بسم الله الرحمن الرحيم

اسأل الله ان اجدكم في احسن حال و الصحة, انتم و ذويكم

السلام عليكم و رحمه الله و بركاته

Salam, j’ espère que vous allez bien ainsi que votre famille

La récursivité

Rappel et exercices

- Principe
- Implémentation
- Exemple
- Finalité
- Illustration d'une solution récursive
- Exercices 1-2-3 série 2
- développement des solutions des exercices

La récursivité

Rappel et exercices

- Principe
- Implémentation
- Exemple
- Finalité
- Illustration d'une solution récursive
- Exercices 1-2-3 série 2
- développement des solutions des exercices

La récursivité

Principe

❖Tout objet est dit récursif s'il se définit à partir de lui-même

❖Ainsi, une fonction est dite récursive si elle comporte, dans son corps, au moins un appel à elle-même

- ❑ Correspondance mathématique
  - ▪Principe de récurrence

    Exemple : définition des entiers (Peano)

    • 0 est un entier

    • Si *n est un entier, alors n+1 est un* entier
- ❑ Exemples de fonctions récursives
  - ▪Calcul de la somme des entiers de 1 à *n*

    • On calcule la somme jusqu'à *n-1*

    • Puis on ajoute *n*

    •Idem avec le produit (fonction factorielle)

Un peu de vocabulaire

- Pour une fonction récursive, on parlera :
  - De récursivité terminale si aucune instruction n'est exécutée après l'appel de la fonction à elle-même
  - De récursivité non terminale dans l'autre cas

○ Exemple

Terminale

```
void f(int n){
        if(n==0) printf ("Hello");
        else f(n-1);
    }
```

Non terminale

```
void f(int n) {
        if(n>0)   f(n-1);
        printf ("Hello");
    }
```

Récursivité directe

- Lorsque *f s'appelle elle-même,* on parle de récursivité directe
- Lorsque *f appelle g qui appelle f,* il s'agit aussi de récursivité, on l'appelle alors indirecte.

Exemple Récursivité directe

```
Void K()
{  -
   -
   K()  // appel de la fonction à
elle-même
   -
}
```

Récursivité indirecte (croisée)

```
Void K()
{  -
   -
   G()  // appel de la fonction G
   -
}
```

```
Void G()
{  -
   -
   K()  // appel de la fonction K
   -
}
```

La récursivité

Rappel et exercices

- ✓ Principe
- ➢ Implémentation
- ➢ Exemple
- ➢ Finalité
- ➢ Illustration d'une solution récursive
- ➢ Exercices 1-2-3 série 2
- ➢ développement des solutions des exercices

Implémentation

- Comment programmer une fonction récursive ?
- Quels sont les pièges à éviter ?

❑ Il suffit de la faire s'appeler elle-même

```
int f(int n){
                return f(n-1);
            }
```

❑ *La fonction f est récursive : elle* s'appelle elle-même

❑ Mais *f n'a-t-elle pas un problème ?*

La fonction *f telle qu'elle est écrite ne* s'arrête pas :

- Appel : **f(2)**
- Appel : **f(1)**
- Appel : **f(0)**
- Appel : **f(-1)**
- Appel : **f(-2)**
- Etc...

Implémentation

- Comment y parvenir ?

❖ Première étape : la condition terminale

- *Obligatoirement au début de toute fonction récursive*
- Une condition : le cas particulier
- Pour ce cas, pas d'autre appel à la fonction : la chaîne d'appels s'arrête

Exemple

```
void f(int n){
                if(n==0)
                        printf ("Hello");
                else f(n-1);
               }
```

- Ici quand *n vaut 0, on s'arrête !!!*
- Problème : arrive-t-on à *n = 0 ?*

Implémentation

- ## Terminaison de la fonction

Il faut que la fonction s'arrête La condition terminale ne sert à rien si elle ne devient jamais vraie

Exemple avec la fonction précédente :

- **f(-2) provoque une pile d'appels infinie**
  - Appel : **f(-2)**
  - Appel : **f(-3)**
  - Appel : **f(-4)**
  - Appel : **f(-5)**
  - Etc...
- Probablement d'autres tests à faire (si *n<0, envoyer une exception par exemple)*

Implémentation

Une bonne solution

```
void f(int n) {
                    if(n<0)  break;
                    if(n==0)
                              printf ("Hello");
                     else f(n-1);
}
```

Pourquoi ça marche ?

- Si *n est négatif : on s'arrête sur une exception*
- Si *n est nul : c'est le cas d'arrêt (« Hello »)*
- Si *n est positif : on appelle f avec la valeur n-1*
- Chaine d'appels avec des valeurs entières strictement décroissantes de 1 en 1 ⟶ On arrive forcément à 0 ⟶ On affiche « Hello »
- On remonte la pile des appels (sans rien faire, ici la récursivité est terminale)

Implémentation

- En résumé

✓ Une fonction récursive doit comporter :

- Un cas d'arrêt dans lequel aucun autre appel n'est effectué
- Un cas général dans lequel un ou plusieurs autres appels sont effectués

✓ *La chaîne d'appel doit conduire au critère d'arrêt*

- Optionnellement, des cas impossibles ou incorrects à traiter par des exceptions

- Exemple : fonction factorielle n!

$$f(n) = n! = n \times (n-1) \times (n-2) \times \ldots\ldots \times 1$$

La récursivité

Rappel et exercices

- ✓ Principe
- ✓ Implémentation
- ➢ Exemple
- ➢ Finalité
- ➢ Illustration d'une solution récursive
- ➢ Exercices 1-2-3 série 2
- ➢ développement des solutions des exercices

Exemple

- Exemple : fonction factorielle n!

f(n) = n! = n x (n-1) x (n-2) x……x1

Solution Itérative

Repose sur la définition suivante :

n! = n x (n-1) x (n-2) x……x1
Cas particulier 0! = 1

```
int fact (int n) {
    int res = 1;
    for (int i=1; i<=n; i++)
        res = res*i;
    return res;
}
```

Solution récursive

Repose sur la définition suivante :

n! = n x (n-1) !
Cas particulier 0! = 1

```
int fact (int n){
                if(n==0)
                    return 1;
                else
                  return  n* fact (n-1);
}
```

Exemple

- Appel de fact(5) récursif

➢ *Phase de descente récursive*

Appel à **fact(5)**
Appel à **fact(4)**
Appel à **fact(3)**
Appel à **fact(2)**
Appel à **fact(1)**
Appel à **fact(0)**
Condition terminale
Retour de la valeur 1

Phase de remontée (après l'appel à fact(n-1)) on multiplie par *n* : *la récursivité n'est pas terminale)*

Retour de la valeur 120 (fact(5))
Retour de la valeur 24 (fact(4))
Retour de la valeur 6 (fact(3))
Retour de la valeur 2 (fact(2))
Retour de la valeur 1 (fact(1))
Retour de la valeur 1 (fact(0))

La récursivité

Rappel et exercices

- ✓ Principe
- ✓ Implémentation
- ✓ Exemple
- ➢ Finalité
- ➢ Illustration d'une solution récursive
- ➢ Exercices 1-2-3 série 2
- ➢ développement des solutions des exercices

Finalité

- Quelques conséquences

✓ La plupart des traitement sur les tableaux peuvent se mettre sous forme récursive :

- Tris (sélection, insertion)
- Recherche séquentielle (attention: pas dichotomique)
- Inversion
- Etc...

✓L'écriture sous forme récursive est toujours plus simple que l'écriture sous forme itérative

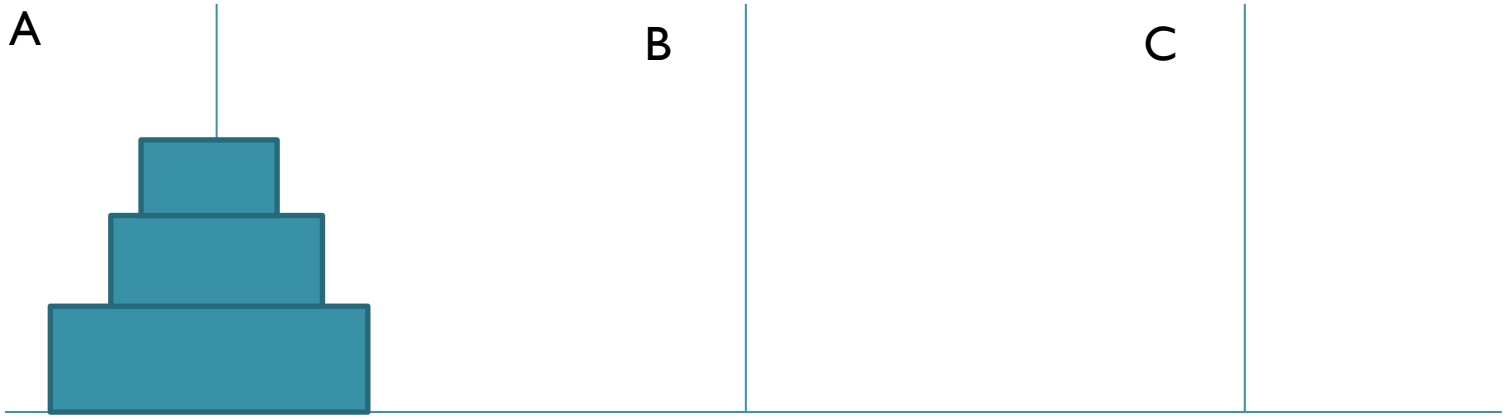
✓La plupart des traitements itératifs simples sont facilement traduisibles sous forme récursive

✓L'inverse est faux ⟶ Il arrive même qu'un problème ait une solution récursive triviale alors qu'il est très difficile d'en trouver une solution itérative

Finalité

- En résumé
  - On transforme un problème de taille *n* en deux problèmes de taille *n-1* *et un* problème de taille 1
  - Tactique classique : « diviser pour régner »
- Un Problème : Les tours de Hanoï

Transférer n disques de l'axe A à l'axe C, en utilisant B, de sorte que jamais un disque ne repose sur un disque de plus petit diamètre Ici : *n = 3*



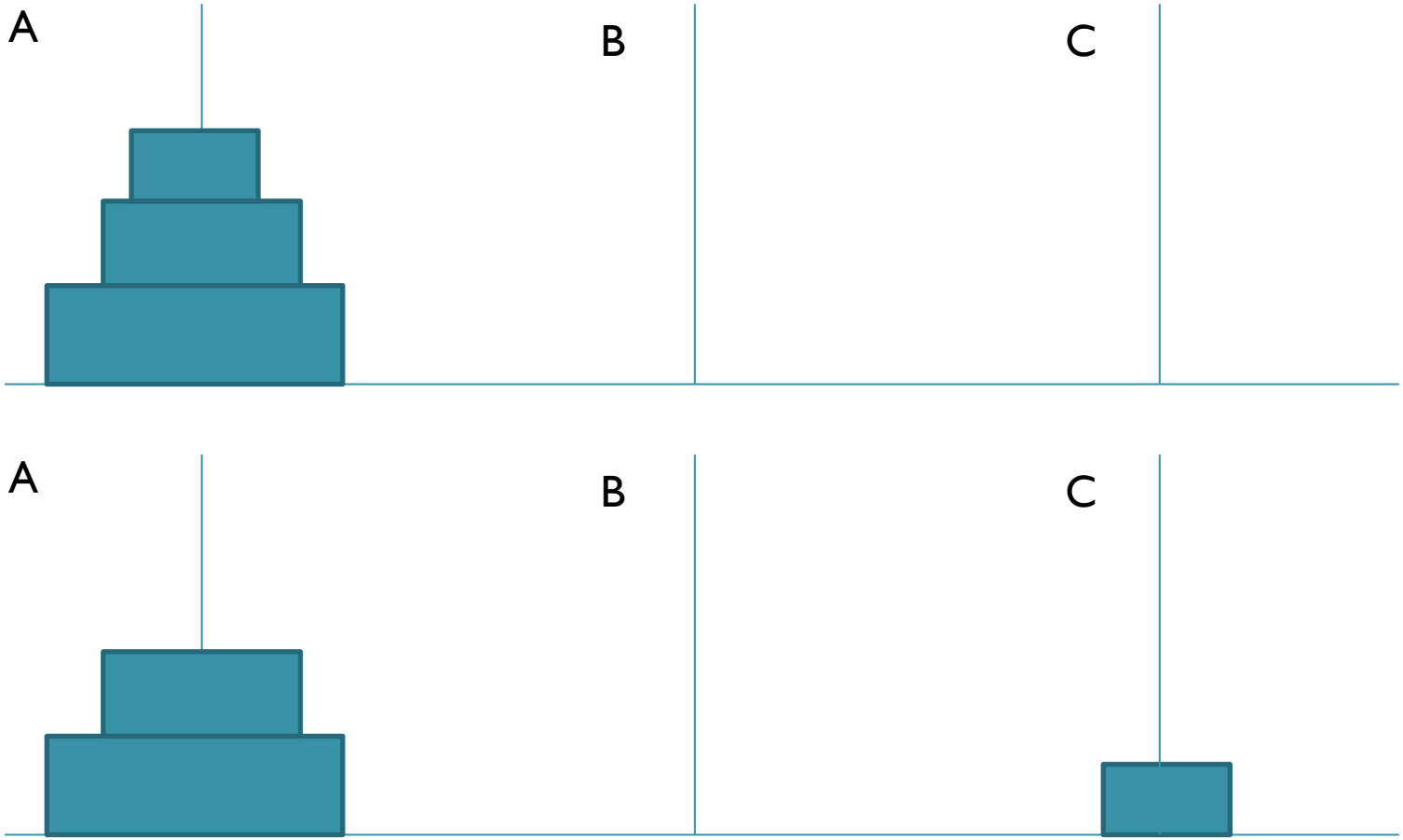
La récursivité

Rappel et exercices

- ✓ Principe
- ✓ Implémentation
- ✓ Exemple
- ✓ Finalité
- ➢ Illustration d'une solution récursive
- ➢ Exercices 1-2-3 série 2
- ➢ développement des solutions des exercices

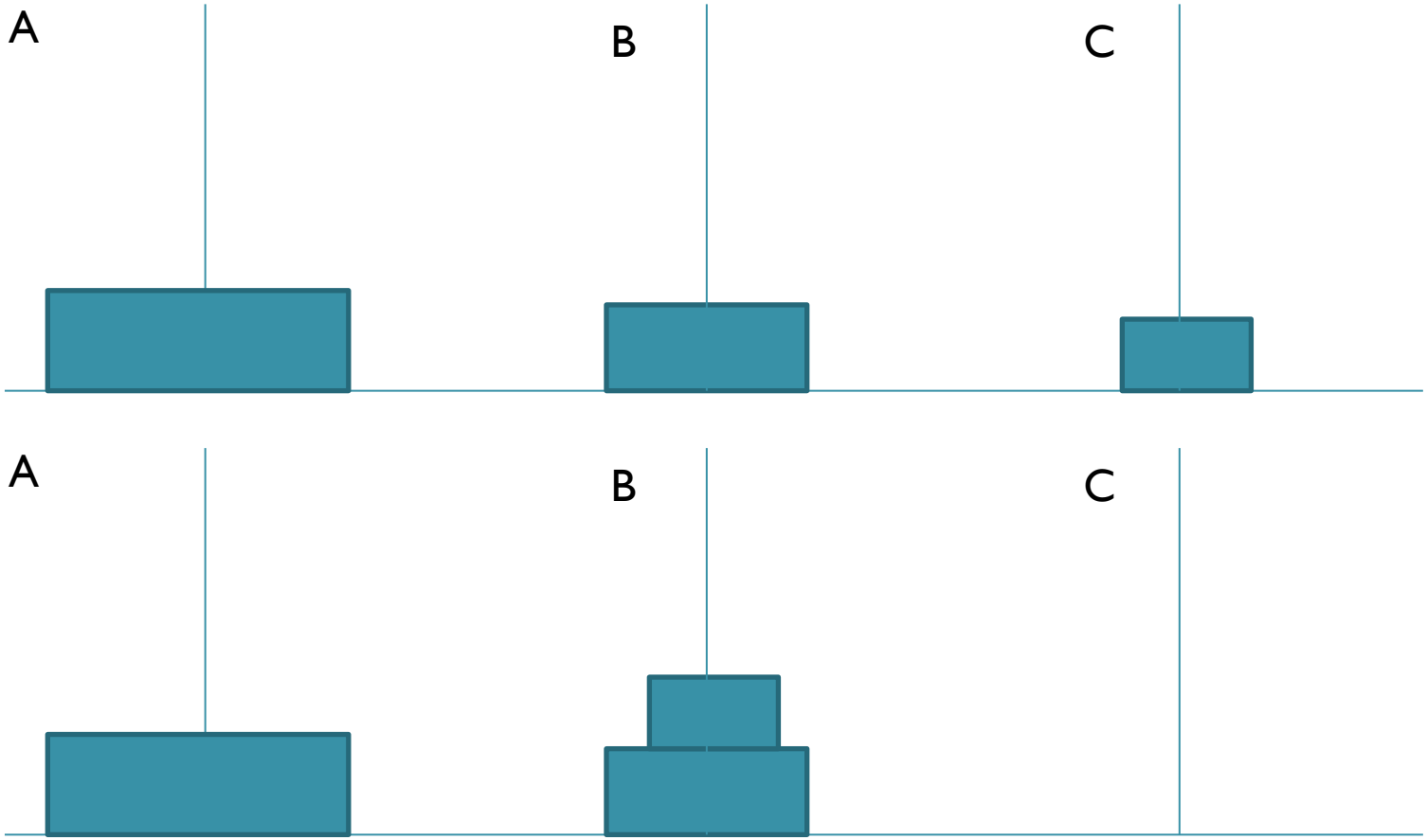
Illustration

- Solution: Les tours de Hanoï



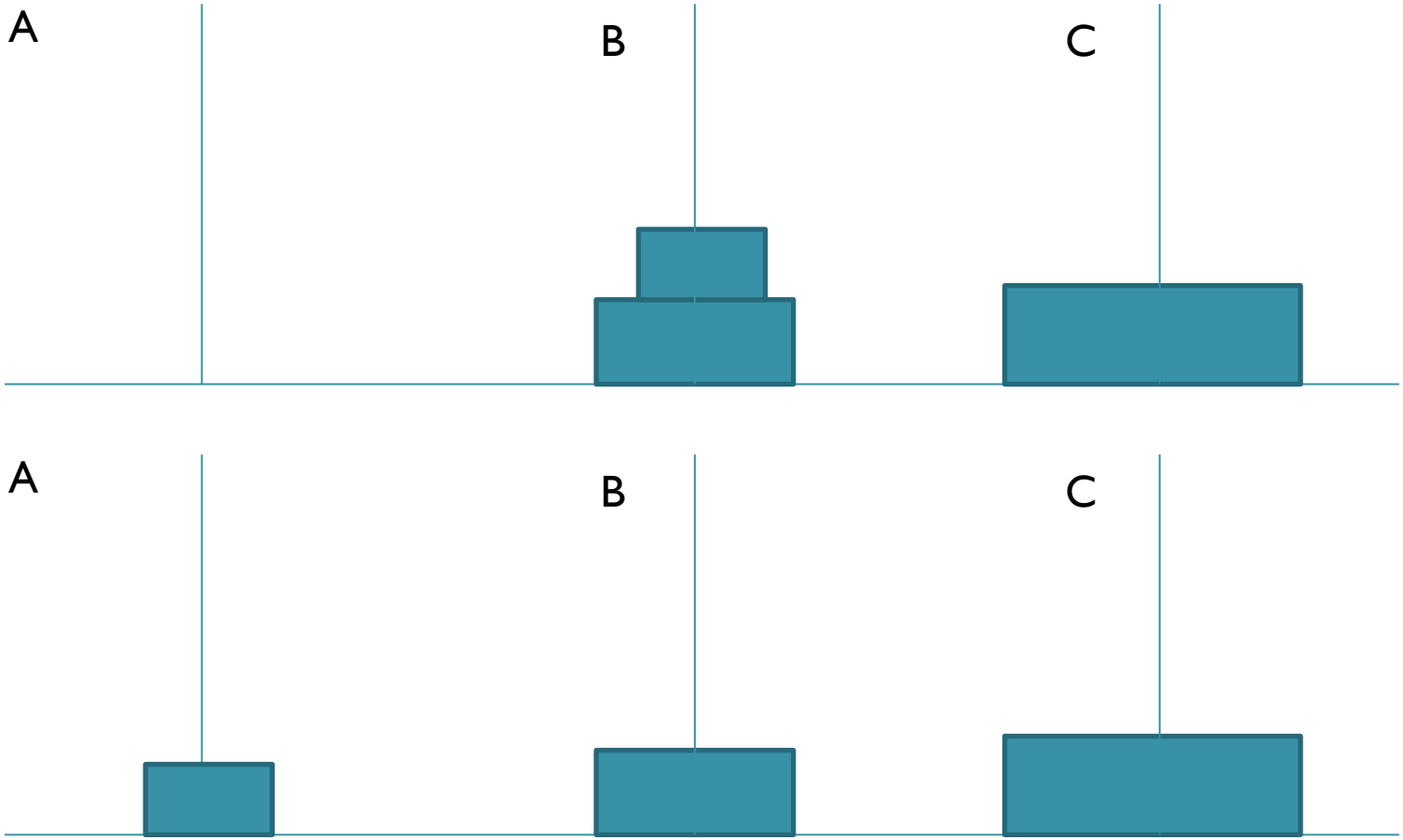
Illustration

- Solution: Les tours de Hanoï



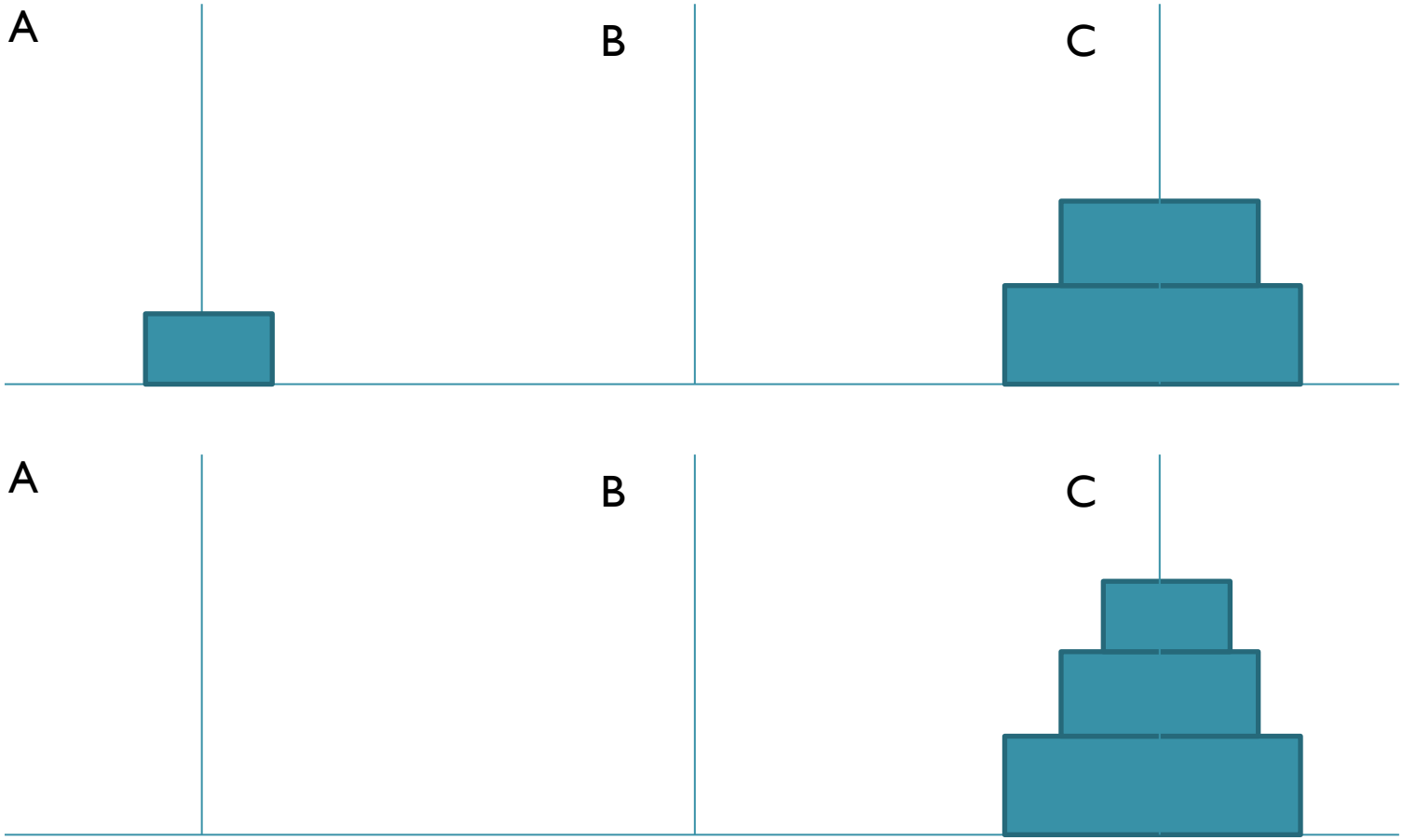
Illustration

- Solution: Les tours de Hanoï



Illustration

- Solution: Les tours de Hanoï



La récursivité

Rappel et exercices

- ✓ Principe
- ✓ Implémentation
- ✓ Exemple
- ✓ Finalité
- ✓ Illustration d'une solution récursive
- ➢ Exercices 1-2-3 série 2
- ➢ développement des solutions des exercices

Exercices 1-2-3 série 2

EX1 : Ecrire un sous-programme récursif qui calcule la somme $U_n = 1+2^4+3^4+\ldots+n^4$.

EX2 : La suite de Fibonacci est définie comme suit :

$U_0=0$;

$U_1=1$;

$U_n=U_{n-1}+U_{n-2}$ pour n>=2.

Ecrire un sous-programme récursif qui calcule cette fonction.

EX3 : Ecrire un sous-programme récursif qui inverse les éléments d'un tableau d'entiers.