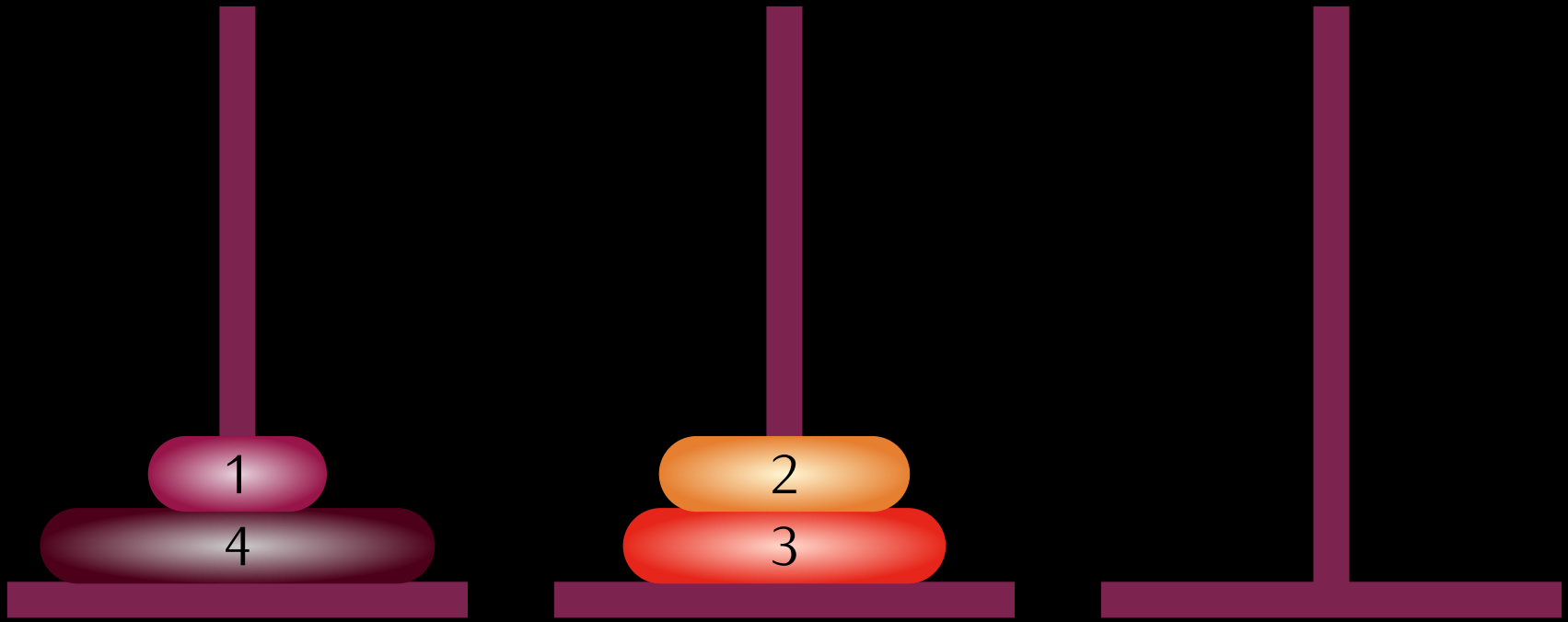
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



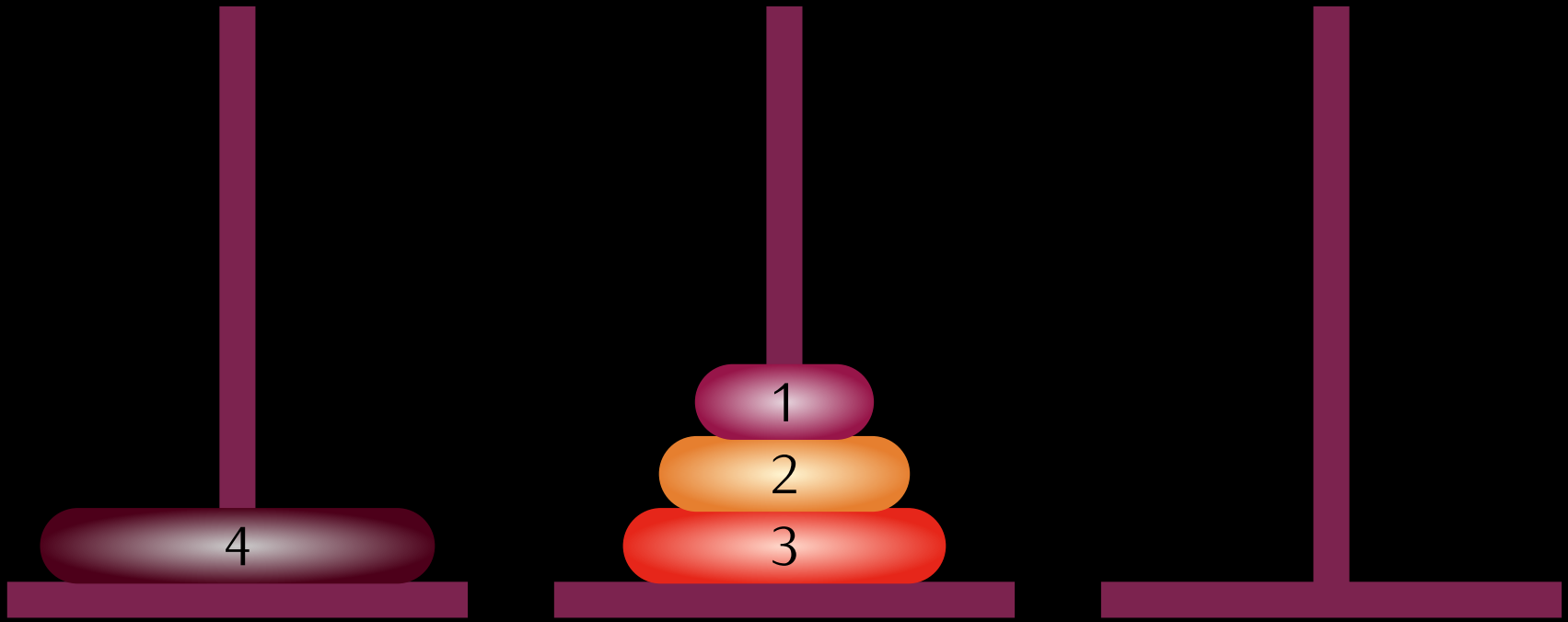
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 4 Disques



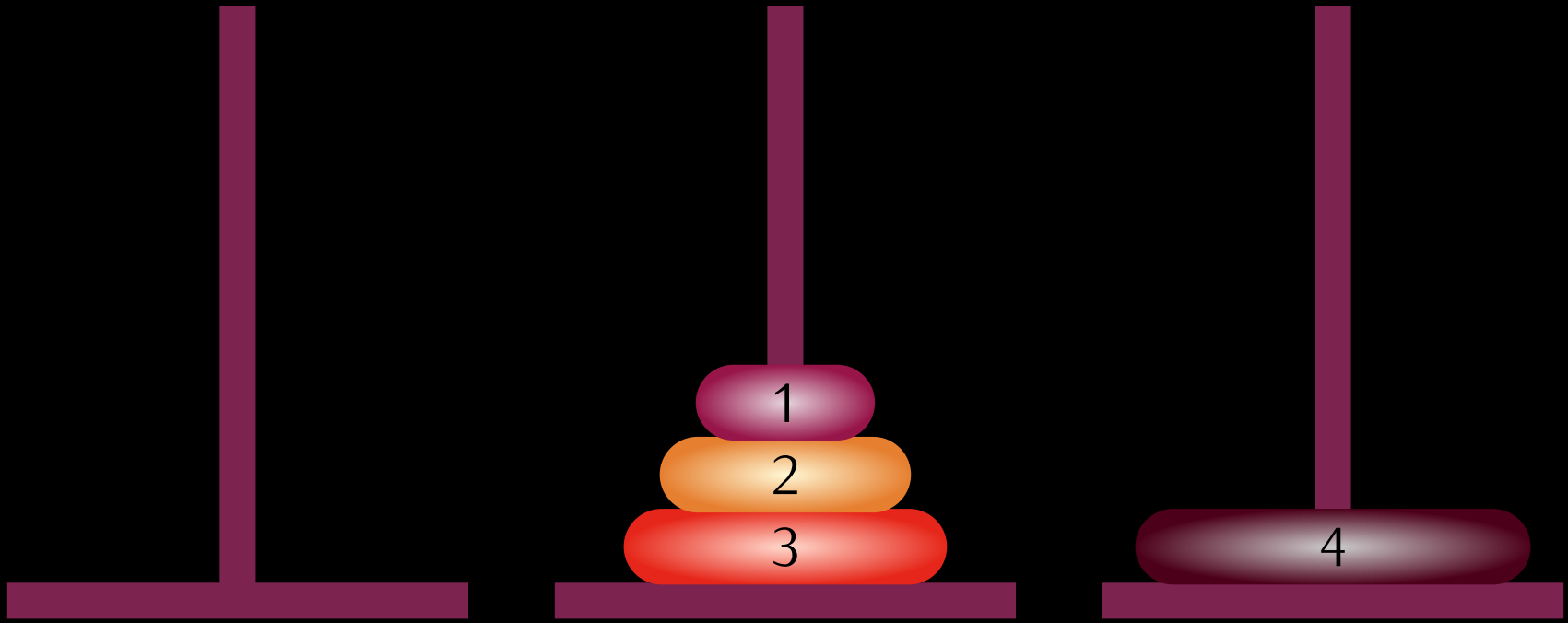
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 4 Disques



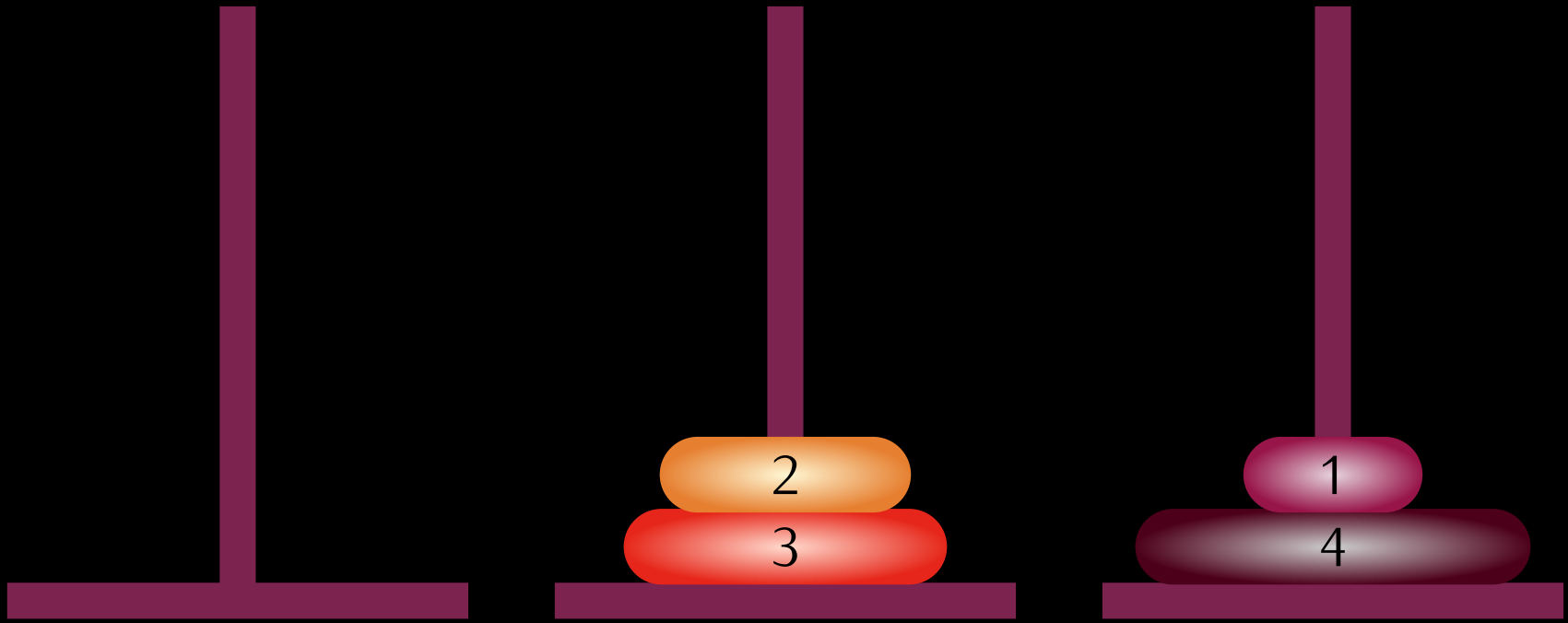
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



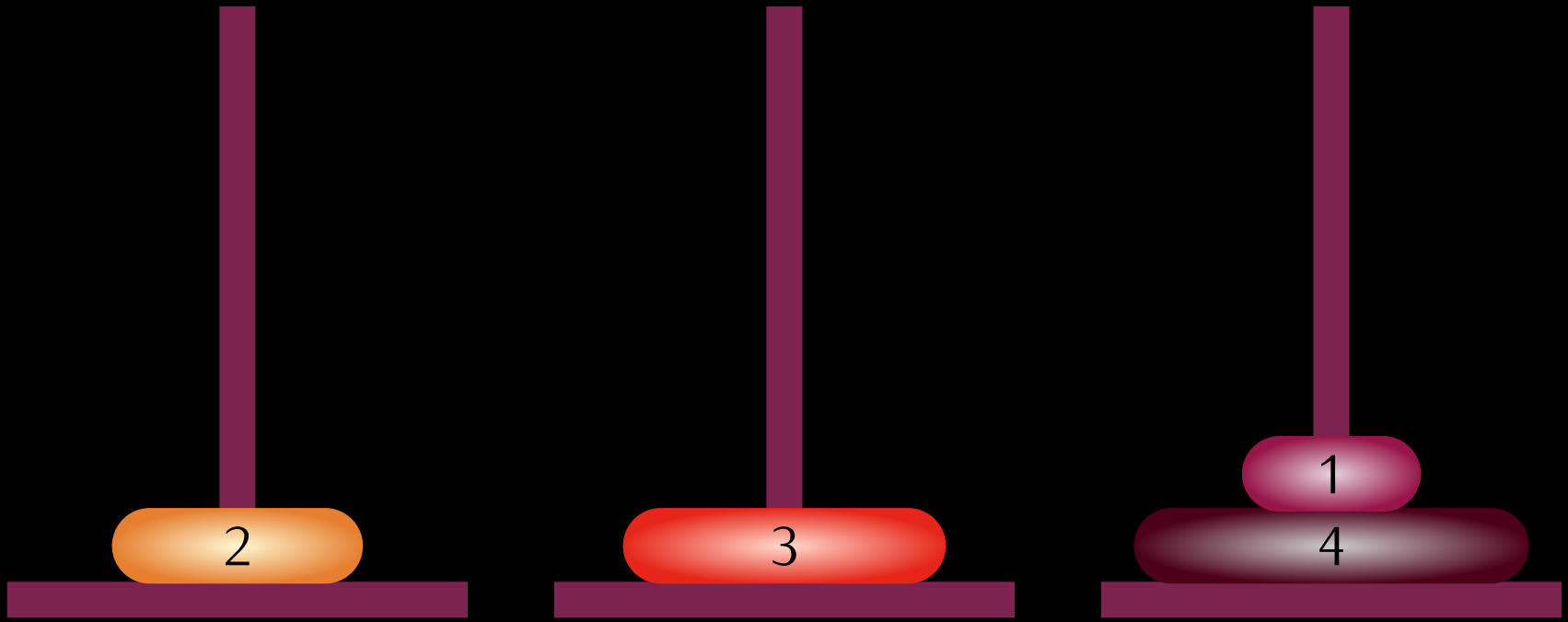
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



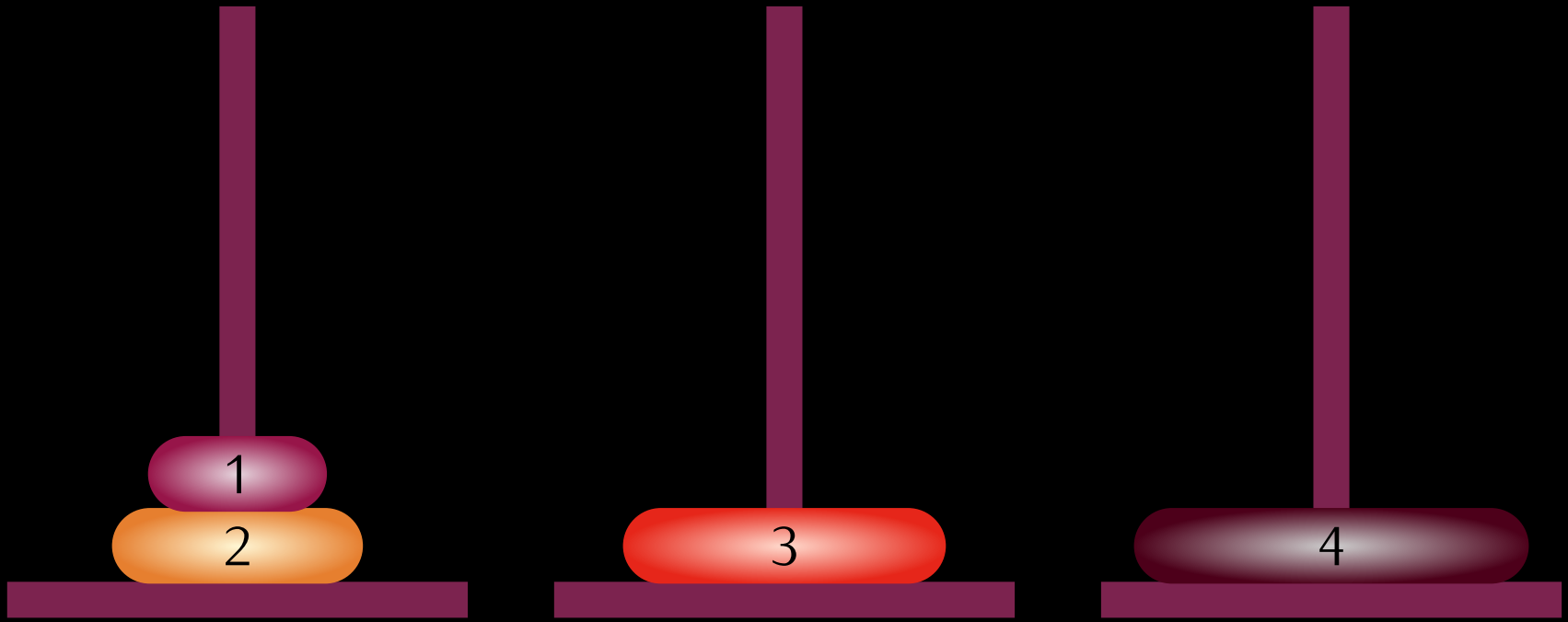
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



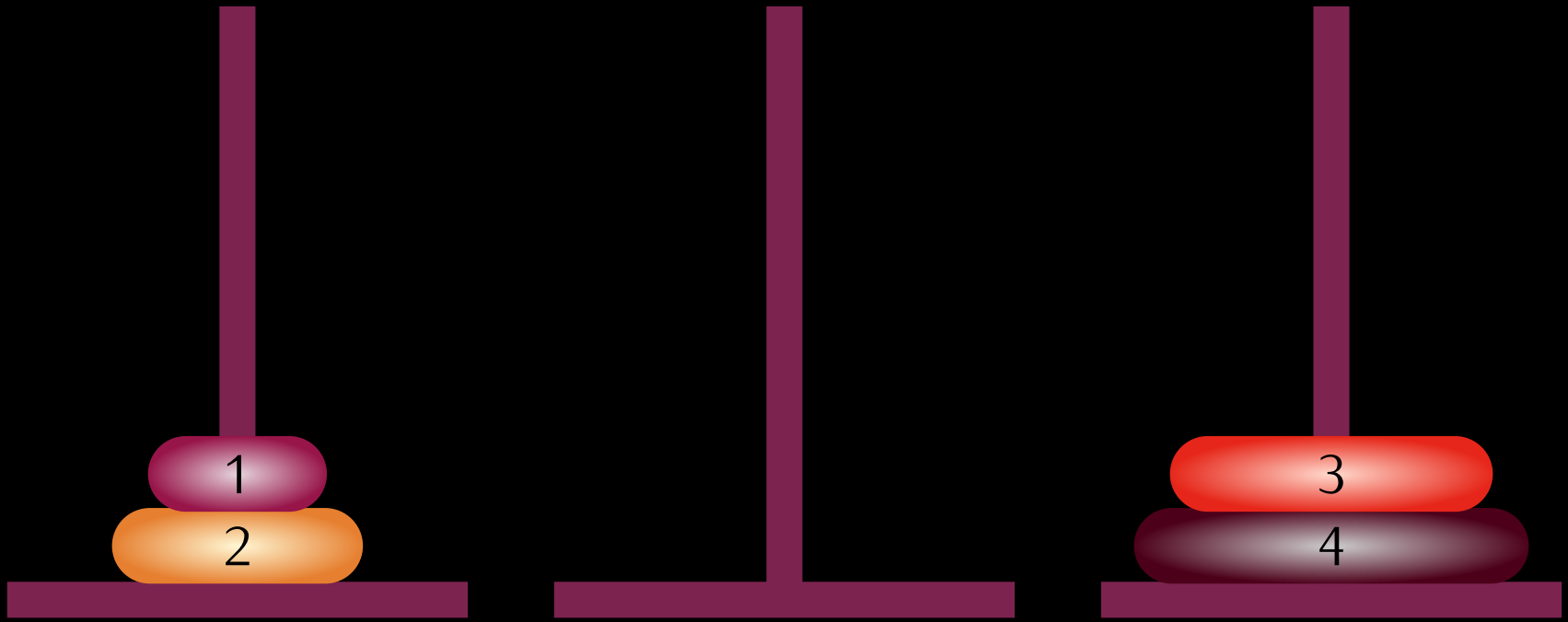
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 4 Disques



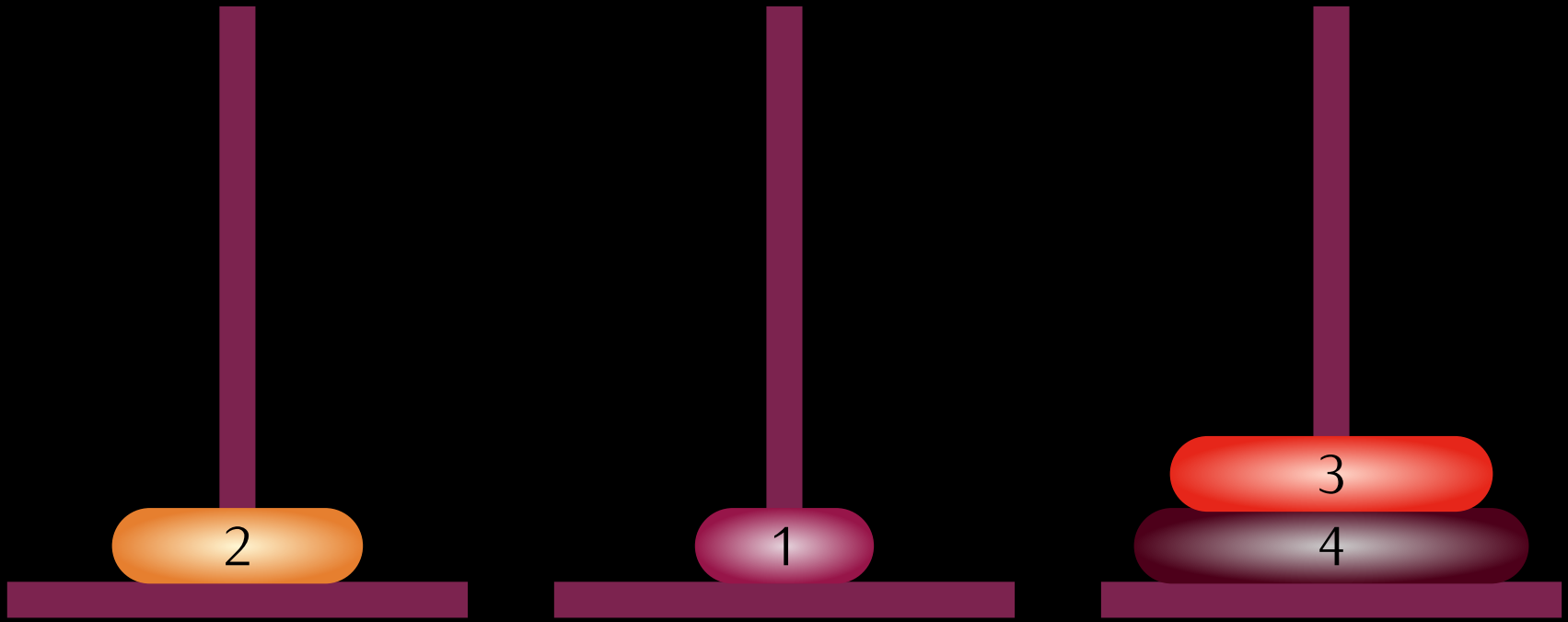
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 4 Disques



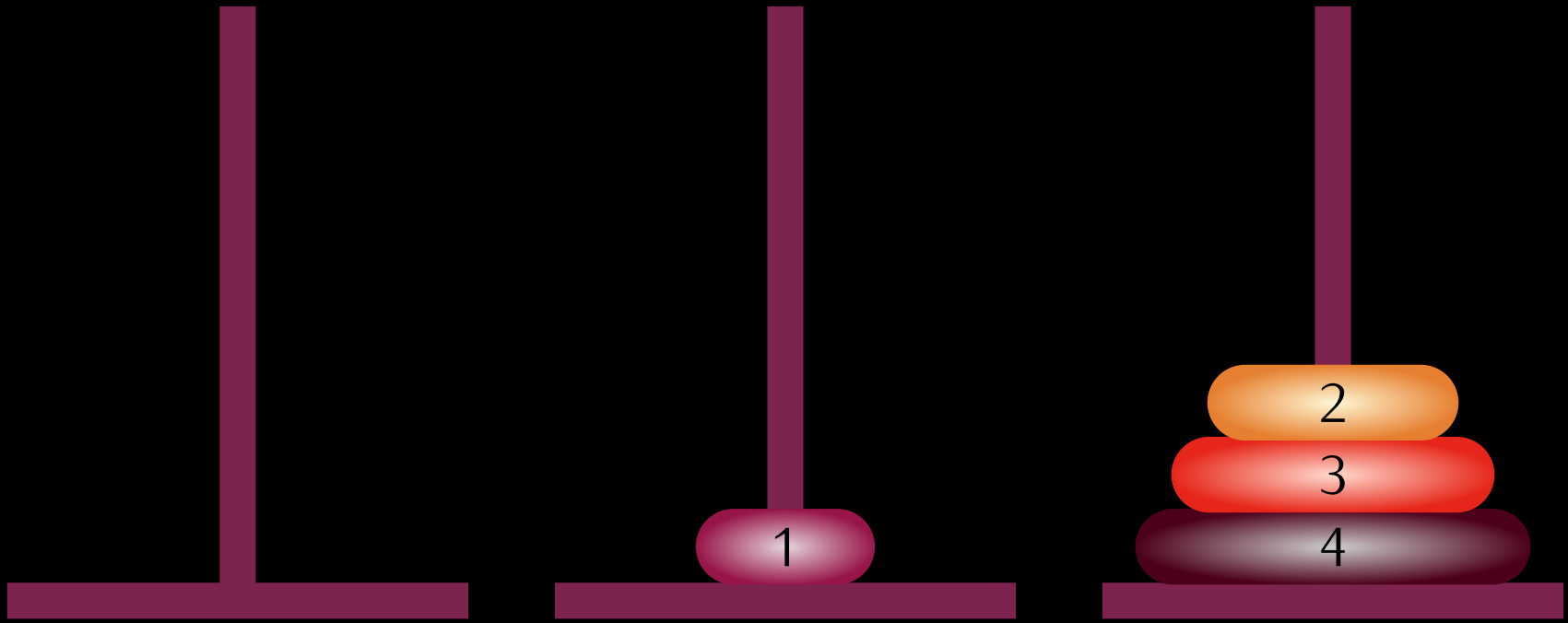
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



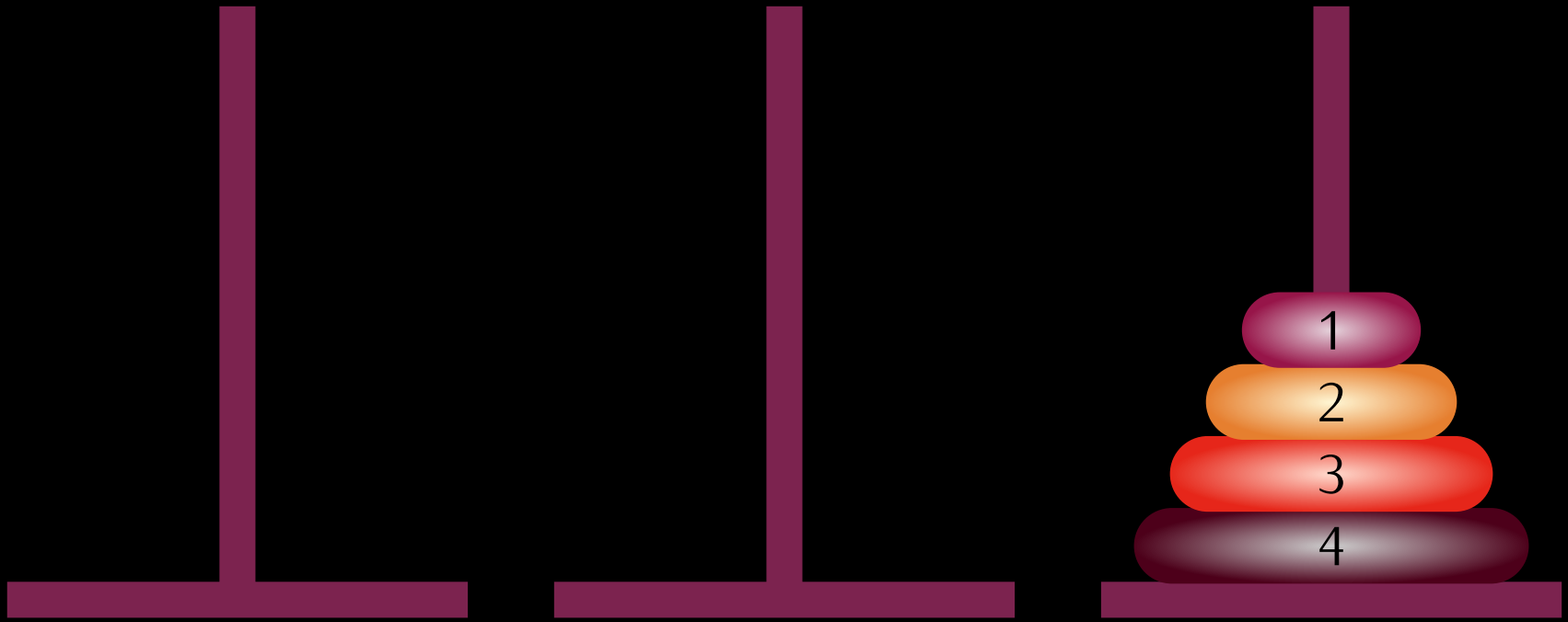
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



On déplace le disque de la tige 1 vers la tige 3.

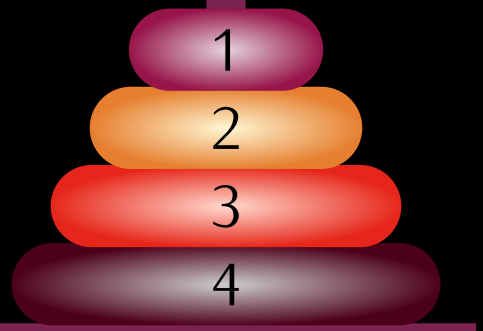
Tours de Hanoi – 4 Disques



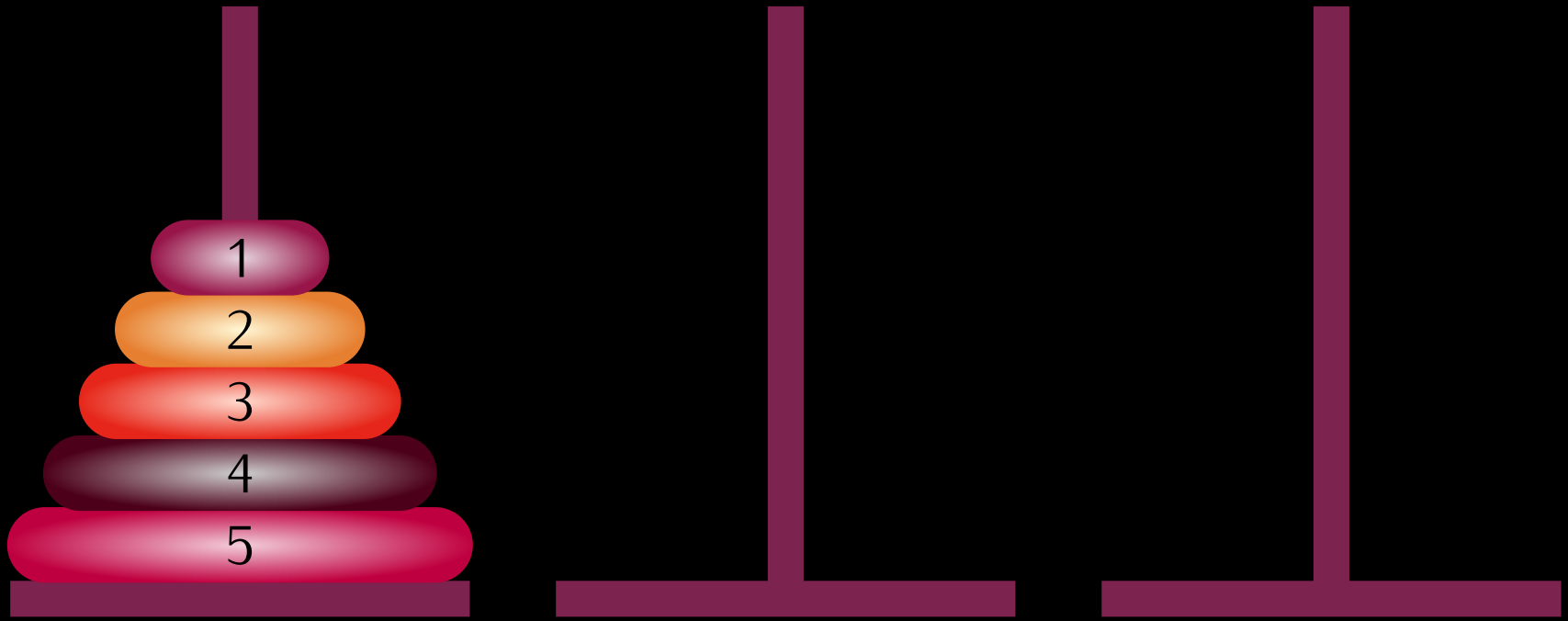
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques

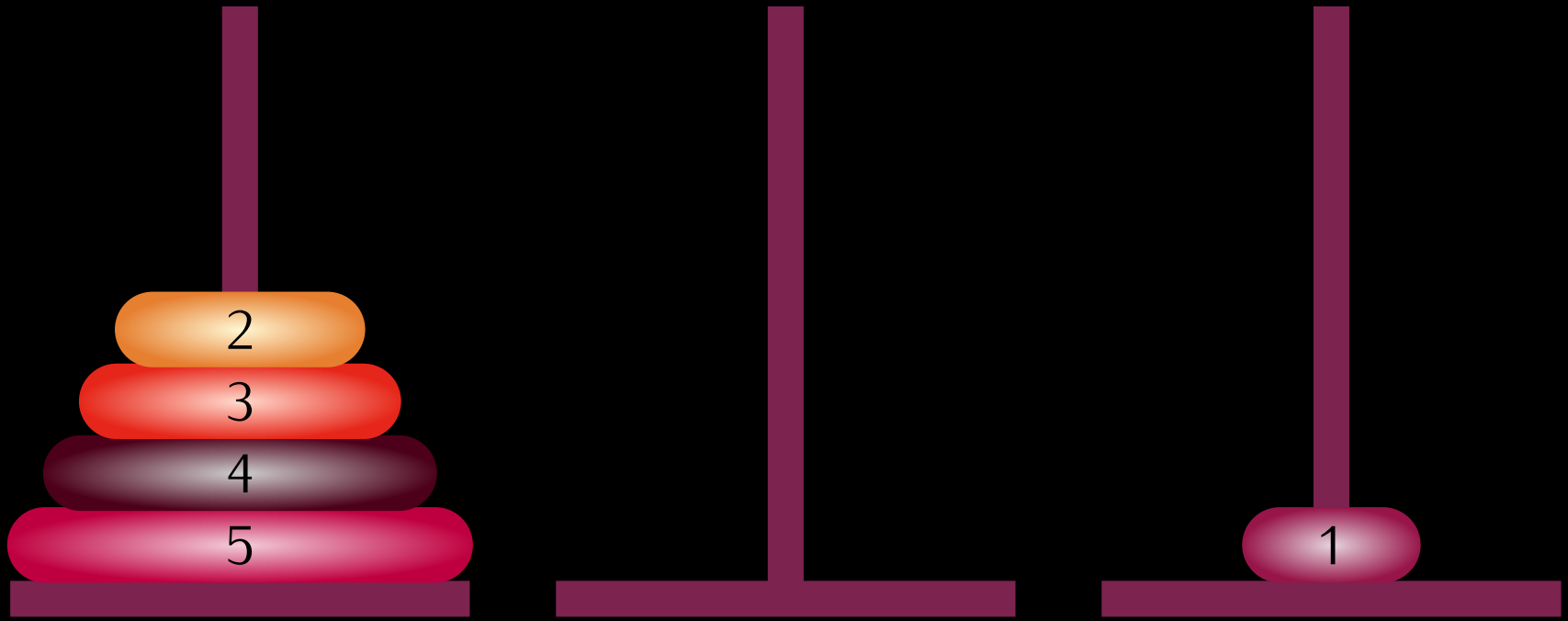
Ok



Tours de Hanoi – 5 Disques

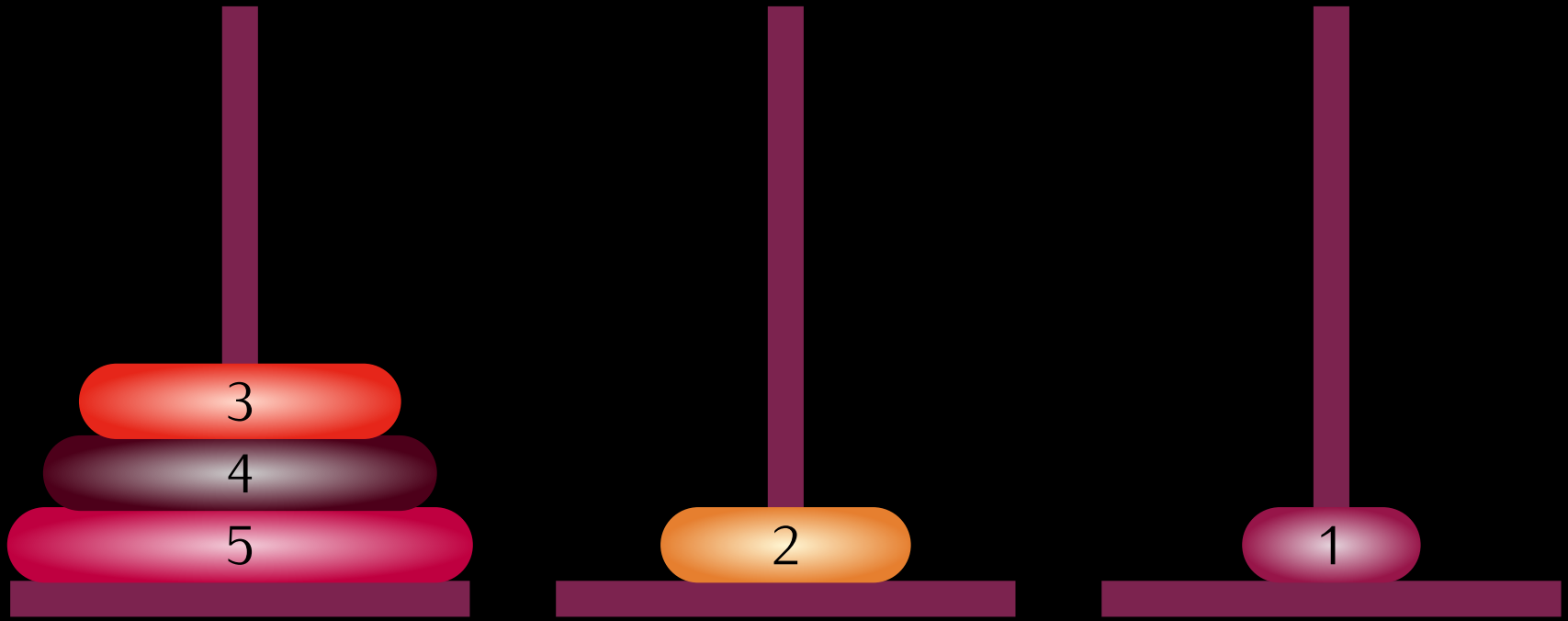


Tours de Hanoi – 5 Disques



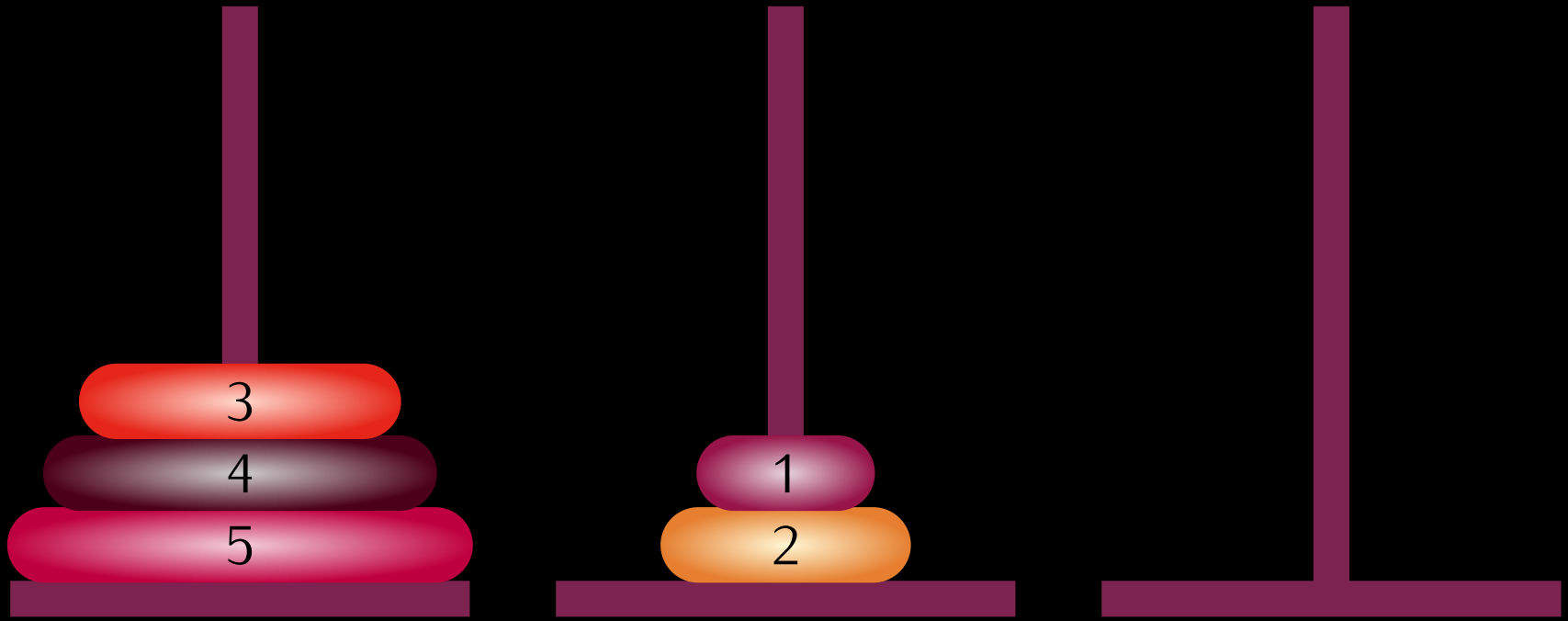
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



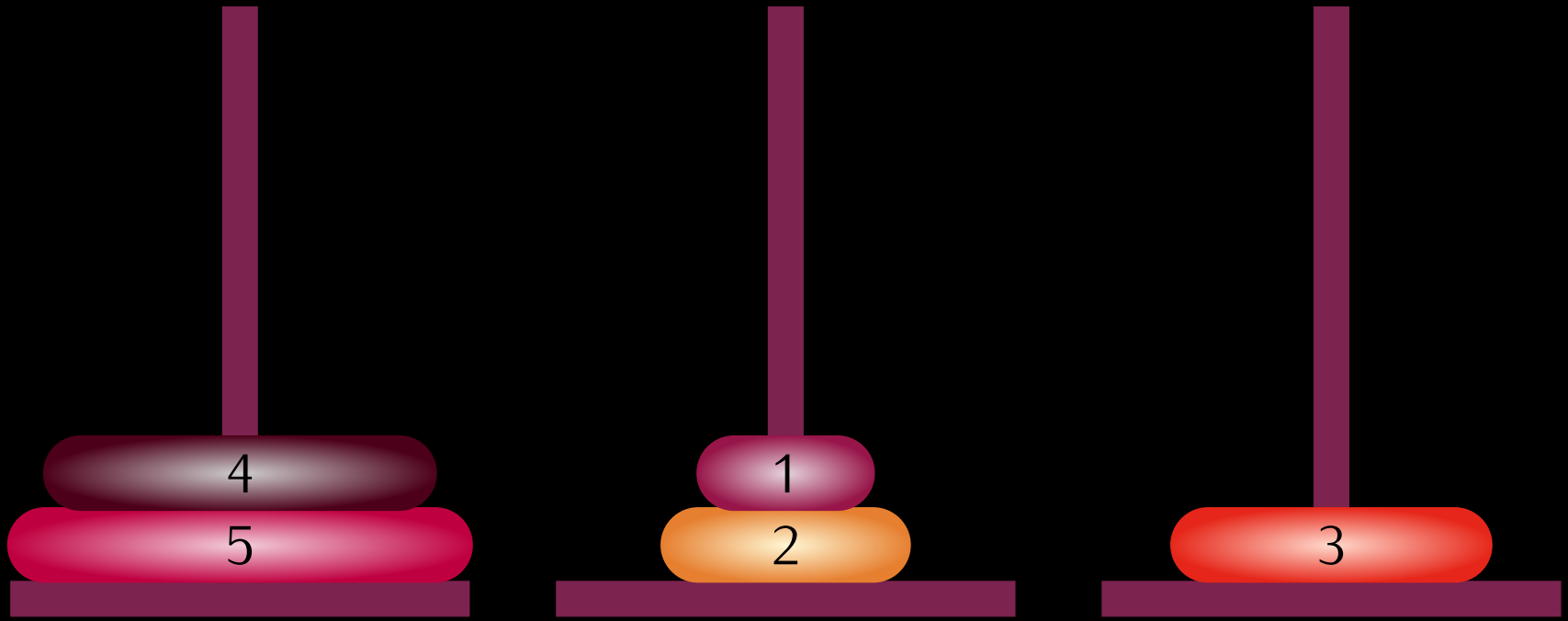
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



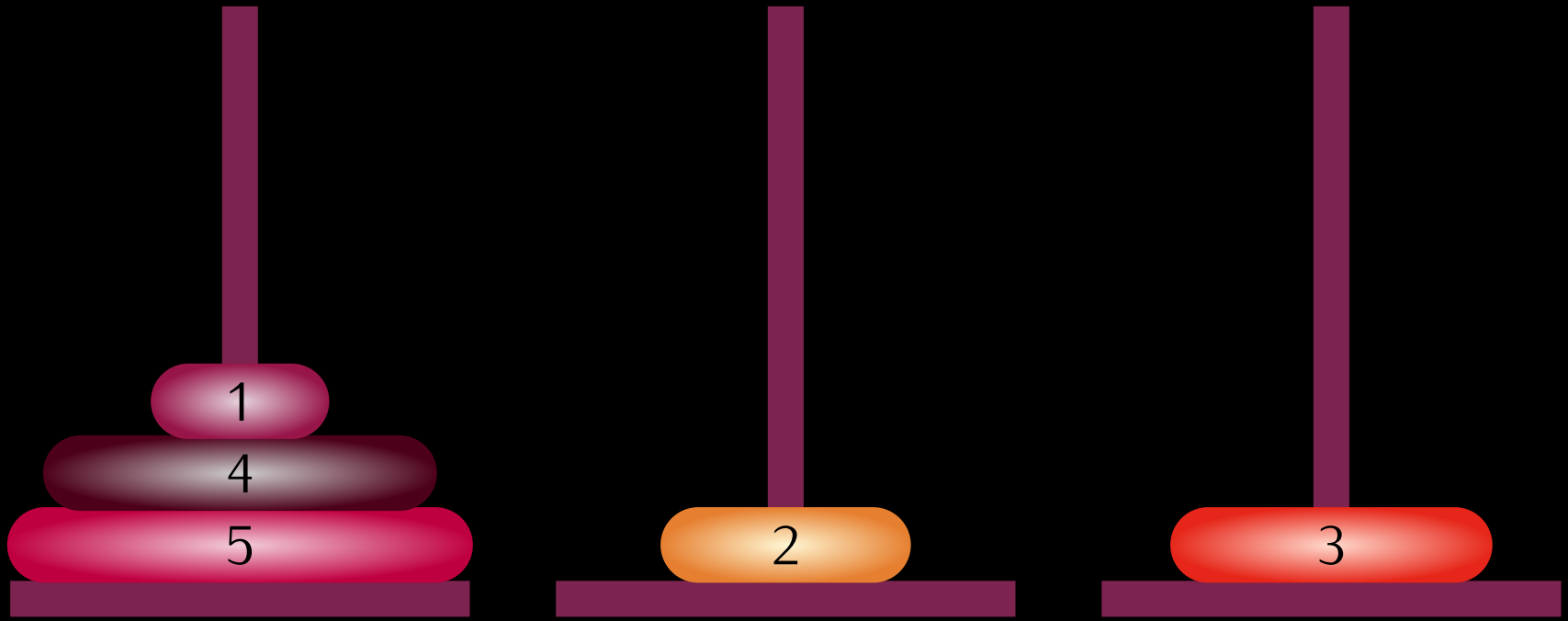
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



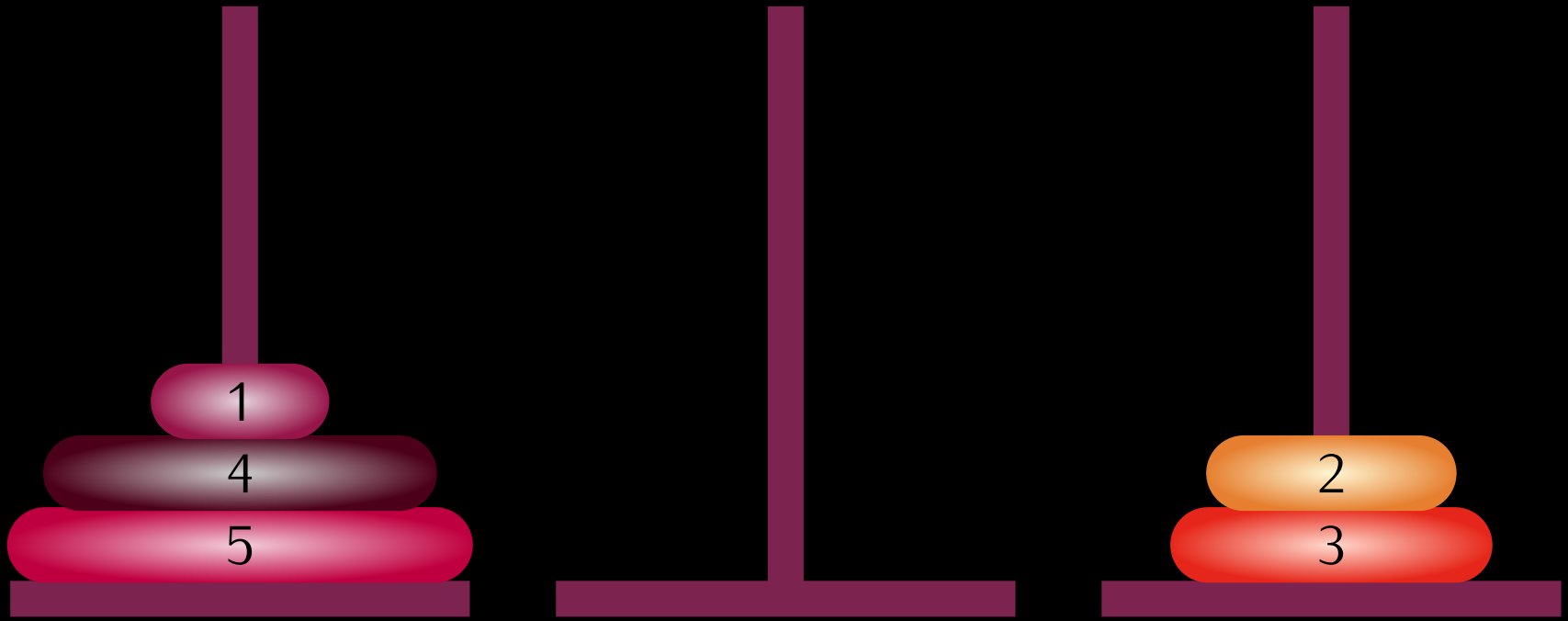
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



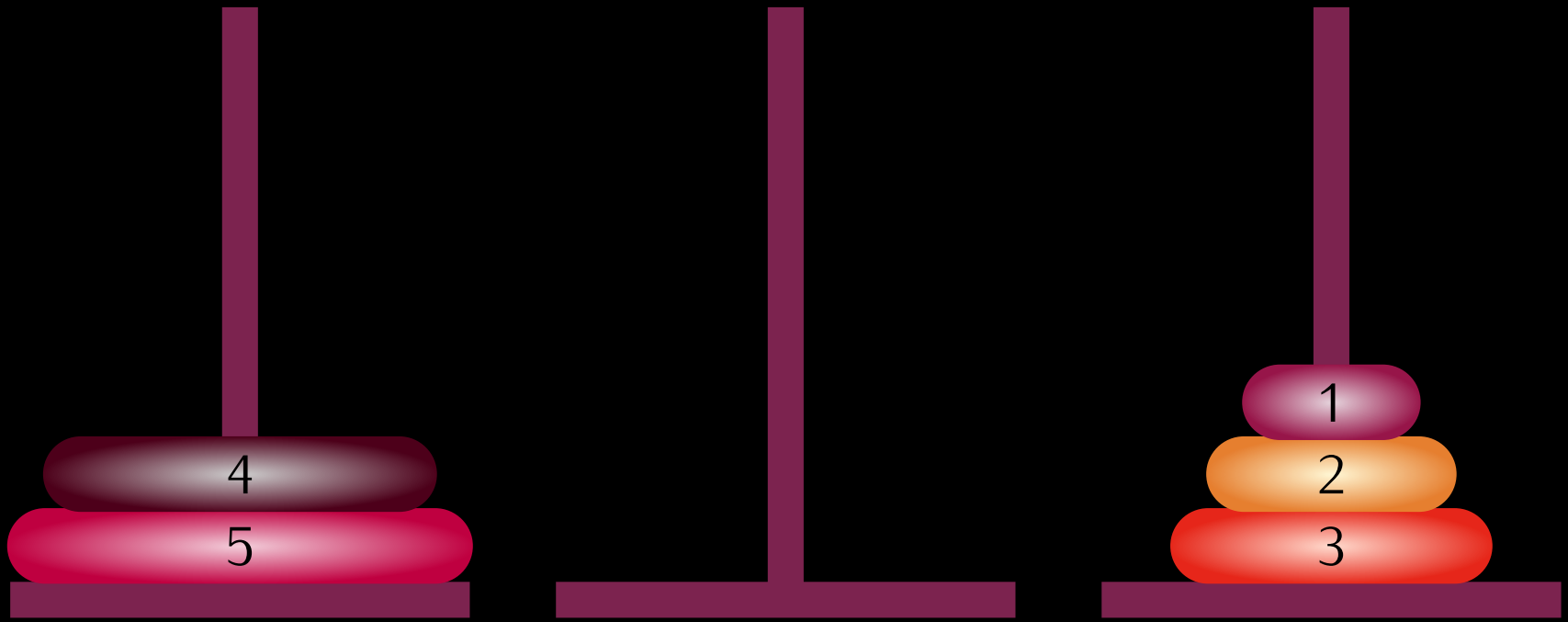
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



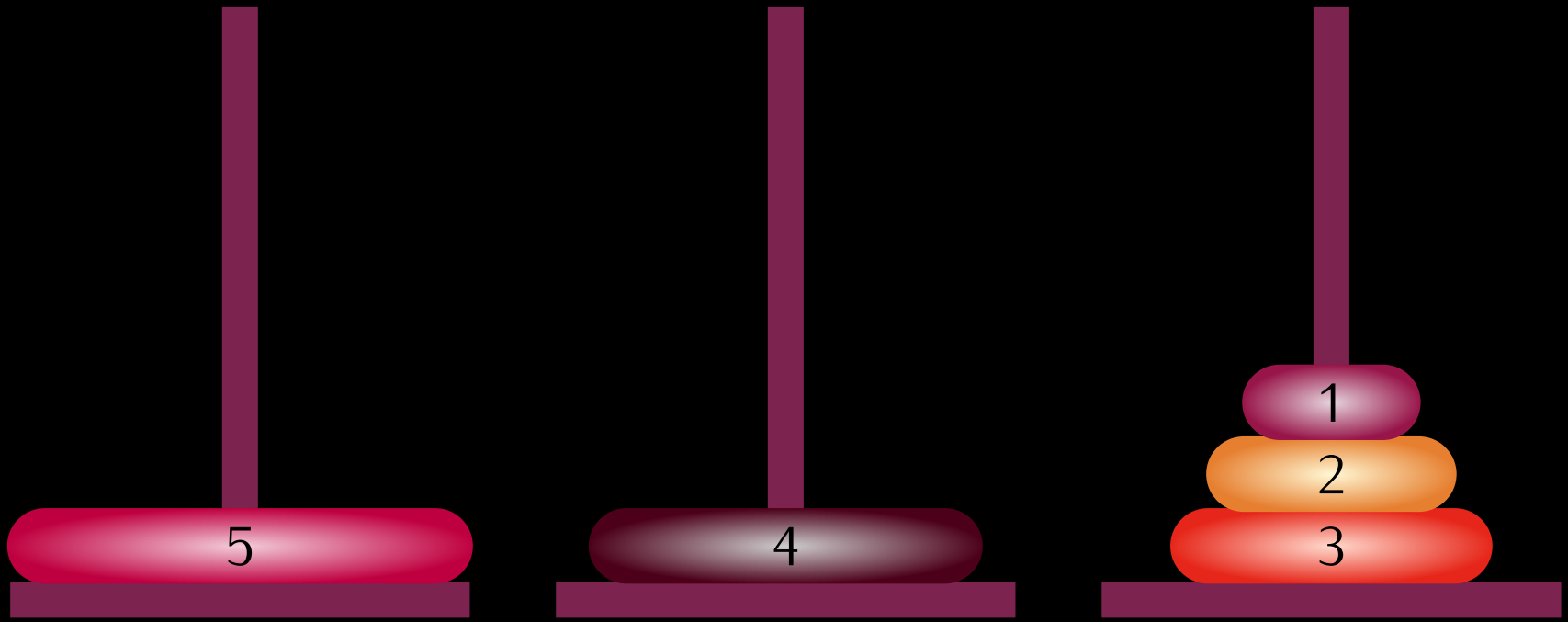
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



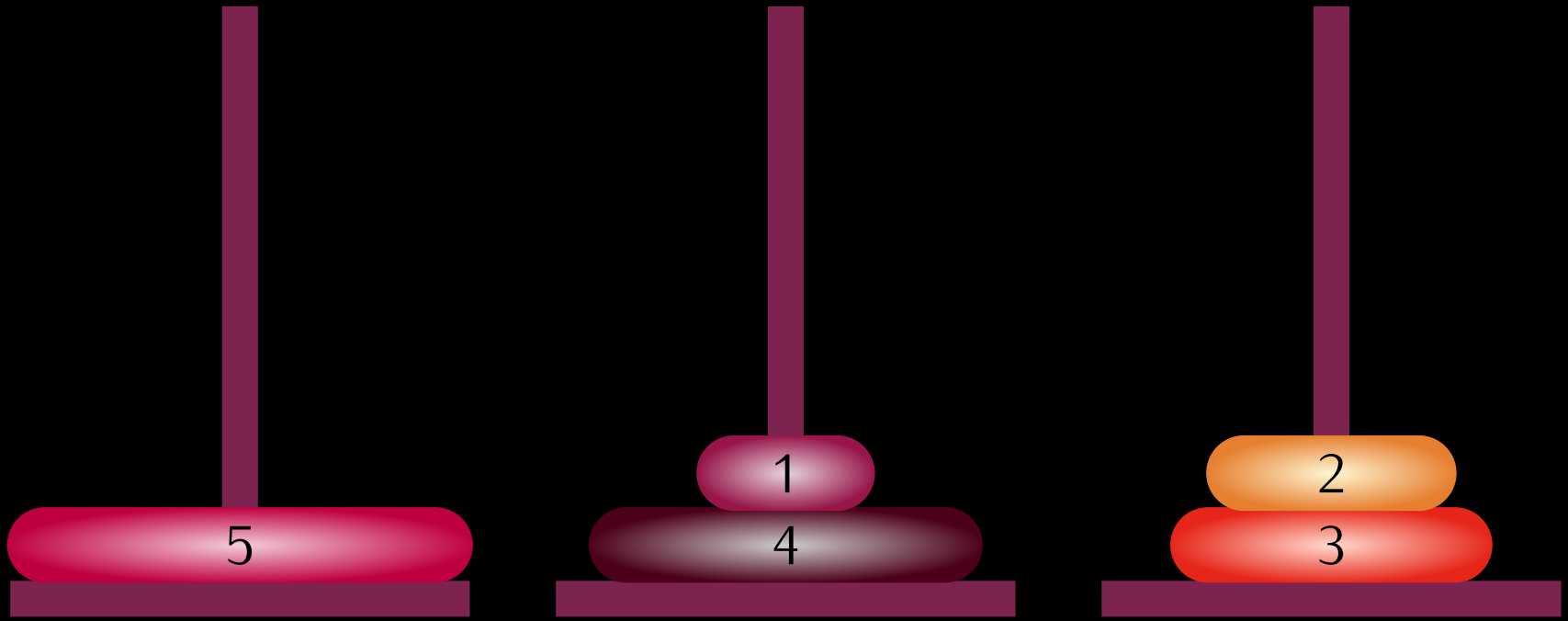
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



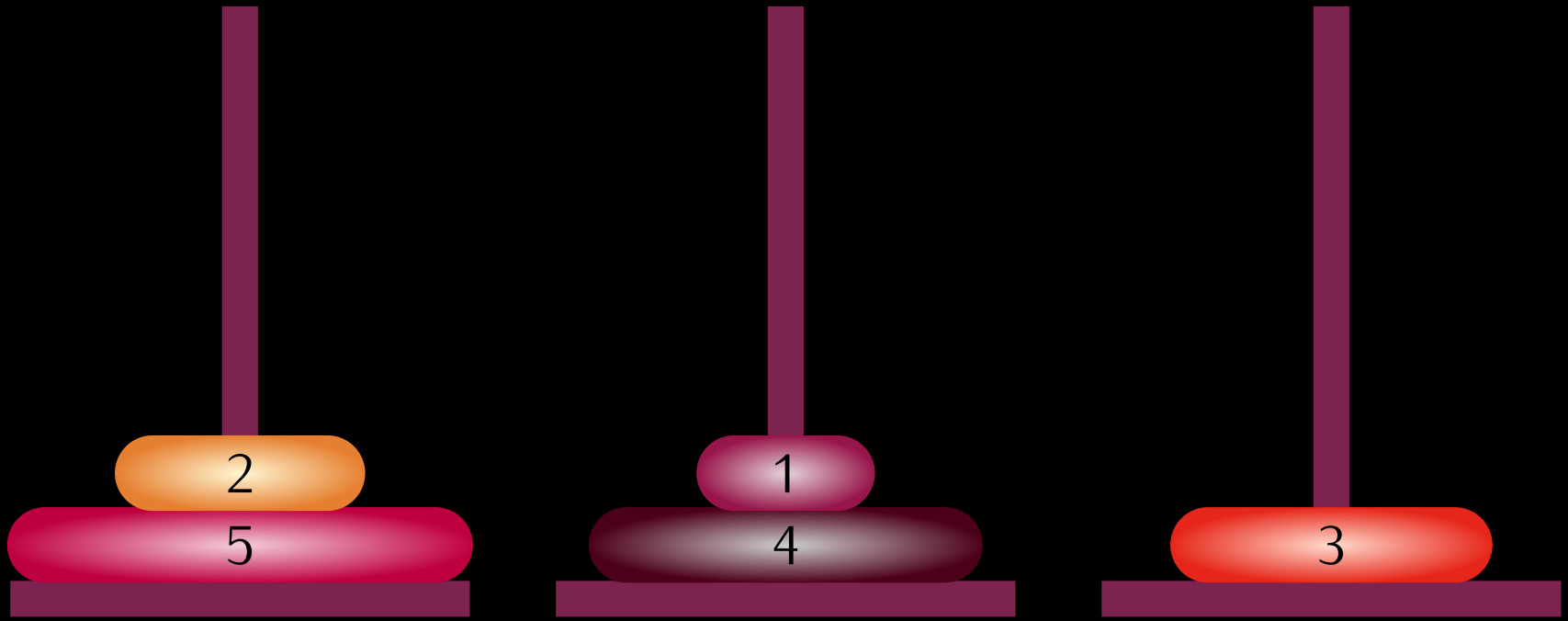
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



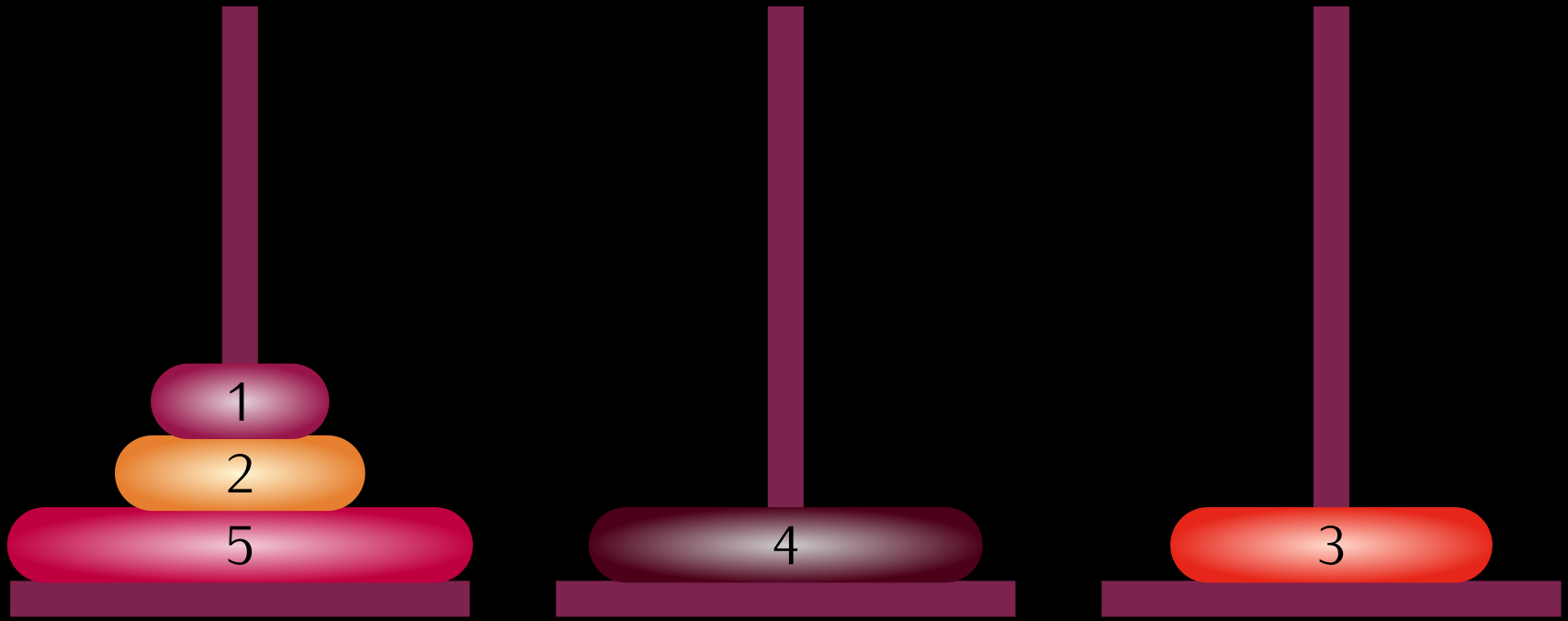
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



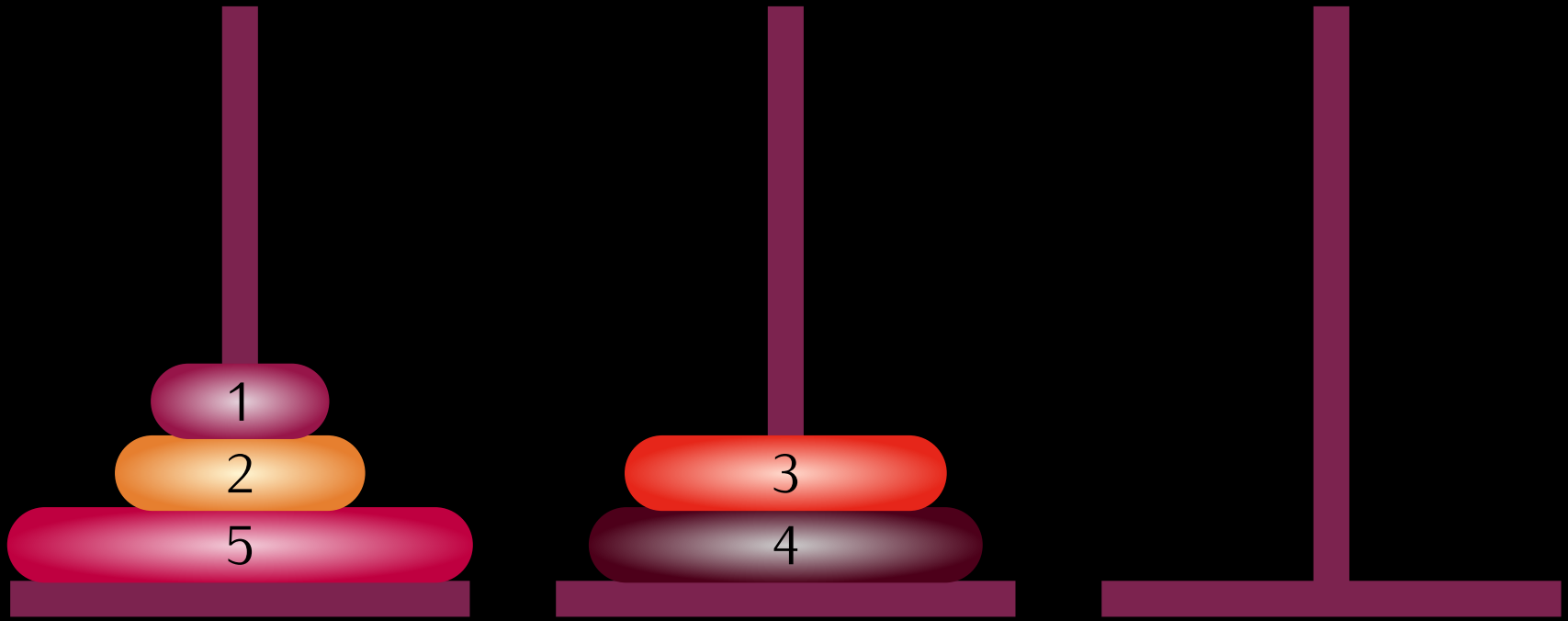
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 5 Disques



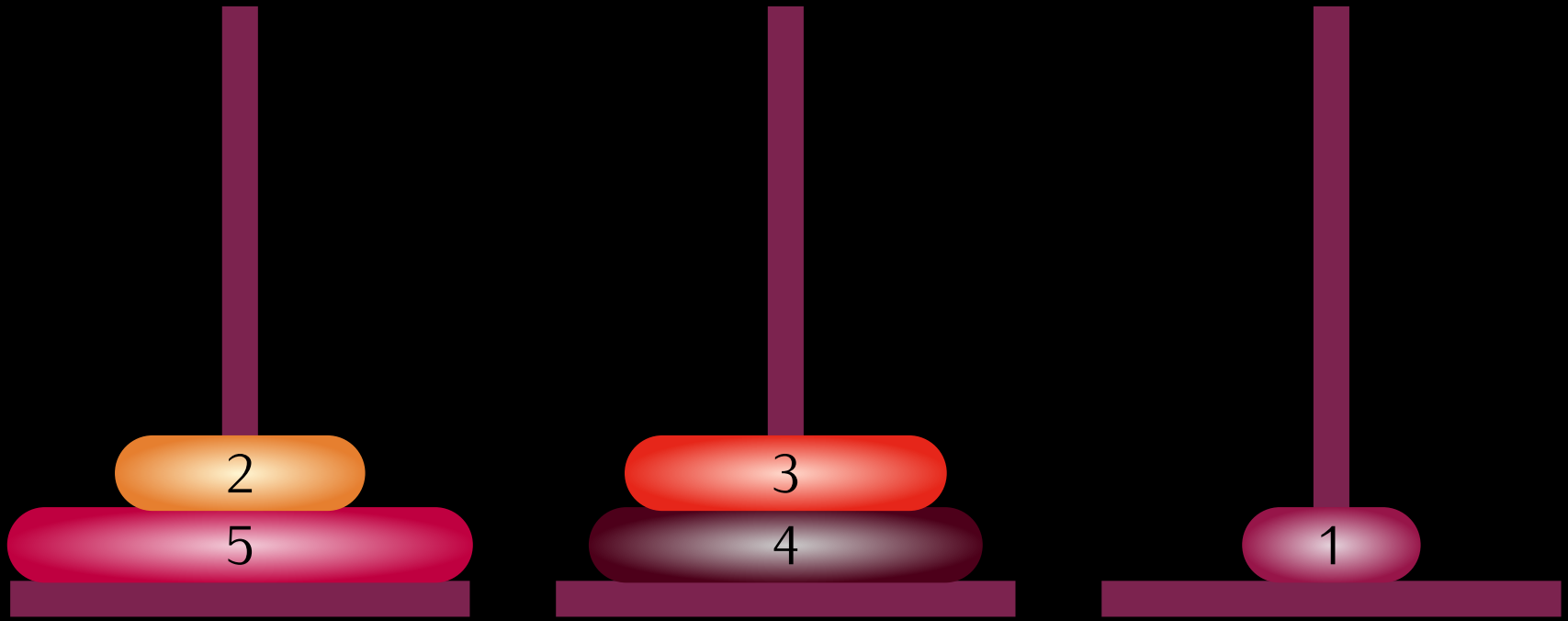
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



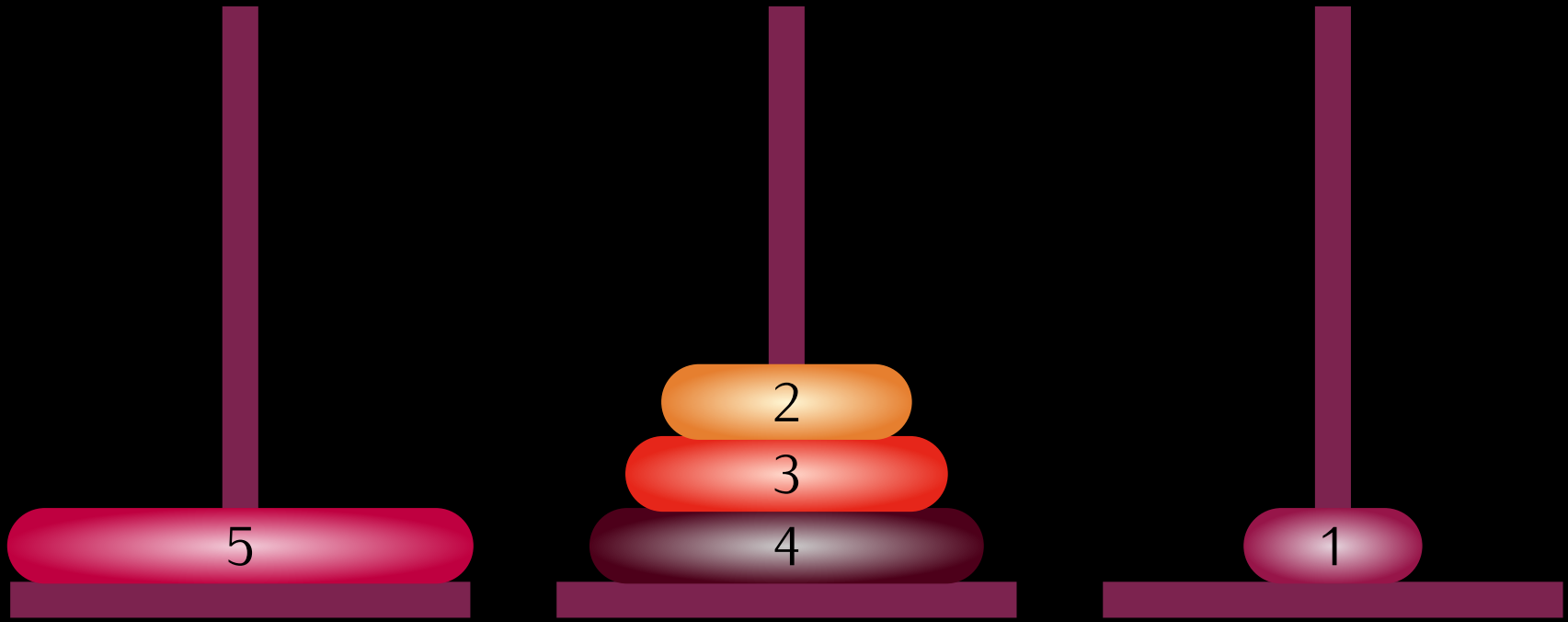
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



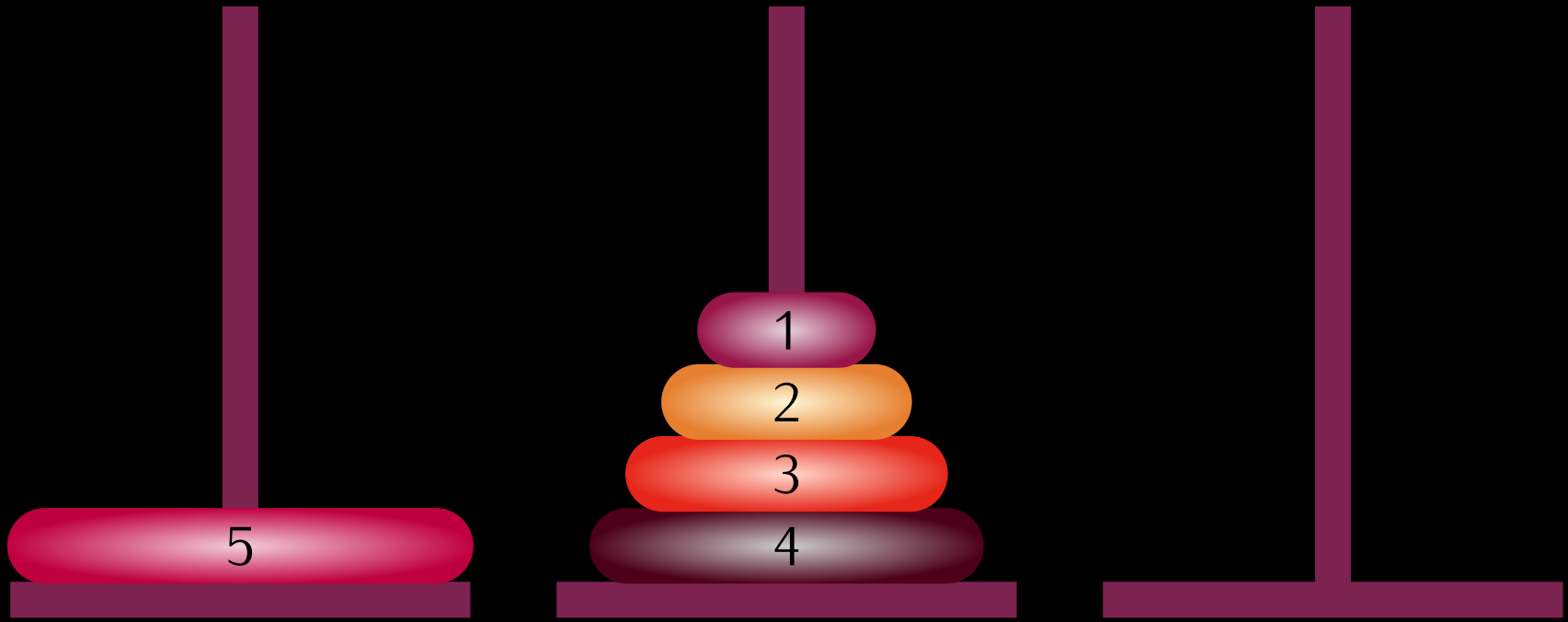
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



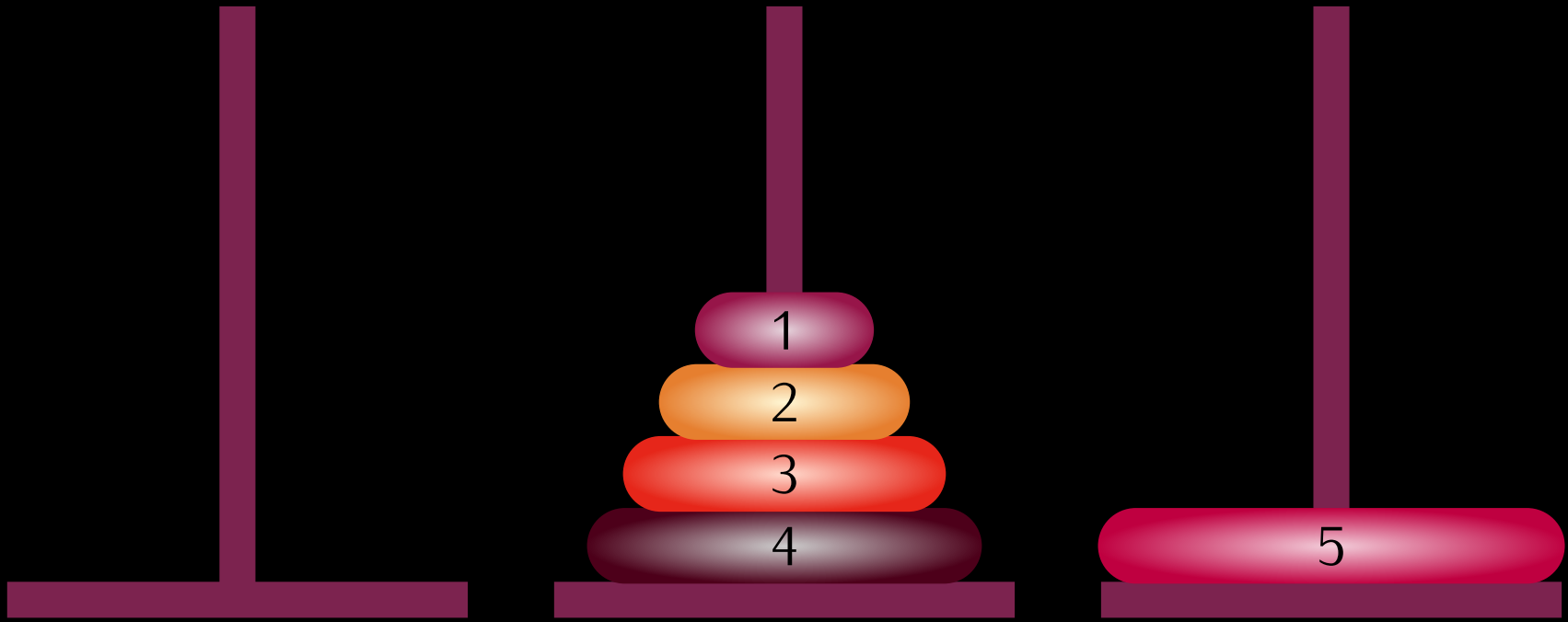
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



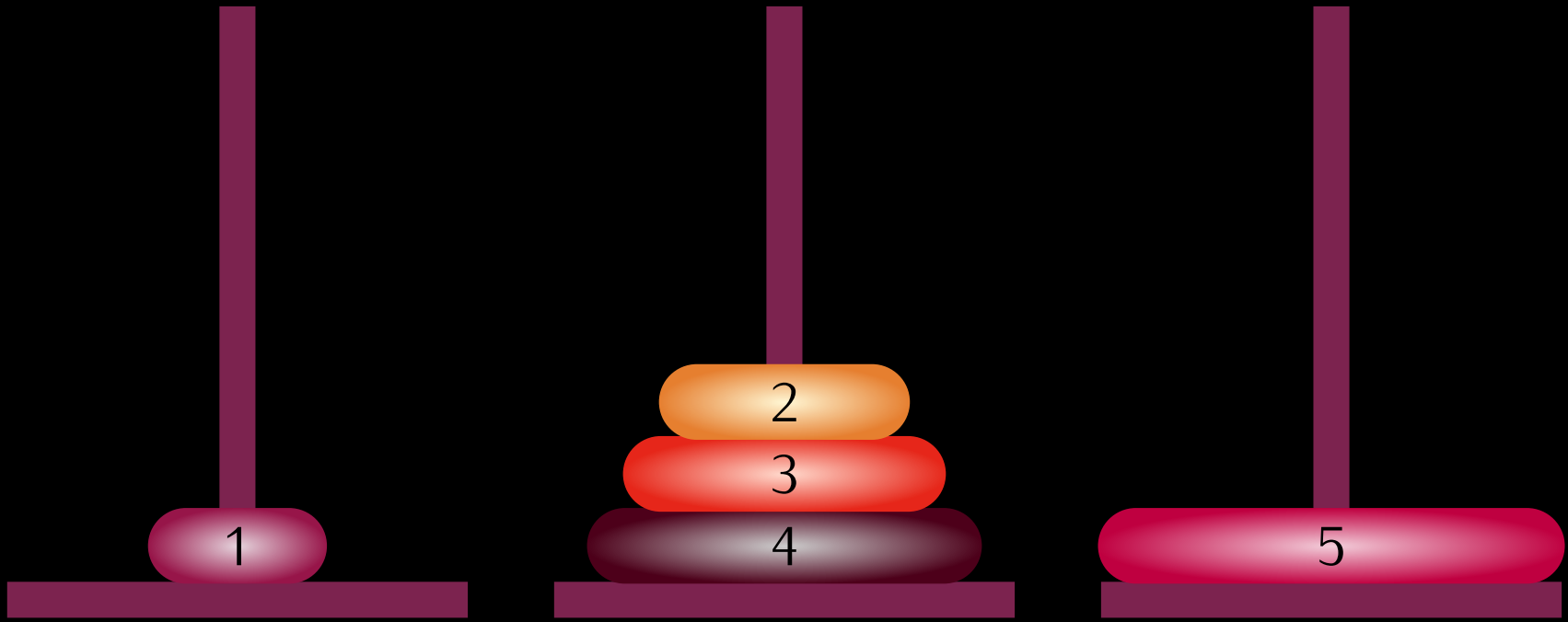
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



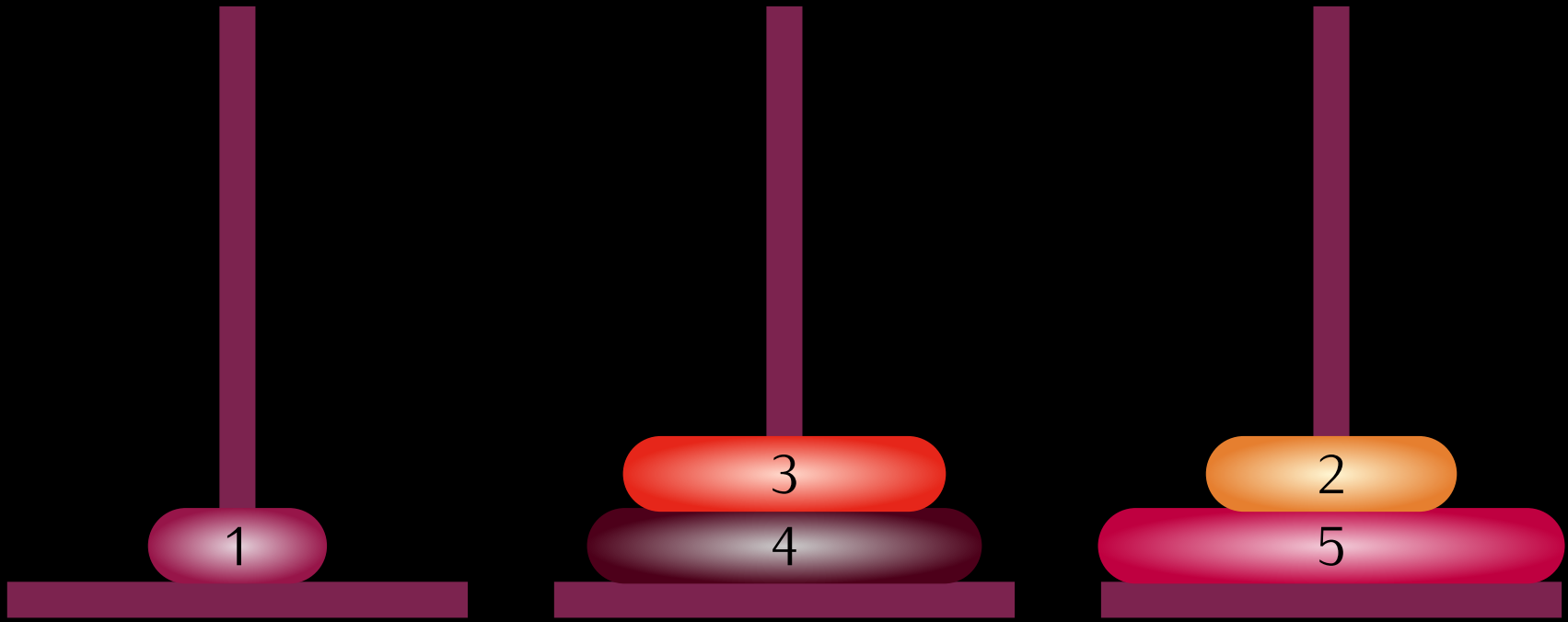
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



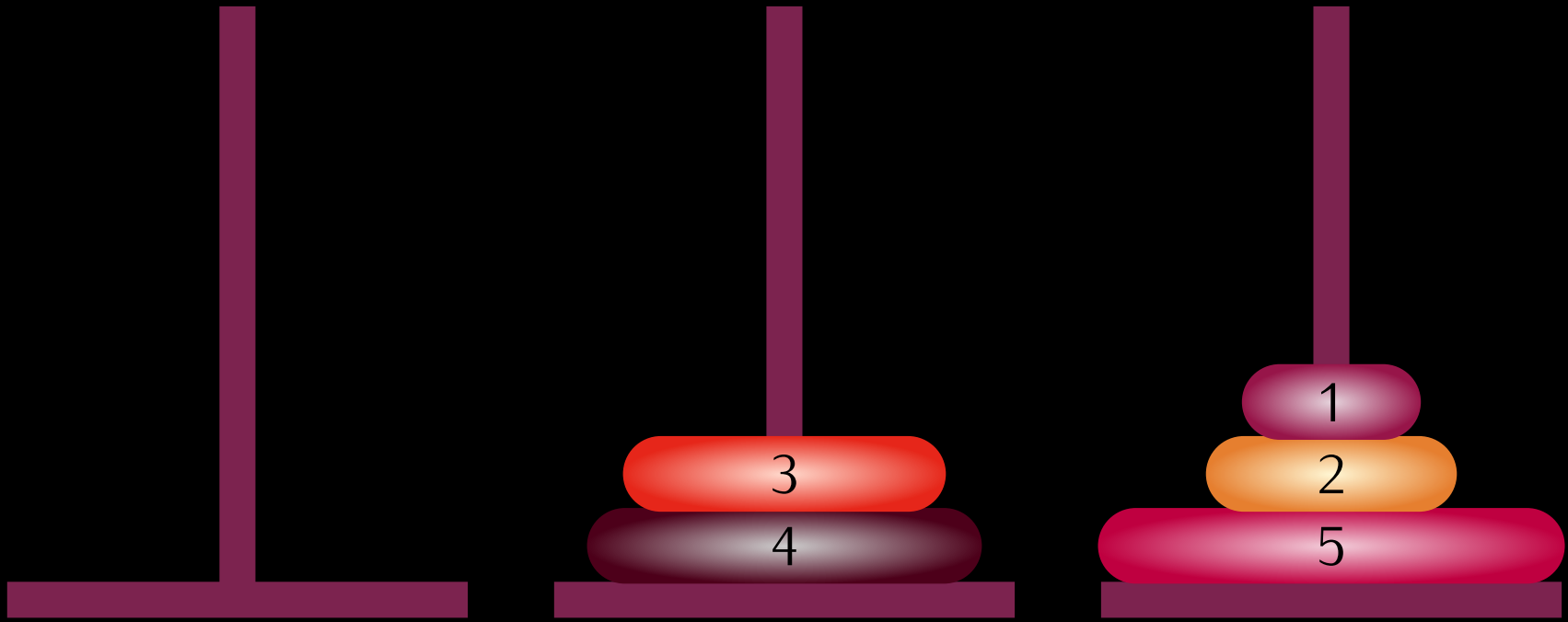
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



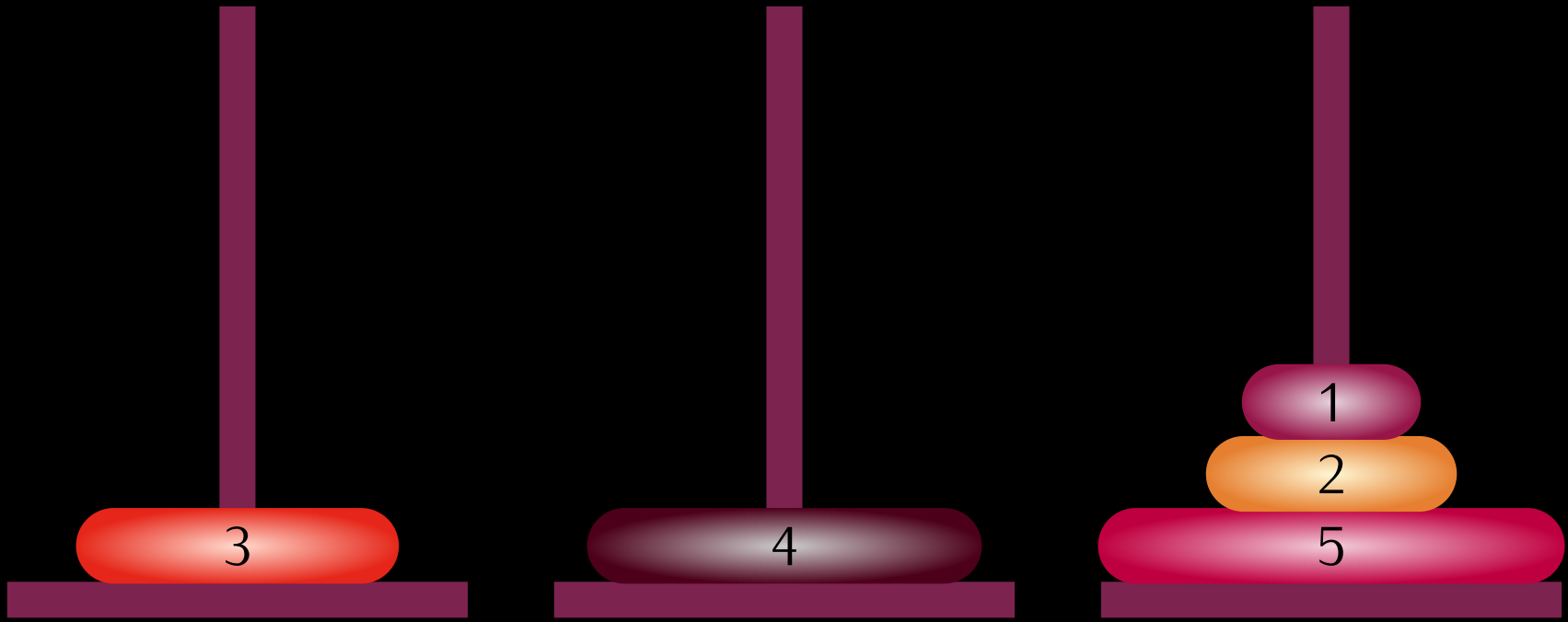
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



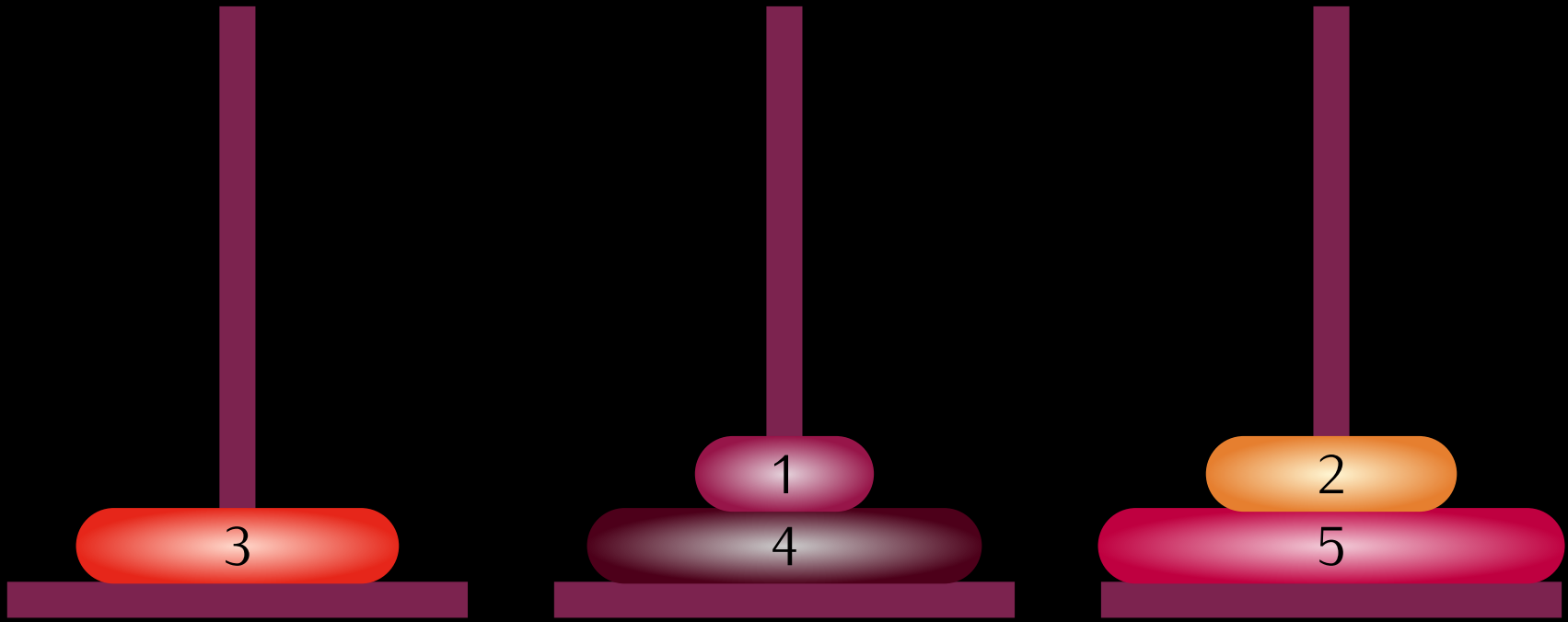
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



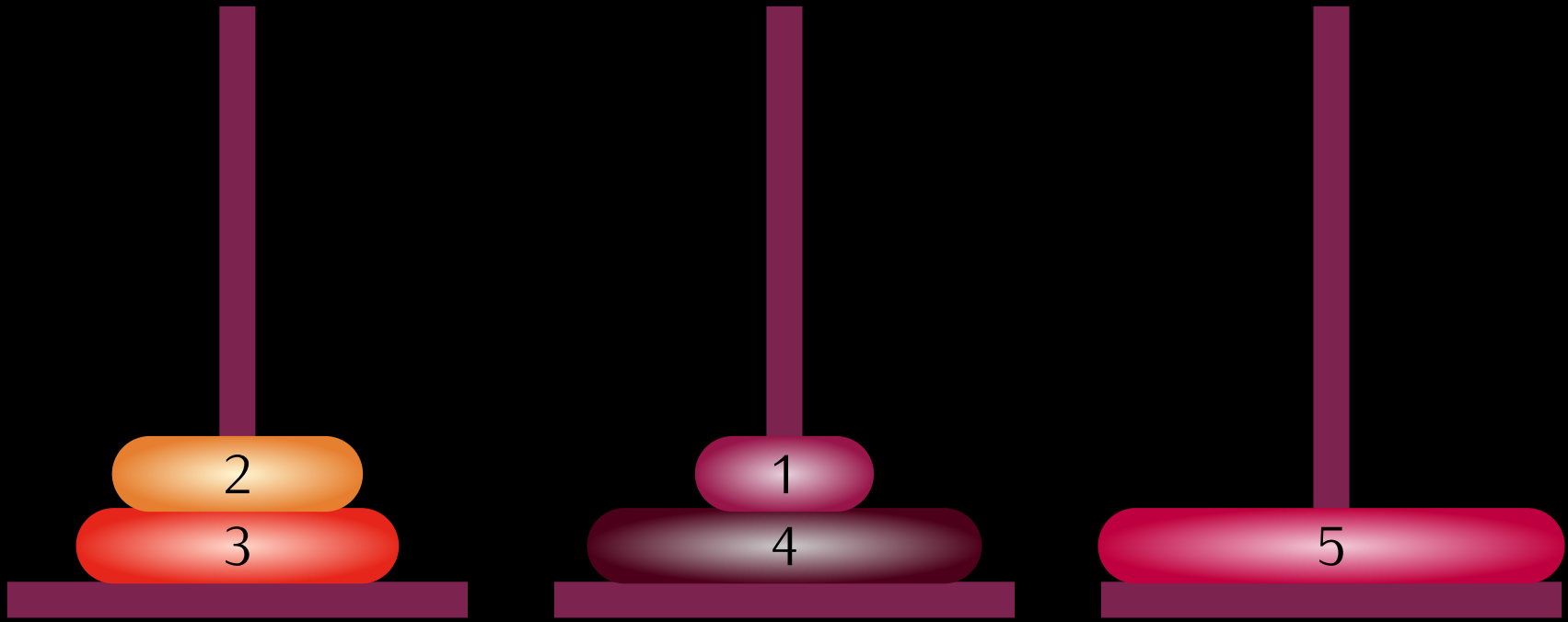
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



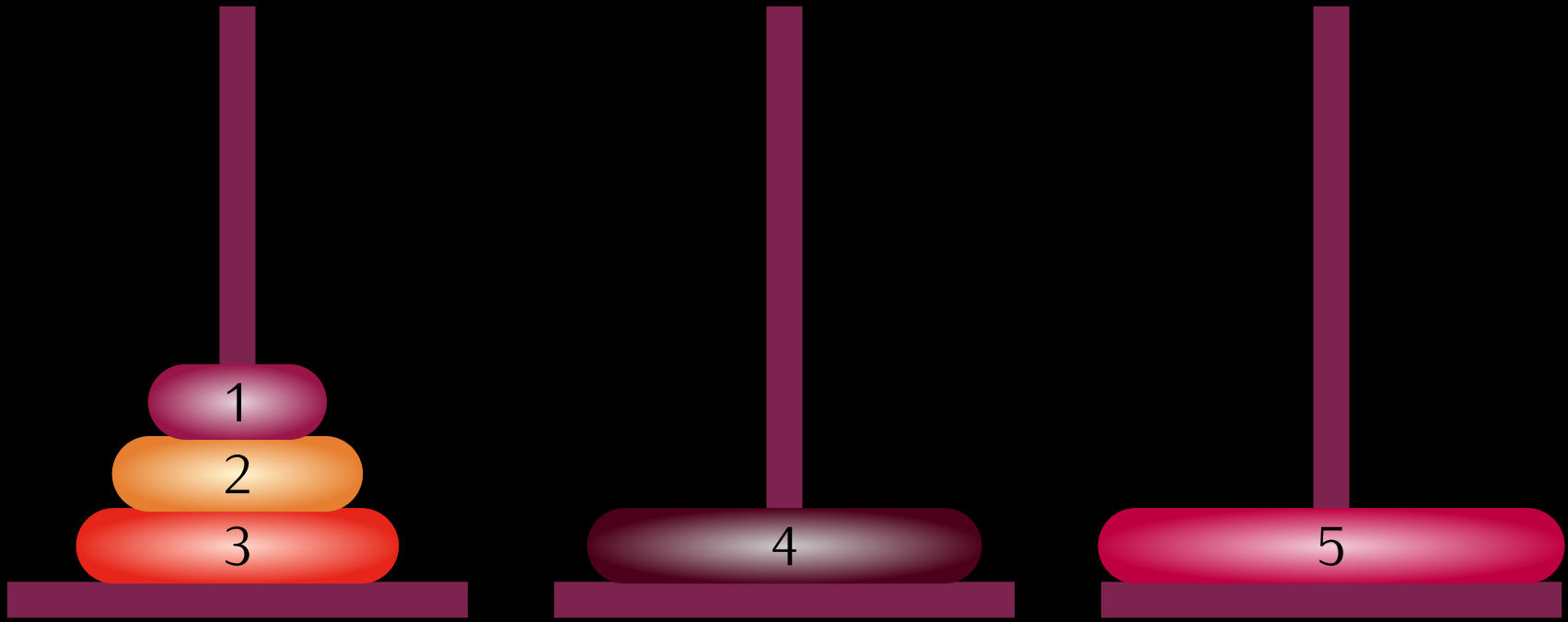
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



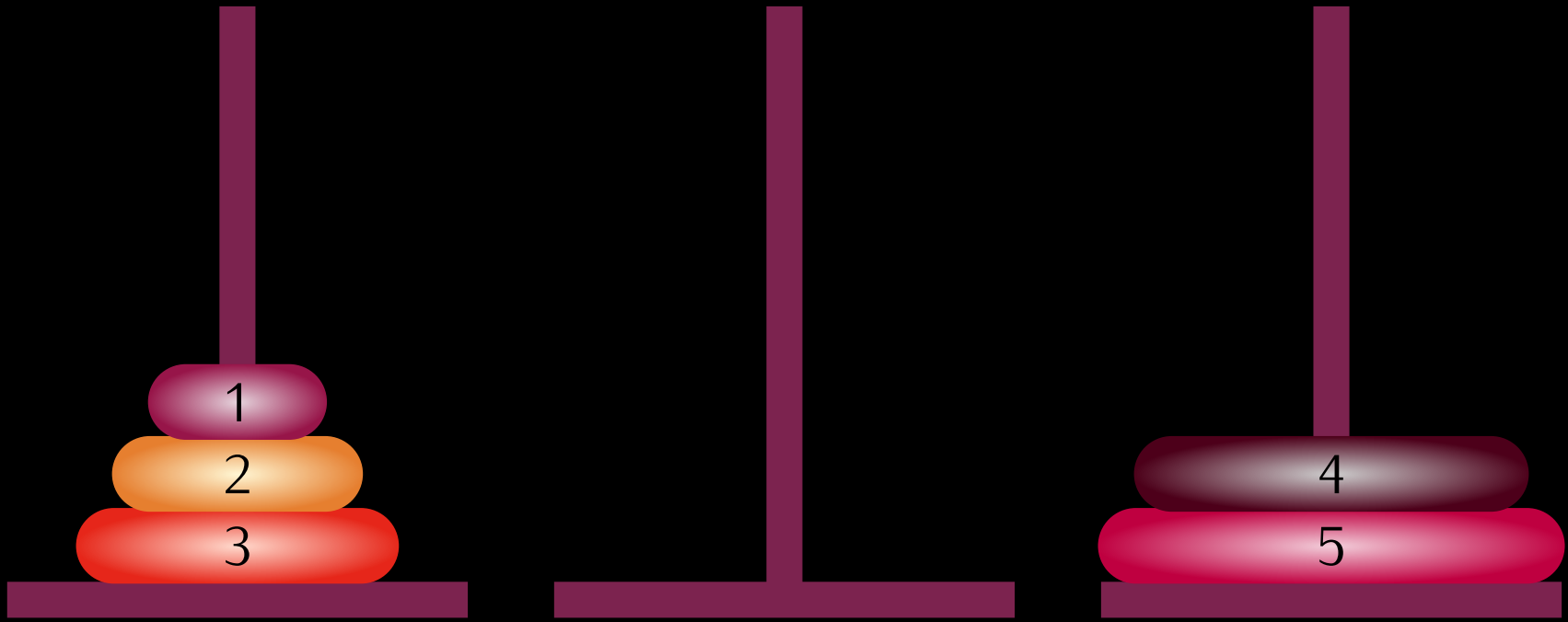
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 5 Disques



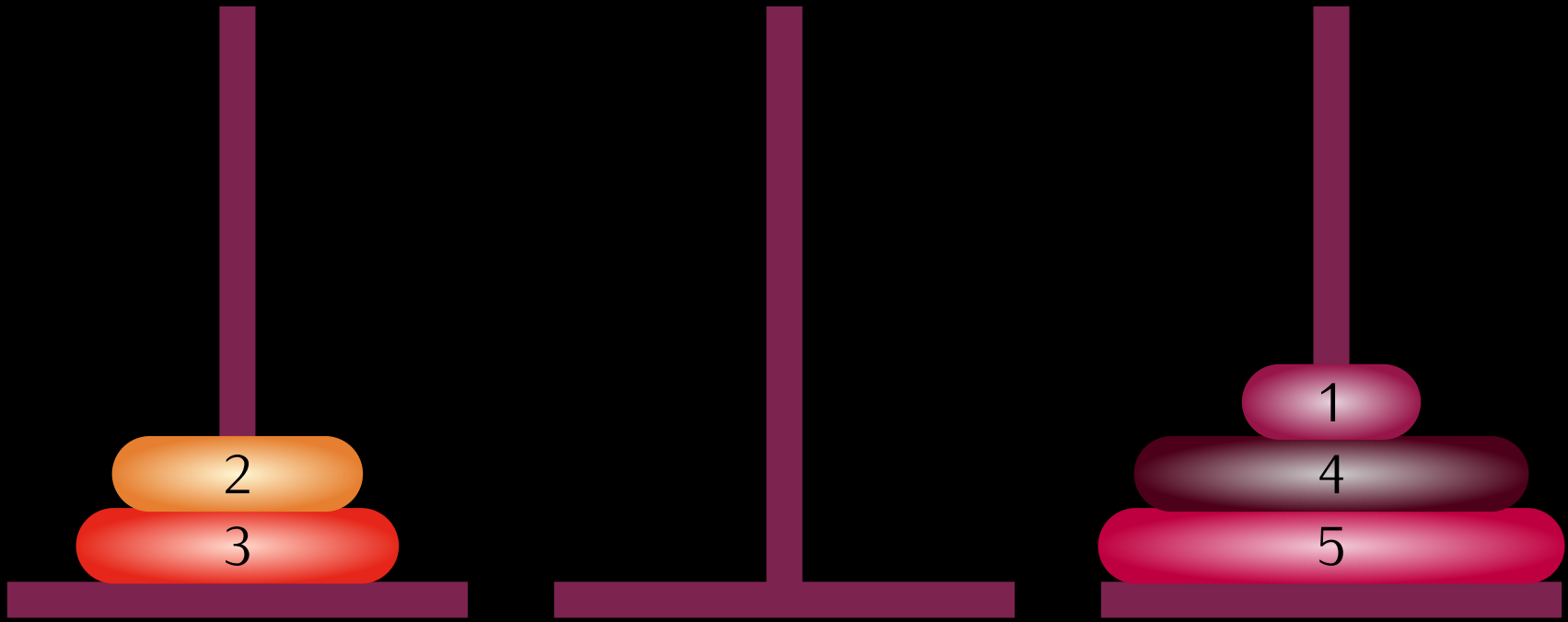
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



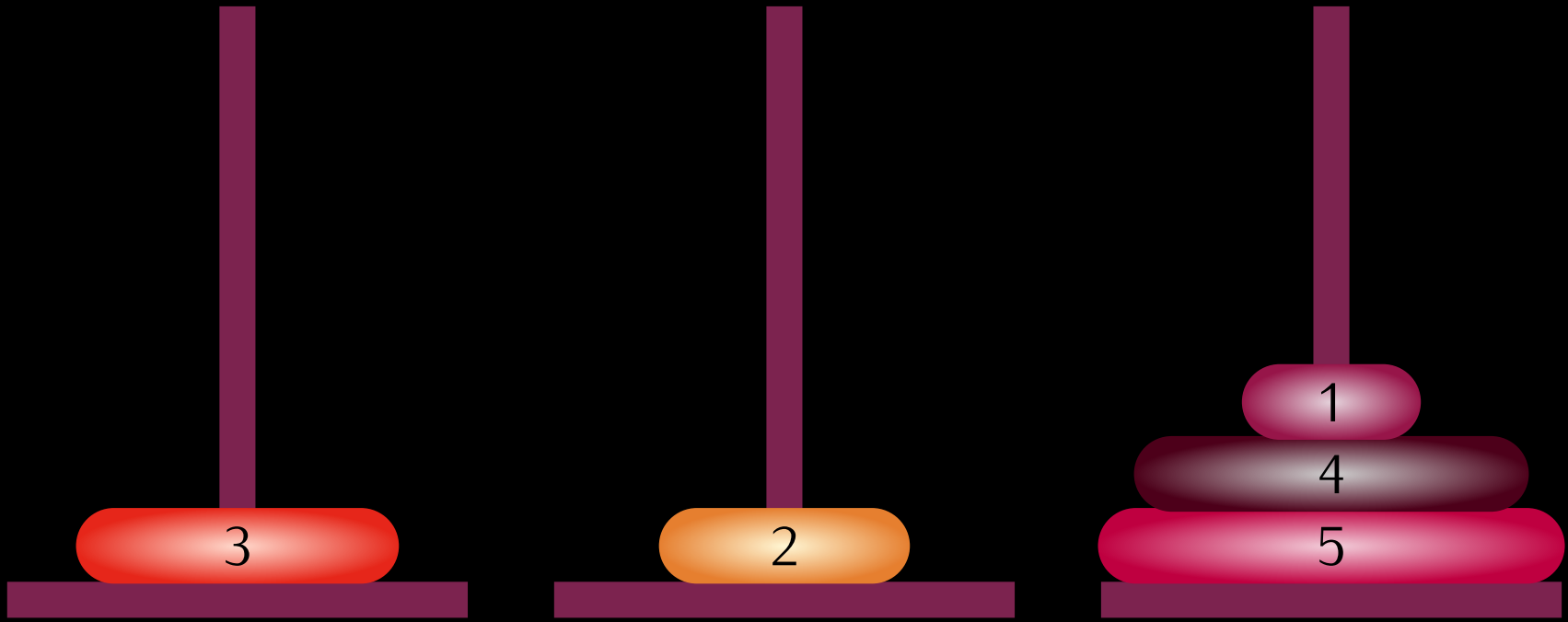
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



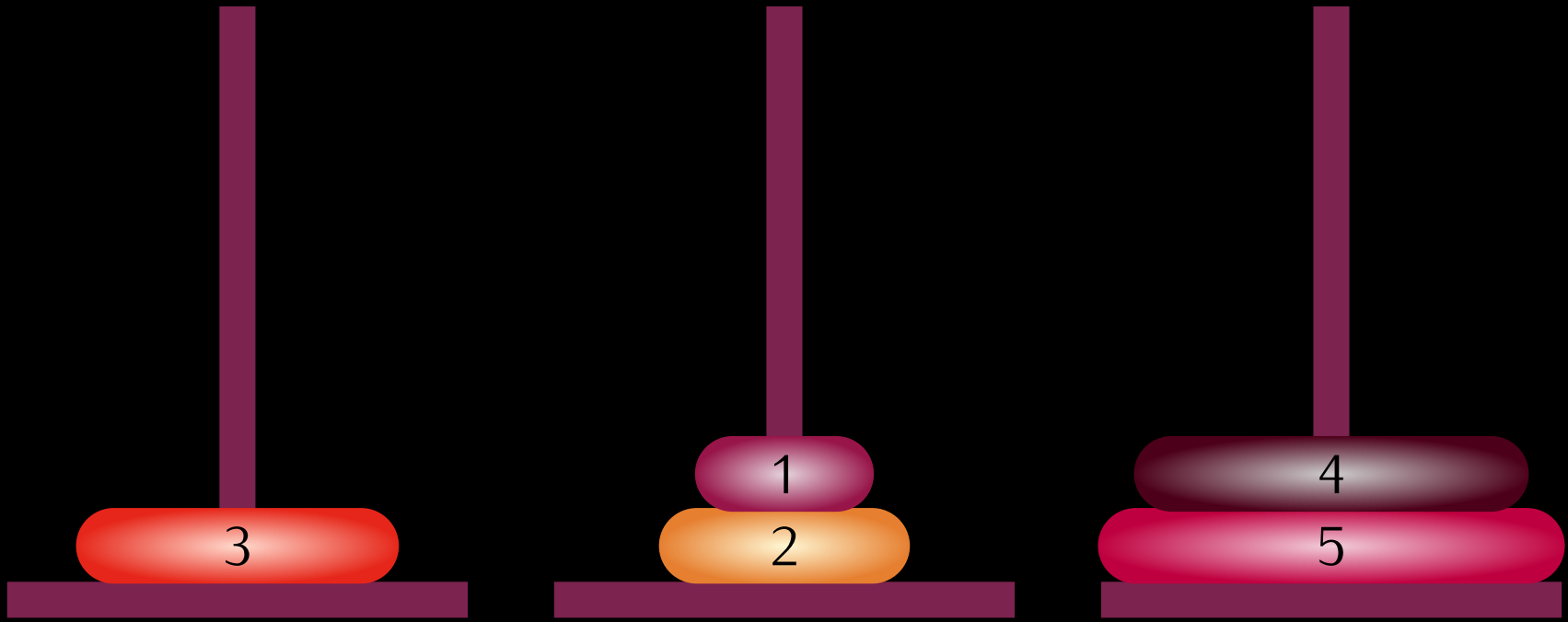
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



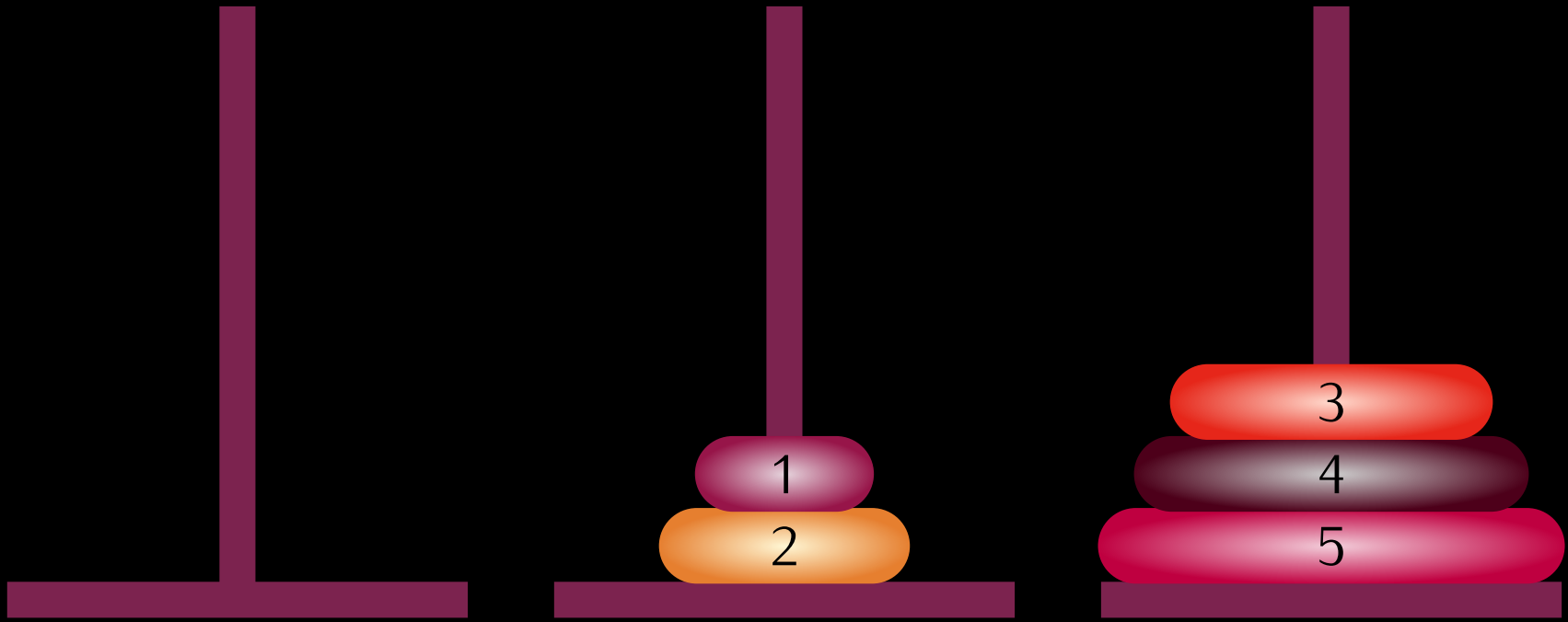
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



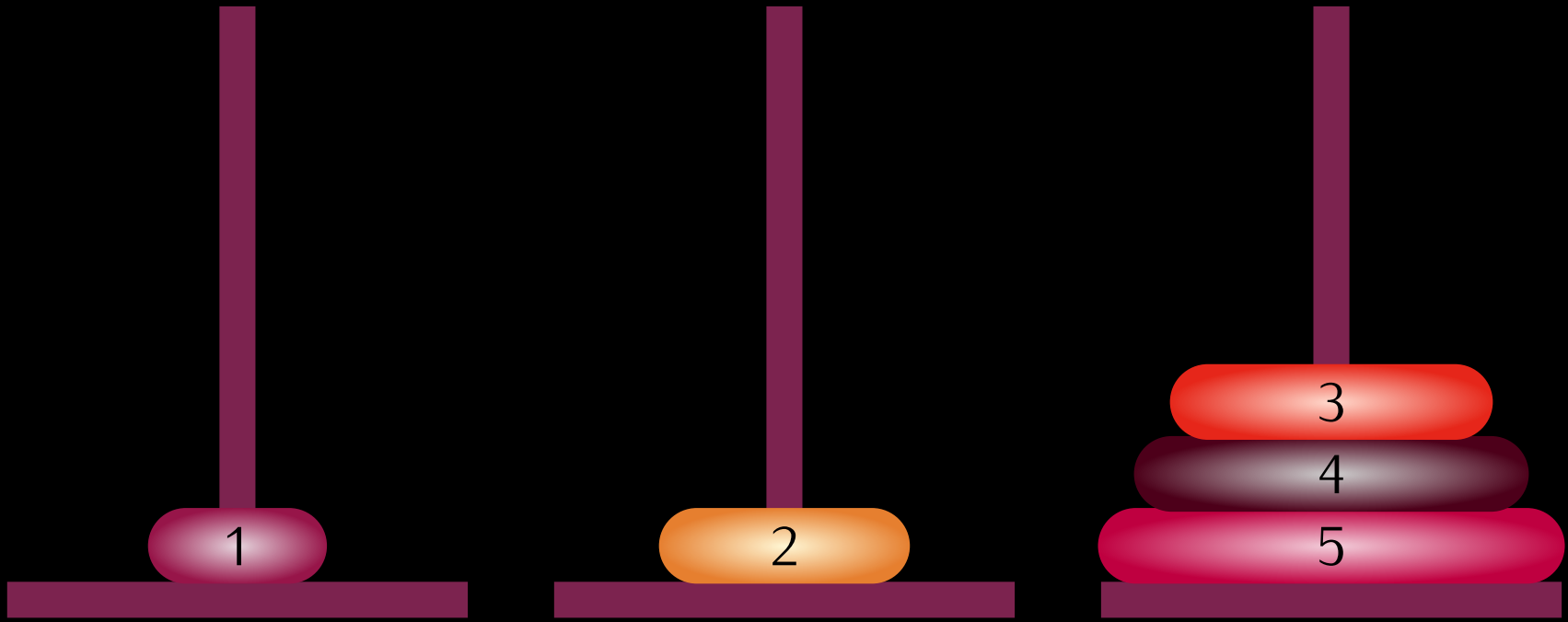
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



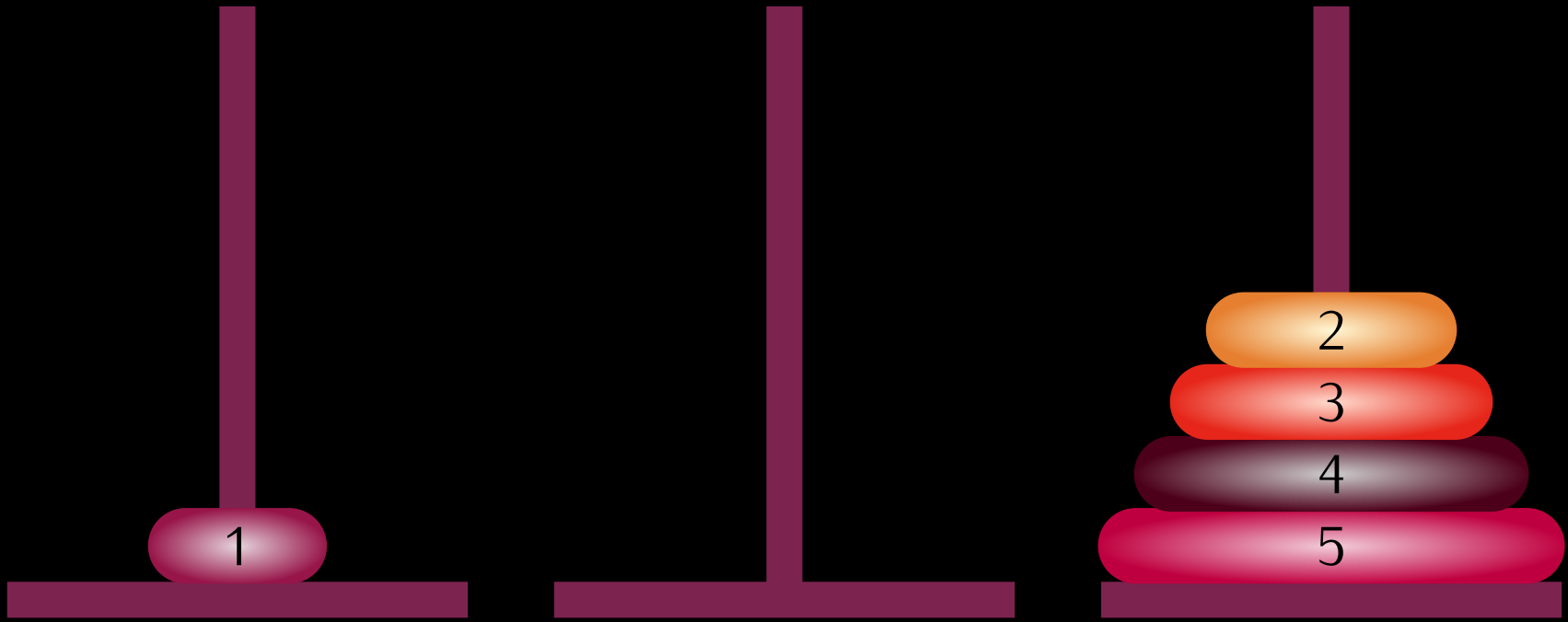
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



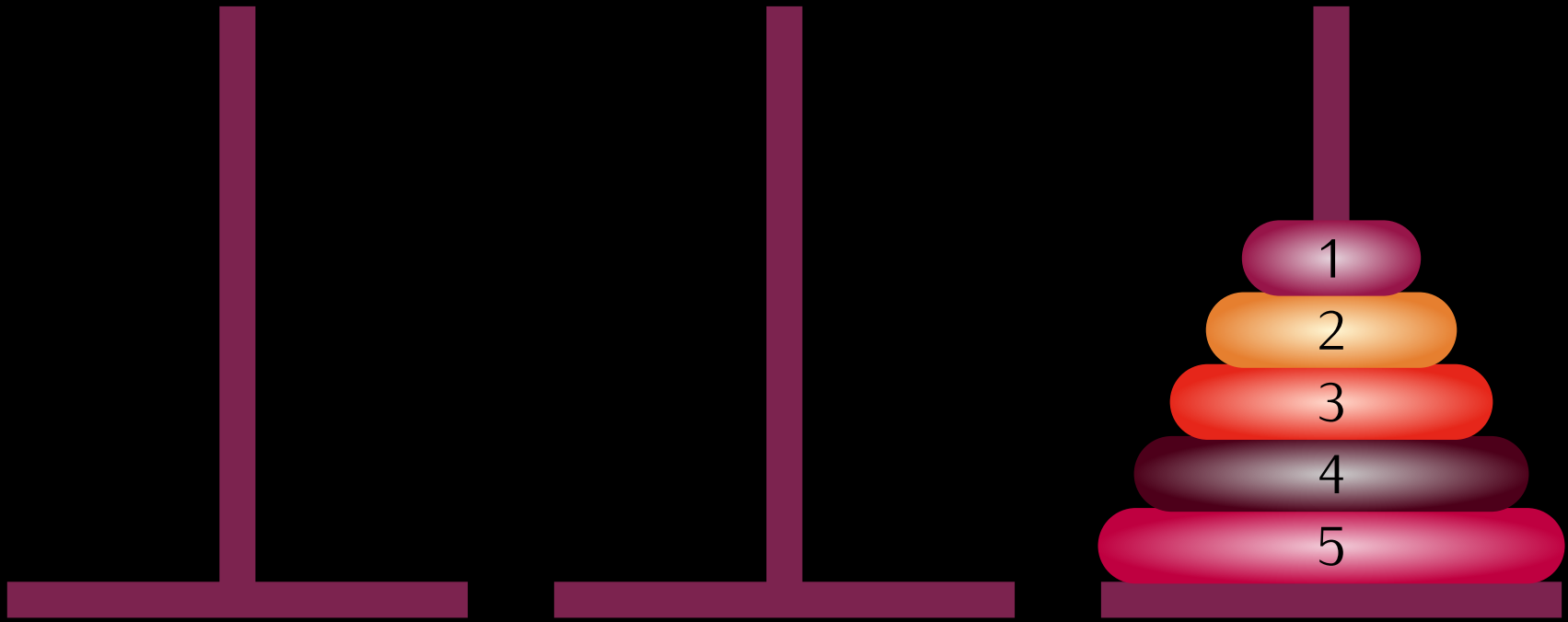
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 2 vers la tige 3.

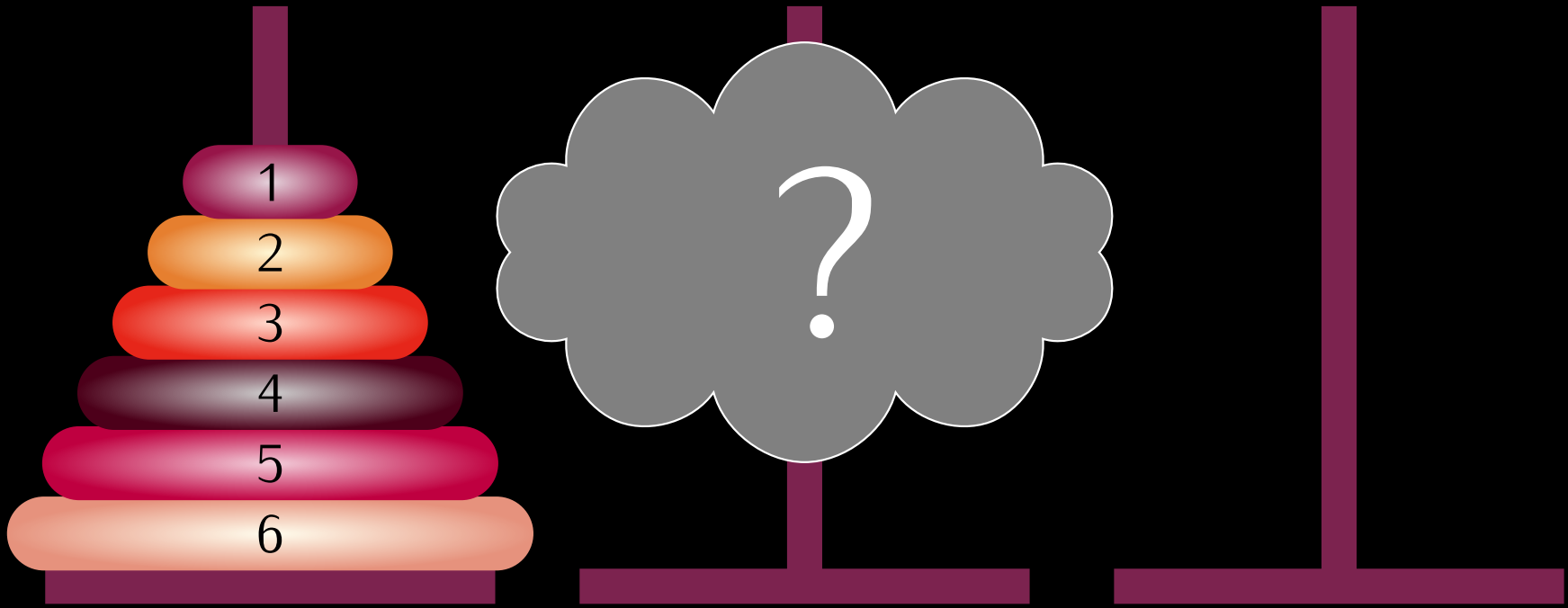
Tours de Hanoi – 5 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques

Tour de Hanoi – 6 Disques



Sommaire

Sommaire

Expérience informatique

- k actionnaires
- $N = 2^n$ étages
- RDC = 0
- Il existe un étage fatal
- RDC non fatal
- Minimiser le nombre d'essais

- k actionnaires
- $N = 2^n$ étages
- RDC = 0
- Il existe un étage fatal
- RDC non fatal
- Minimiser le nombre d'essais

- k actionnaires
- $N = 2^n$ étages
- RDC = 0
- Il existe un étage fatal
- RDC non fatal
- Minimiser le nombre d'essais

- k actionnaires
- $N = 2^n$ étages
- RDC = 0
- Il existe un étage fatal
- RDC non fatal
- Minimiser le nombre d'essais

- k actionnaires
- $N = 2^n$ étages
- RDC = 0
- Il existe un étage fatal
- RDC non fatal
- Minimiser le nombre d'essais

- k actionnaires
- $N = 2^n$ étages
- RDC = 0
- Il existe un étage fatal
- RDC non fatal
- Minimiser le nombre d'essais

- Première idée : on commence au rez-de-chaussée et on progresse d'un étage.
- Combien d'essais au pire ?
- En moyenne

- Première idée : on commence au rez-de-chaussée et on progresse d'un étage.
- Combien d'essais au pire ?
- En moyenne

- Première idée : on commence au rez-de-chaussée et on progresse d'un étage.
- Combien d'essais au pire ?
- En moyenne

Sommaire

孫子
(544–496 av. J.-C.)

Le commandement du grand nombre est le même pour le petit nombre, ce n'est qu'une question de division en groupes.

in « L'art de la guerre » 孙子兵法 de Sun Zi (VI^e siècle avant JC)

孫子
(544–496 av. J.-C.)

Le commandement du grand nombre est le même pour le petit nombre, ce n'est qu'une question de division en groupes.

in « L'art de la guerre » 孙子兵法 *de Sun Zi (VI^e siècle avant JC)*

```
inf ← Étage inférieur
sup ← Étage supérieur
milieu ← (inf + sup)/2
Si estFatal(milieu) Alors
    ChercherEntre(inf,milieu)
Sinon
    ChercherEntre(milieu + 1, sup)
FinSi
```

```
Fonction ChercherEntre(inf, sup : Entiers) : Entier
{ pré-condition: il existe au moins un étage fatal entre inf et sup }
{ invariant: le plus petit étage fatal est entre inf et sup }
{ post-condition: la valeur retrournée est le plus petit étage fatal }
Si sup == inf Alors
    Retourner sup
Sinon
    milieu ← (inf + sup)/2
    Si estFatal(milieu) Alors
        ChercherEntre(inf,milieu)
    Sinon
        ChercherEntre(milieu + 1, sup)
    FinSi
FinSi
```

- terminaison
- correction
- complexité

- terminaison
- correction
- complexité

- terminaison
- correction
- complexité

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$\{\inf\}$

- complexité

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - Tableau : $\mathcal{O}(n)$

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - tableau statique
  - n ?

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - tableau statique
  - n ?

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - tableau statique
  - n ?

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - tableau statique
  - n ? $\log_2(N) + 1$

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - tableau statique
  - n ? $\log_2(N) + 1$

- terminaison

$$\ell_i = \frac{N}{2^i} = \frac{2^n}{2^i} = 2^{n-i}$$

- correction

$$\{\text{inf}\}$$

- complexité
  - Liste chaînée
  - tableau statique
  - n ? $\log_2(N) + 1$

```java
public static int binarySearch(int[] a, int key) {
        int low  = 0;
        int high = a.length - 1;

        while (low <= high) {
            int mid    = (low + high) / 2;
            int midVal = a[mid];

            if (midVal < key)
                low = mid + 1
            else if (midVal > key)
                high = mid - 1;
            else
                return mid; // key found
        }
        return -(low + 1);  // key not found.
    }
```

$$2^{63} - 1 + 1 = -2^{63}$$

```
int mid = low + ((high - low) / 2);
```

$$2^{63} - 1 + 1 = -2^{63}$$

```
int mid = low + ((high - low) / 2);
```

$$2^{63} - 1 + 1 = -2^{63}$$

```
int mid = low + ((high - low) / 2);
```

Sommaire

- Que se passe-t-il si nous avons moins de $n = \log_2(N)$ actionnaires pour tester ?
- Que se passe-t-il si nous n'avons que 2 actionnaires et que n est pair ?
- Que se passe-t-il quand N n'est pas une puissance de 2 ?

- Que se passe-t-il si nous avons moins de $n = \log_2(N)$ actionnaires pour tester ?
- Que se passe-t-il si nous n'avons que 2 actionnaires et que n est pair ?
- Que se passe-t-il quand N n'est pas une puissance de 2 ?

- Que se passe-t-il si nous avons moins de $n = \log_2(N)$ actionnaires pour tester ?
- Que se passe-t-il si nous n'avons que 2 actionnaires et que n est pair ?
- Que se passe-t-il quand N n'est pas une puissance de 2 ?

Trichotomie ?

Trichotomie manuelle

$$2\log_3(N) = 2\frac{\ln(N)}{\ln(2)} \times \frac{\ln(2)}{\ln(3)} \approx 1,3\log_2(N)$$

Trichotomie ?

Trichotomie manuelle

$$2\log_3(N) = 2\frac{\ln(N)}{\ln(2)} \times \frac{\ln(2)}{\ln(3)} \approx 1,3\log_2(N)$$

Trichotomie ?

Trichotomie manuelle

$$2\log_3(N) = 2\frac{\ln(N)}{\ln(2)} \times \frac{\ln(2)}{\ln(3)} \approx 1,3\log_2(N)$$

Sommaire

La multiplication de deux entiers de l'école primaire : complexité ?

- Additions de deux entiers de n chiffres ?
- Multiplication d'un nombre de n chiffres par un nombre de 1 chiffre ?
- Multiplication de deux nombres de n chiffres ?
- Diviser pour régner ?

La multiplication de deux entiers de l'école primaire : complexité ?

- Additions de deux entiers de n chiffres ?
- Multiplication d'un nombre de n chiffres par un nombre de 1 chiffre ?
- Multiplication de deux nombres de n chiffres ?
- Diviser pour régner ?

La multiplication de deux entiers de l'école primaire : complexité ?

- Additions de deux entiers de n chiffres ?
- Multiplication d'un nombre de n chiffres par un nombre de 1 chiffre ?
- Multiplication de deux nombres de n chiffres ?
- Diviser pour régner ?

La multiplication de deux entiers de l'école primaire : complexité ?

- Additions de deux entiers de n chiffres ?
- Multiplication d'un nombre de n chiffres par un nombre de 1 chiffre ?
- Multiplication de deux nombres de n chiffres ?
- Diviser pour régner ?

Fonction MUL(x:*entier* ,y: entier):*entier*

Si n==1 Alors

  Retourner x.y

Sinon

  $m \leftarrow \lfloor n/2 \rfloor$

  $x_1 \leftarrow \lfloor x/10^m \rfloor$

  $x_2 \leftarrow x \bmod 10^m$

  $y_1 \leftarrow \lfloor y/10^m \rfloor$

  $y_2 \leftarrow y \bmod 10^m$

  $a \leftarrow$ MUL(x_1,y_1,m)

  $b \leftarrow$ MUL(x_2,y_1,m)

  $c \leftarrow$ MUL(x_1,y_2,m)

  $d \leftarrow$ MUL(x_2,y_2,m)

  Retourner $10^{2m}a + 10^m(b+c) + d$

FinSi

$$T(n) = 4T(\lfloor n/2 \rfloor) + \lambda n \qquad T(1) = 1$$

$$n = 2^k$$

$$x_k = 4x_{k-1} + \lambda 2^k \qquad x_0 = 1$$

$$x_k = 4(4x_{k-2} + \lambda' 2^{k-1}) + \lambda 2^k = 4^k x_0 + \sum_{i=1}^{k} \Lambda_k 2^k = 4^k + k\Lambda_k 2^k$$

$$T(n) \sim n^2$$

$$T(n) = 4T(\lfloor n/2 \rfloor) + \lambda n \qquad T(1) = 1$$

$$n = 2^k$$

$$x_k = 4x_{k-1} + \lambda 2^k \qquad x_0 = 1$$

$$x_k = 4(4x_{k-2} + \lambda' 2^{k-1}) + \lambda 2^k = 4^k x_0 + \sum_{i=1}^{k} \Lambda_k 2^k = 4^k + k\Lambda_k 2^k$$

$$T(n) \sim n^2$$

$$T(n) = 4T(\lfloor n/2 \rfloor) + \lambda n \qquad T(1) = 1$$

$$n = 2^k$$

$$x_k = 4x_{k-1} + \lambda 2^k \qquad x_0 = 1$$

$$x_k = 4(4x_{k-2} + \lambda' 2^{k-1}) + \lambda 2^k = 4^k x_0 + \sum_{i=1}^{k} \Lambda_k 2^k = 4^k + k\Lambda_k 2^k$$

$$T(n) \sim n^2$$

$$T(n) = 4T(\lfloor n/2 \rfloor) + \lambda n \qquad T(1) = 1$$

$$n = 2^k$$

$$x_k = 4x_{k-1} + \lambda 2^k \qquad x_0 = 1$$

$$x_k = 4(4x_{k-2} + \lambda' 2^{k-1}) + \lambda 2^k = 4^k x_0 + \sum_{i=1}^{k} \Lambda_k 2^k = 4^k + k\Lambda_k 2^k$$

$$T(n) \sim n^2$$

$$T(n) = 4T(\lfloor n/2 \rfloor) + \lambda n \qquad T(1) = 1$$

$$n = 2^k$$

$$x_k = 4x_{k-1} + \lambda 2^k \qquad x_0 = 1$$

$$x_k = 4(4x_{k-2} + \lambda' 2^{k-1}) + \lambda 2^k = 4^k x_0 + \sum_{i=1}^{k} \Lambda_k 2^k = 4^k + k\Lambda_k 2^k$$

$$T(n) \sim n^2$$

Sommaire

$x^2 - 2 = 0$ précision 2^{-10}

- solution $\longrightarrow$ étage
- tableau $\longrightarrow$ immeuble
- précision $\longrightarrow$ nombre d'étages

$x^2 - 2 = 0$ précision 2^{-10}

- solution $\longrightarrow$ étage
- tableau $\longrightarrow$ immeuble
- précision $\longrightarrow$ nombre d'étages
- `estFatal` $\longrightarrow$ `x` $\mapsto$ `x * x <= 2`

$x^2 - 2 = 0$ précision 2^{-10}

- solution $\longrightarrow$ étage
- tableau $\longrightarrow$ immeuble
- précision $\longrightarrow$ nombre d'étages
- `estFatal` $\longrightarrow$ `x` $\mapsto$ `x * x <= 2`

$x^2 - 2 = 0$ précision 2^{-10}

- solution $\longrightarrow$ étage
- tableau $\longrightarrow$ immeuble
- précision $\longrightarrow$ nombre d'étages
- `estFatal` $\longrightarrow$ `x` $\mapsto$ `x * x <= 2`

$x^2 - 2 = 0$ précision 2^{-10}

- solution $\longrightarrow$ étage
- tableau $\longrightarrow$ immeuble
- précision $\longrightarrow$ nombre d'étages
- `estFatal` $\longrightarrow$ `x` $\mapsto$ `x * x <= 2`

1	$1+2^{-10}$	$1+2\times 2^{-10}$	$1+3\times 2^{-10}$	$\cdots$	$1+2^{10}\times 2^{-10}$

```
def racineDicho(prec):
  cpt = 0
  inf = 1
  sup = 2
  while (sup - inf > prec):
      m = inf + (sup - inf) / 2
      cpt += 1
      if m*m <= 2:
          inf = m
      else:
          sup = m
  return sup,cpt
```

```
In [1]: racineDicho(2**(-10))
Out[1]: (1.4150390625, 10)

In [2]: racineDicho(2**(-15))
Out[2]: (1.414215087890625, 15)

In [3]: racineDicho(2**(-20))
Out[3]: (1.4142141342163086, 20)

In [4]: racineDicho(2**(-30))
Out[4]: (1.4142135623842478, 30)

In [5]: racineDicho(2**(-50))
Out[5]: (1.4142135623730958, 50)
```

```
In [6]: sqrt(2)
Out[6]: 1.4142135623730951
```

```
In [1]: racineDicho(2**(-10))
Out[1]: (1.4150390625, 10)

In [2]: racineDicho(2**(-15))
Out[2]: (1.414215087890625, 15)

In [3]: racineDicho(2**(-20))
Out[3]: (1.4142141342163086, 20)

In [4]: racineDicho(2**(-30))
Out[4]: (1.4142135623842478, 30)

In [5]: racineDicho(2**(-50))
Out[5]: (1.4142135623730958, 50)
```

```
In [6]: sqrt(2)
Out[6]: 1.4142135623730951
```

Sommaire

Le problème

On dispose d'une liste L de N nombres. Déterminez les triplets d'éléments de L tous distincts dont la somme est nulle

Sommaire

```python
def trois_sommes(xs):
    N = len(xs)
    cpt = 0
    for i in range(N):
        for j in range(i + 1, N):
            for k in range(j + 1, N):
                if xs[i] + xs[j] + xs[k] == 0:
                    cpt += 1
    return cpt
```

```
In [1]: %timeit trois_sommes([randint(-10000,10000) for k in
        ↪  range(100)])
10 loops, best of 3: 27.5 ms per loop

In [2]: %timeit trois_sommes([randint(-10000,10000) for k in
        ↪  range(200)])
1 loops, best of 3: 216 ms per loop

In [3]: %timeit trois_sommes([randint(-10000,10000) for k in
        ↪  range(400)])
1 loops, best of 3: 1.82 s per loop
```

```python
from time import perf_counter
from math import log

def temps(xs):
    debut = perf_counter()
    trois_sommes(xs)
    return perf_counter() - debut

t = [temps(range(100 * 2**k)) for k in range(4)]

ratio = [t[k + 1] / t[k] for k in range(3)]

def logD(x):
    return log(x)/log(2)

logratio = map(logD,ratio)
```

```
In [4]: ratio
Out[4]: [7.523860206447286, 9.118789882406599, 8.5098312160934]

In [5]: list(logratio)
Out[5]: [2.940628541715559, 3.133284732580891, 3.128693841844642]
```

aN^3 *a ?*

```
In [6]: temps(range(400))
Out[6]: 4.0054843200001415
```

$4,00 = a \times 400^3$

$a \approx 6,25 \times 10^{-8}$

aN^3 a ?

```
In [6]: temps(range(400))
Out[6]: 4.005484320001415
```

$4,00 = a \times 400^3$

$a \approx 6,25 \times 10^{-8}$

aN^3 a ?

```
In [6]: temps(range(400))
Out[6]: 4.005484320001415
```

$4,00 = a \times 400^3$

$a \approx 6,25 \times 10^{-8}$

aN^3 a ?

```
In [6]: temps(range(400))
Out[6]: 4.005484320001415
```

$4,00 = a \times 400^3$

$a \approx 6,25 \times 10^{-8}$

aN^3 a ?

```
In [6]: temps(range(400))
Out[6]: 4.005484320001415
```

$4,00 = a \times 400^3$
$a \approx 6,25 \times 10^{-8}$

$N = 1000$ $6,25 \times 10^{-8} \times 10^{9} = 62,5$

```
In [7]: temps(range(1000))
Out[7]: 68.54615448799996
```

$N = 1000\ 6,25 \times 10^{-8} \times 10^{9} = 62,5$

```
In [7]: temps(range(1000))
Out[7]: 68.54615448799996
```

$N = 1000\ 6,25 \times 10^{-8} \times 10^{9} = 62,5$

```
In [7]: temps(range(1000))
Out[7]: 68.54615448799996
```

```c
#include <stdio.h>
#include <time.h>
typedef int Liste[13000];

int trois_sommes(int N)
{
  int cpt = 0;
  Liste liste;

  for ( int k = 0; k < N; k++)
    {
      liste[k] = k - (N / 2);
    }

  for (int i = 0; i < N; i++)
    {
      for (int j = i + 1; j < N; j++)
{
  for (int k = j + 1; k < N; k++)
    {
      if (liste[i] + liste[j] + liste[k] == 0) {cpt++;}
    }
}
     }
  return cpt;
}
```

```c
void chrono(int N)
{
  clock_t temps_initial, temps_final;
  float temps_cpu;

  temps_initial = clock();
  trois_sommes(N);
  temps_final   = clock();
  temps_cpu = ((double) temps_final - temps_initial) / CLOCKS_PER_SEC;
  printf("Temps en sec pour %d : %f\n",N, temps_cpu);

}
int main(void)
{
  chrono(100);
  chrono(200);
  chrono(400);
  chrono(800);
  chrono(1600);
  chrono(3200);
  chrono(6400);
  chrono(12800);

  return 1;
}
```

```
$ gcc  -std=c99 -Wall -Wextra  -Werror -pedantic -O4 -o somm3 Trois_Sommes.c
$ ./somm3
Temps en sec pour 100 : 0.00000
Temps en sec pour 200 : 0.000000
Temps en sec pour 400 : 0.020000
Temps en sec pour 800 : 0.090000
Temps en sec pour 1600 : 0.720000
Temps en sec pour 3200 : 5.760000
Temps en sec pour 6400 : 45.619999
Temps en sec pour 12800 : 360.839996
```

$45,61 = a \times 6400^3$ d'où $a \approx 1,74 \times 10^{-10}$

$360,84 = a \times 12800^3$ d'où $a \approx 1.72 \times 10^{-10}$

Rappel Python : $a \approx 6,25 \times 10^{-8}$

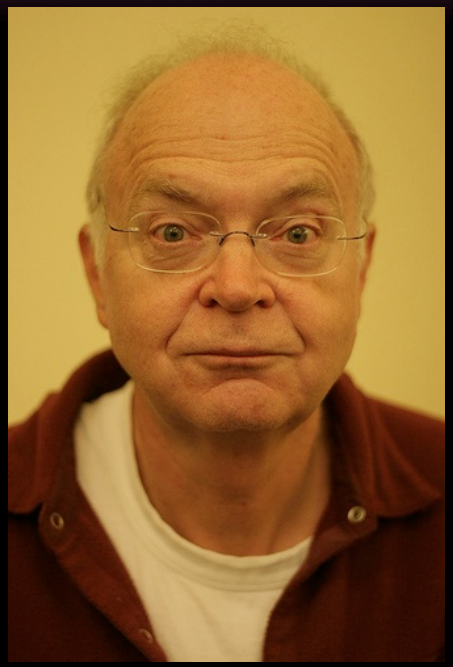
ORDRE DE CROISSANCE

$45,61 = a \times 6400^3$ d'où $a \approx 1,74 \times 10^{-10}$
$360,84 = a \times 12800^3$ d'où $a \approx 1.72 \times 10^{-10}$
Rappel Python : $a \approx 6,25 \times 10^{-8}$
ORDRE DE CROISSANCE

$45,61 = a \times 6400^3$ d'où $a \approx 1,74 \times 10^{-10}$
$360,84 = a \times 12800^3$ d'où $a \approx 1.72 \times 10^{-10}$
Rappel Python : $a \approx 6,25 \times 10^{-8}$
ORDRE DE CROISSANCE

$45,61 = a \times 6400^3$ d'où $a \approx 1,74 \times 10^{-10}$
$360,84 = a \times 12800^3$ d'où $a \approx 1.72 \times 10^{-10}$
Rappel Python : $a \approx 6,25 \times 10^{-8}$
ORDRE D*E* CROISSANCE

Sommaire



D. E. Knuth
Né en 1938

coût \ n	100	1000	10^6	10^9
$\log_2(n)$	≈ 7	≈ 10	≈ 20	≈ 30
$n \log_2(n)$	≈ 665	$\approx 10\,000$	$\approx 2 \cdot 10^7$	$\approx 3 \cdot 10^{10}$
n^2	10^4	10^6	10^{12}	10^{18}
n^3	10^6	10^9	10^{18}	10^{27}
2^n	$\approx 10^{30}$	$> 10^{300}$	$> 10^{10^5}$	$> 10^{10^8}$

Gardez en tête que l'âge de l'Univers est environ de 10^{18} secondes...

coût \ n	100	1000	10^6	10^9
$\log_2(n)$	≈ 7	≈ 10	≈ 20	≈ 30
$n \log_2(n)$	≈ 665	$\approx 10\,000$	$\approx 2 \cdot 10^7$	$\approx 3 \cdot 10^{10}$
n^2	10^4	10^6	10^{12}	10^{18}
n^3	10^6	10^9	10^{18}	10^{27}
2^n	$\approx 10^{30}$	$> 10^{300}$	$> 10^{10^5}$	$> 10^{10^8}$

Gardez en tête que l'âge de l'Univers est environ de 10^{18} secondes...

```python
def trois_sommes(xs):
    N = len(xs)
    cpt = 0
    for i in range(N):
        xi = xs[i]
        for j in range(i + 1, N):
            sij = xi + xs[j]
            for k in range(j + 1, N):
                if sij + xs[k] == 0:
                    cpt += 1
    return cpt
```

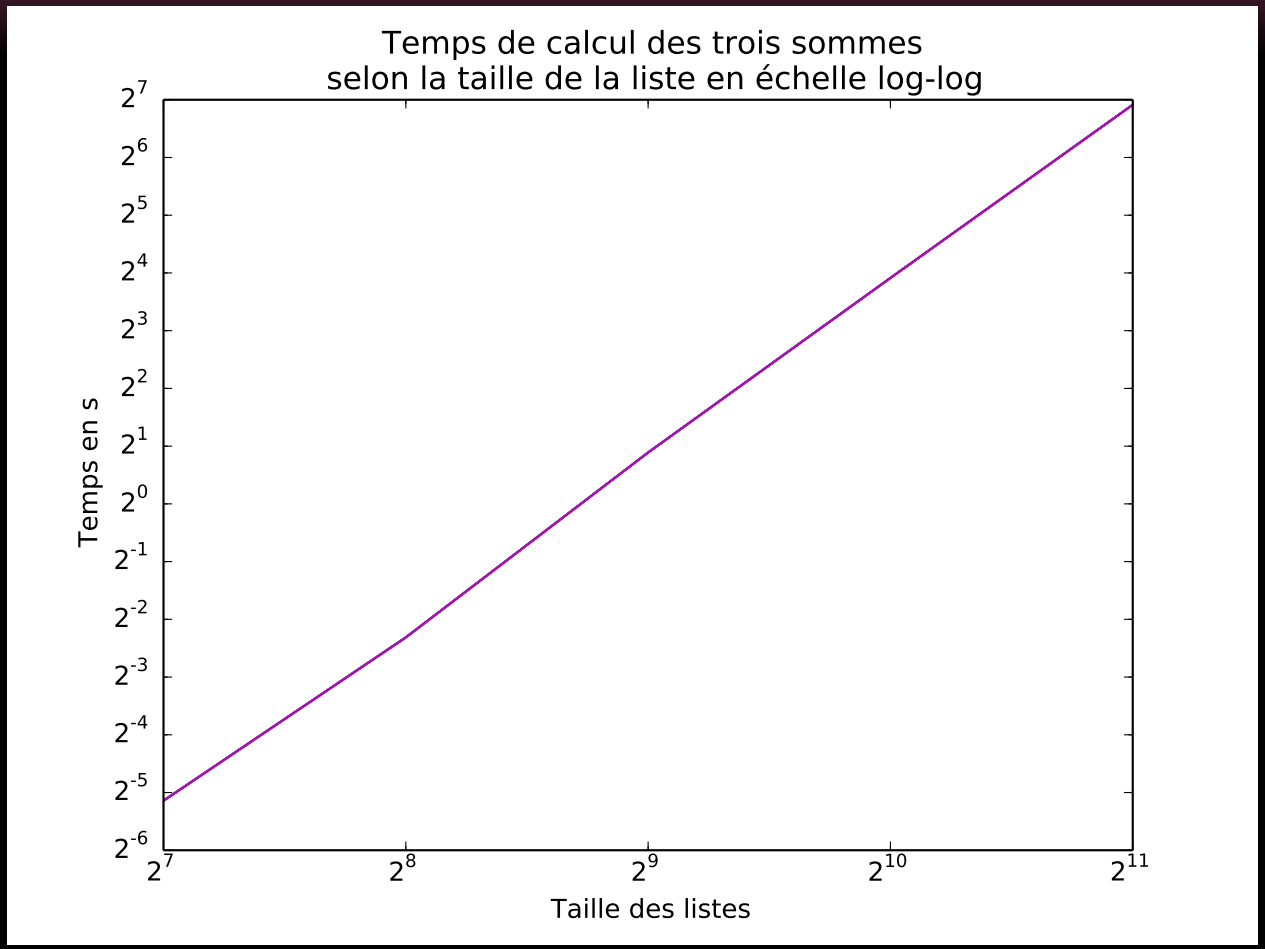
```
In [28]:  xs = list(range(-50,51))
In [29]: %timeit trois_sommes(xs)
100 loops, best of 3: 12.9 ms per loop

In [30]:  xs = list(range(-100,101))
In [31]: %timeit trois_sommes(xs)
10 loops, best of 3: 94.4 ms per loop

In [32]:  xs = list(range(-200,201))
In [33]: %timeit trois_sommes(xs)
1 loops, best of 3: 851 ms per loop

In [34]:  xs = list(range(-400,401))
In [35]: %timeit trois_sommes(xs)
1 loops, best of 3: 7.04 s per loop
```

```
tailles = [2**k for k in range(7,11)]*
listes = [list(range(- N//2, N//2 + 1)) for N in tailles]
t = [temps(xs) for xs in listes]
for taille in tailles:
    plt.loglog(tailles, t, basex = 2, basey = 2)
plt.title('Temps de calcul selon la taille de la liste en echelle
        ↪ log-log')
plt.xlabel('Taille des listes')
plt.ylabel('Temps en s')
```


Temps de calcul des trois sommes
selon la taille de la liste en échelle log-log
Temps en s
2^{7}
2^{6}
2^{5}
2^{4}
2^{3}
2^{2}
2^{1}
2^{0}
2^{-1}
2^{-2}
2^{-3}
2^{-4}
2^{-5}
2^{-6}
2^{7}
2^{8}
2^{9}
2^{10}
2^{11}
Taille des listes

OPÉRATION	FRÉQUENCE
Déclaration de la fonction et du paramètre (l. 1)	2
Déclaration de N, cpt et i (l. 2, 3 et 4)	3
Affectation de N, cpt et i (l. 2, 3 et 4)	3
Déclaration de xi (l. 5)	N
Affectation de xi (l. 5)	N
Accès à xs[i] (l. 5)	N
Déclaration de j (l.6)	N
Calcul de l'incrément de i (l. 6)	N
Affectation de j (l.6)	N
Déclaration de sij (l. 7)	S_1
Affectation de sij (l. 7)	S_1

OPÉRATION	FRÉQUENCE
Accès à xs[j] (l.7)	S_1
Somme (l.7)	S_1
Déclaration de k (l.8)	S_1
Incrément de j (l. 8)	S_1
Affectation de k (l.8)	S_1
Accès à x[k] (l.9)	S_2
Calcul de la somme (l.9)	S_2
Comparaison à 0 (l.9)	S_2
Incrément de cpt (l.9)	entre 0 et S_2
Affectation de la valeur de retour (l.11)	1

$$S_1 = \sum_{i=0}^{N-1} N - (i+1) = \sum_{i'=0}^{N-1} i' = \frac{N(N-1)}{2}$$

$$
\begin{aligned}
S_2 &= \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} N-(j+1) \\
&= \sum_{i=0}^{N-1} \sum_{j'=0}^{N-(i+2)} j' \\
&= \sum_{i=0}^{N-2} \frac{(N-(i+2))(N-(i+1))}{2} \\
&= \sum_{i'=1}^{N-2} \frac{i'(i'+1)}{2} \\
&= \frac{1}{2}\left(\sum_{i=1}^{N-2} i^2 + \sum_{i=1}^{N-2} i\right) \\
&= \frac{1}{2}\left(\frac{(N-2)(2N-3)(N-1)}{6} + \frac{(N-2)(N-1)}{2}\right) \\
&= \frac{N(N-1)(N-2)}{6}
\end{aligned}
$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

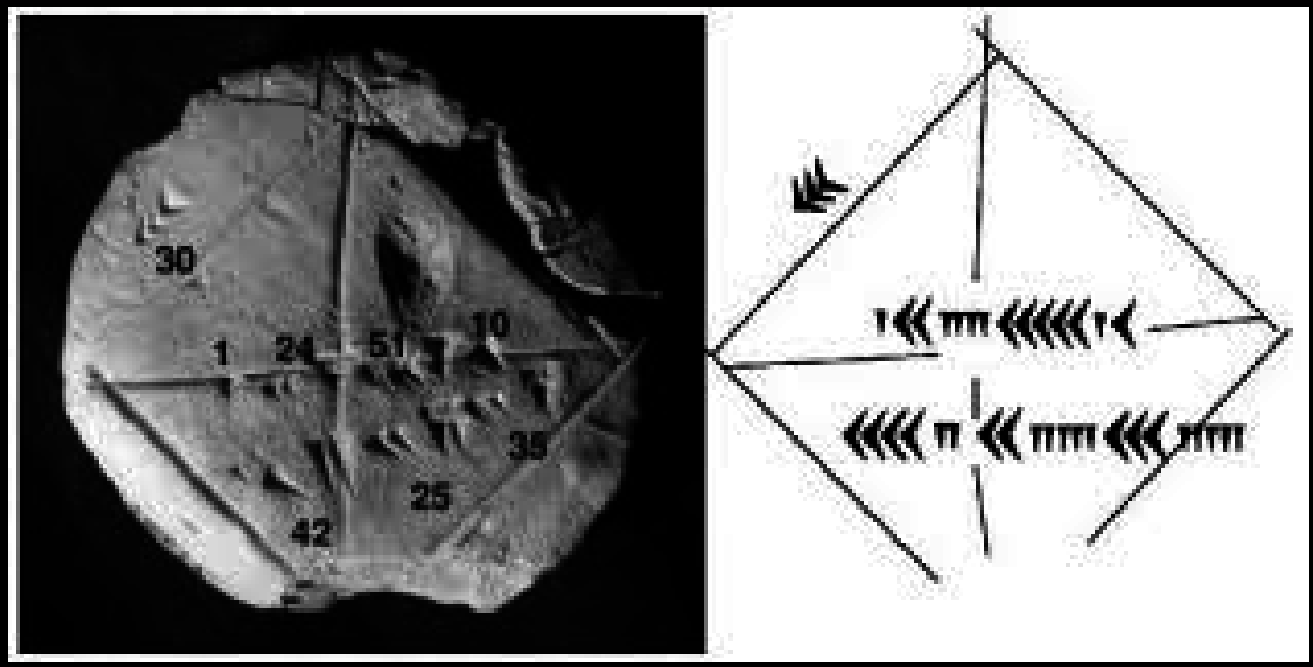
Notons

- a le temps constant d'affectation
- d le temps constant de déclaration
- x le temps constant d'accès à une cellule
- s le temps constant d'une somme
- c le temps constant d'une comparaison.

$$\tau(N) \leqslant (2d+3d+3a+a)+(d+a+x+d+s+a)N+(d+a+x+s+d+s+a)S_1+(x+s+c+s)S_2$$

$$\tau(N) = O(N^3)$$

Sommaire



Ordinateur Babylonien

Si x_n est une approximation strictement positive par défaut de $\sqrt{a}$, alors a/x_n est une approximation par excès de $\sqrt{a}$ et vice-versa.

La moyenne arithmétique de ces deux approximations est $\frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$ et constitue une meilleure approximation que les deux précédentes.

On peut montrer c'est une approximation par excès (en développant $\left(x_n - \sqrt{a}\right)^2$ par exemple).

Si x_n est une approximation strictement positive par défaut de $\sqrt{a}$, alors a/x_n est une approximation par excès de $\sqrt{a}$ et vice-versa.

La moyenne arithmétique de ces deux approximations est $\frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$ et constitue une meilleure approximation que les deux précédentes.

On peut montrer c'est une approximation par excès (en développant $\left(x_n - \sqrt{a}\right)^2$ par exemple).

Si x_n est une approximation strictement positive par défaut de $\sqrt{a}$, alors a/x_n est une approximation par excès de $\sqrt{a}$ et vice-versa.

La moyenne arithmétique de ces deux approximations est $\frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$ et constitue une meilleure approximation que les deux précédentes.

On peut montrer c'est une approximation par excès (en développant $\left(x_n - \sqrt{a}\right)^2$ par exemple).

```python
def heron_rec(a,fo,n):
        if n == 0:
                return fo
        return heron_rec(a,(fo + a / fo) / 2, n - 1)

def heron_it(a,fo,n):
        app = fo
        for k in range(n):
                app = (app + a / app) / 2
        return app
```

```
In [12]: heron_rec(2,1,6)
Out[12]: 1.414213562373095

In [13]: heron_it(2,1,6)
Out[13]: 1.414213562373095

In [14]: from math import sqrt

In [15]: sqrt(2)
Out[15]: 1.4142135623730951
```

$r_{n+1} = f(r_n) = \frac{r_n + \frac{2}{r_n}}{2}$ et $r_0 = 1$

Théorème 1 (Théorème de la limite monotone)

Toute suite croissante majorée (ou décroissante minorée) converge.

$r_{n+1} = f(r_n) = \frac{r_n + \frac{2}{r_n}}{2}$ et $r_0 = 1$

Théorème 1 (Théorème de la limite monotone)

Toute suite croissante majorée (ou décroissante minorée) converge.

Théorème 2 (Le théorème du point fixe)

Soit I un intervalle fermé de $\mathbb{R}$, soit f une fonction de I vers I et soit (r_n) une suite d'éléments de I telle que $r_{n+1} = f(r_n)$ pour tout entier naturel n. Si (r_n) est convergente, alors sa limite est un point fixe de f appartenant à I.

$r_{n+1} = f(r_n) = \frac{r_n + \frac{2}{r_n}}{2}$ et $r_0 = 1$

Théorème 1 (Théorème de la limite monotone)

Toute suite croissante majorée (ou décroissante minorée) converge.

Théorème 2 (Un théorème light du point fixe)

Soit I un intervalle fermé de $\mathbb{R}$, soit f une fonction de I vers I et soit (r_n) une suite d'éléments de I telle que $r_{n+1} = f(r_n)$ pour tout entier naturel n. SI (r_n) est convergente ALORS sa limite est UN point fixe de f appartenant à I.

$r_{n+1} = f(r_n) = \frac{r_n + \frac{2}{r_n}}{2}$ et $r_0 = 1$

Théorème 1 (Théorème de la limite monotone)

Toute suite croissante majorée (ou décroissante minorée) converge.

Théorème 2 (Un théorème light du point fixe)

Soit I un intervalle fermé de $\mathbb{R}$, soit f une fonction de I vers I et soit (r_n) une suite d'éléments de I telle que $r_{n+1} = f(r_n)$ pour tout entier naturel n. SI (r_n) est convergente ALORS sa limite est UN point fixe de f appartenant à I.

1. pour tout entier naturel non nul n, on a $r_n \geqslant \sqrt{2}$;
2. la suite est décroissante ;
3. $\sqrt{2}$ est l'unique point fixe positif de f.

1. pour tout entier naturel non nul n, on a $r_n \geqslant \sqrt{2}$;
2. la suite est décroissante ;
3. $\sqrt{2}$ est l'unique point fixe positif de f.

1. pour tout entier naturel non nul n, on a $r_n \geqslant \sqrt{2}$;
2. la suite est décroissante ;
3. $\sqrt{2}$ est l'unique point fixe positif de f.

$r_n \geqslant 1$ donc

$$\left|r_{n+1} - \sqrt{2}\right| \leqslant \left|(r_n - \sqrt{2})^2\right|$$

$d_n = -\log_{10}|r_n - \sqrt{2}|$

$d_{n+1} \geqslant 2d_n$

$r_n \geqslant 1$ donc

$$\left|r_{n+1} - \sqrt{2}\right| \leqslant \left|(r_n - \sqrt{2})^2\right|$$

$d_n = -\log_{10} |r_n - \sqrt{2}|$

$d_{n+1} \geqslant 2d_n$

$r_n \geqslant 1$ donc

$$\left|r_{n+1} - \sqrt{2}\right| \leqslant \left|(r_n - \sqrt{2})^2\right|$$

$d_n = -\log_{10} |r_n - \sqrt{2}|$

$$d_{n+1} \geqslant 2d_n$$

Sommaire

```
Pour i de 1 à n Faire
    op
FinPour
```

```
Pour i de 1 à n*n Faire
    Pour j de 1 à n*3 Faire
        Pour k de 1 à n Faire
            op
        FinPour
    FinPour
FinPour
```

```
Pour i de 1 à n Faire
    Pour j de 1 à n Faire
        op
    FinPour
    Pour k de 1 à n Faire
        op
    FinPour
FinPour
```

```
Pour i de 1 à n Faire
    Pour j de 1 à n Faire
        op
    FinPour
FinPour
```

```
Pour i de 1 à n Faire
    Pour j de 1 à n Faire
        Pour k de 1 à n Faire
            op
        FinPour
    FinPour
FinPour
```

```
Pour i de 1 à n Faire
    Pour j de 1 à 90n Faire
        op
    FinPour
    Pour k de 40n à 1 Faire
        op
    FinPour
FinPour
```

```
Pour i de 1 à n Faire
    Pour j de 1 à i Faire
        op
    FinPour
FinPour
```

```
Pour i de 1 à n Faire
    Pour j de 1 à i Faire
        Pour k de 1 à j Faire
            op
        FinPour
    FinPour
FinPour
```

```
s ← 0
i ← n
TantQue i > 0 Faire
    Pour j de 0 à i − 1 Faire
        s ← s + 1
    FinPour
    i ← i // 2
FinTantQue
Retourner s
```

```
s ← 0
i ← 1
TantQue i < n Faire
    Pour j de 0 à i − 1 Faire
        s ← s + 1
    FinPour
    i ← i ∗ 2
FinTantQue
Retourner s
```

```
s ← 0
i ← 1
TantQue i < n Faire
    Pour j de 0 à n − 1 Faire
        s ← s + 1
    FinPour
    i ← i ∗ 2
FinTantQue
Retourner s
```

Voici ce qu'apprenaient les petits soviétiques pour multiplier deux entiers. Une variante de cet algorithme a été retrouvée sur le papyrus de Rhind datant de 1650 avant JC, le scribe Ahmes affirmant que cet algorithme était à l'époque vieux de 350 ans. Il a survécu en Europe occidentale jusqu'aux travaux de Fibonacci.

```
1  Fonction MULRUSSE(x:entier ,y: entier, acc:entier):entier
2  Si x==0 Alors
3  |  Retourner acc
4  Sinon
5  |  Si x est pair Alors
6  |  |  Retourner MULRUSSE(x/2,y*2,acc)
7  |  Sinon
8  |  |  Retourner MULRUSSE((x-1)/2,y*2,acc+y)
9  |  FinSi
10 FinSi
```

- Que vaut `acc` au départ ?
- Écrivez une version récursive de cet algorithme en évitant l'alternative des lignes 5 à 9.
- Écrivez une version impérative de cet algorithme.
- Prouvez la correction de cet algorithme.
- Étudiez sa complexité.

En python, `x >> 1` décale l'entier x d'un bit vers la doite et `x << 1` décale x d'un bit vers la gauche en complétant par un zéro à droite, `x& y` renvoie l'entier obtenu en faisant la conjonction logique bit à bit des représentations binaires de x et y et `~x` renvoie le complément à 1 de x. Ré-écrivez la multiplication russe en utilisant que ces opérations bit à bit (pas de division ni de multiplication).