ASD I

LFSI1/LAI1

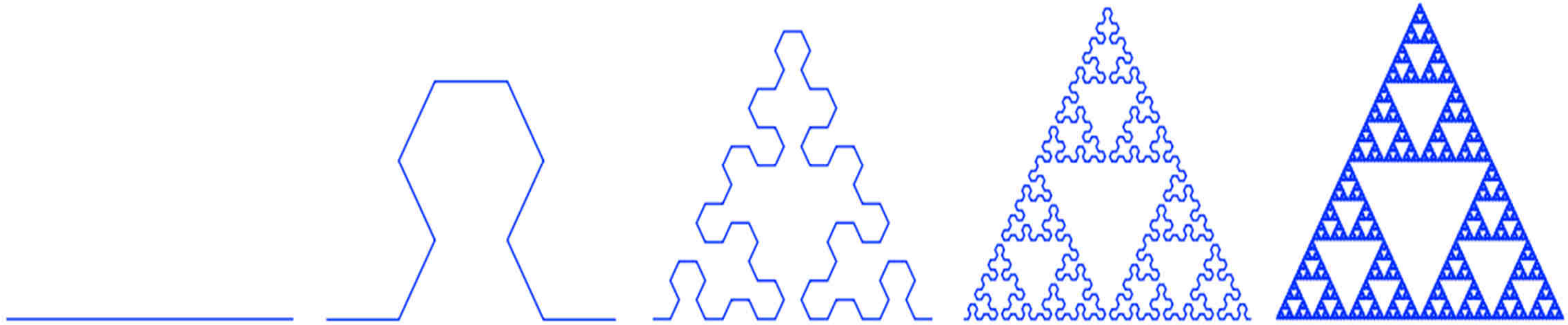
A-U: 2017/2018

5

LA RÉCURSIVITÉ

Introduction

- Une construction est récursive si elle se définit à partir d'elle-même
- Exemple : *le triangle de Sierpinski*



Définition

- L’appel d’un sous-programme à l’intérieur de lui-même est nommé appel récursif.
- Un appel récursif doit obligatoirement être dans une instruction conditionnelle (sinon la récursivité est sans fin).
- Exemple: La fonction suivante calcule la factorielle n! d’un nombre n. On se base sur la relation de récurrence $n! = (n - 1)! * n$

```
FONCTION FactorielleRecursive (n:entier) : entier
DÉBUT
      SI (n <= 0)
            Alors FactorielleRecursive ← 1
      SI NON
            FactorielleRecursive ← n * FactorielleRecursive (n-1) /* appel récursif */
FIN
```

ASD-LFSI1/LAI1–FSG -2017/2018

Définition

Appel à fact(4)
. 4*fact(3) = ?
. Appel à fact(3)
. . 3*fact(2) = ?
. . Appel à fact(2)
. . . 2*fact(1) = ?
. . . Appel à fact(1)
. . . . 1*fact(0) = ?
. . . . Appel à fact(0)
Retour de la valeur 1
. . . . 1*1
. . . Retour de la valeur 1
. . . 2*1
. . Retour de la valeur 2
. . 3*2
. Retour de la valeur 6
. 4*6
Retour de la valeur 24

Structure d'un sous-programme récursif

- Pour créer un sous-programme récursif, il faut :

1. décomposer un problème en un ou plusieurs sous-problèmes du même type. On résoudra les sous-problèmes par des appels récursifs.
2. Les sous-problèmes doivent être de taille plus petite que le problème initial
3. La décomposition doit en fin de compte conduire à un **cas élémentaire**, qui, lui, n'est pas décomposable en sous-problème. (condition d'arrêt).

Structure d’un sous-programme récursif

- Très Important :

1. un appel récursif peut produire lui-même un autre appel récursif, etc, ce qui peut mener à une suite infinie d’appels.
   - Il faut arrêter la suite d’appels au moment où le sous-problème peut être résolu directement.

```
FONCTION FactorielleRecursive (n:entier) : entier
DÉBUT
    SI (n = 0)
        Alors FactorielleRecursive ← 1
    SI NON
        FactorielleRecursive ← n * FactorielleRecursive (n-1) /* appel récursif */
FIN
```

Et si n = -1 !!!!!

ASD-LFSI1/LAI1–FSG -2017/2018

Structure d'un sous-programme récursif

- Très Important :

2. Dans un sous-programme récursif, il faut s'assurer que la condition d'arrêt est atteinte après un nombre fini d'appels.

   - la condition d'arrêt doit être choisie avec soin. Elle doit correspondre en principe au cas "le plus simple" qu'on veut traiter

```
FONCTION FactorielleRecursive (n:entier) : entier
DÉBUT
    SI (n <= 0)
        Alors FactorielleRecursive ← 1
    SI NON
        FactorielleRecursive ← n * FactorielleRecursive (n-1) /* appel récursif */
FIN
```

Quand utiliser la récursivité ?

- Est-ce que le problème dépend d'un (ou plusieurs) paramètre(s) ?
- Est-il possible de résoudre le problème lorsque la (les) valeur(s) du paramètre est "petite(s)" ?
- Est-il possible de résoudre le problème à l'aide de la résolution du problème portant sur une (des) "plus petite(s)" valeur(s) du paramètre ?

➢ Si oui, oui, oui, alors la résolution par un sous-programme récursif est à envisager.

Types de récursivité

- Récursivité directe (simple)
- Récursivité indirecte (croisée)
- Récursivité terminale
- Récursivité non terminale
- Récursivité imbriquée

Types de récursivité

- Récursivité directe :
  - Quand un sous-programme fait appel à lui même, comme dans le cas de la factorielle, on appelle cela récursivité directe.
- Récursivité indirecte
  - Si un sous-programme A fait appel à un sous-programme B, qui lui même fait appel au sous-programme A.

Types de récursivité

- Récursivité indirecte :
  - Exemple: la définition récursive des nombres pairs et impairs. Un nombre n positif est pair si n-1 est impair ; un nombre n positif est impair si n-1 est pair..

```
fonction estPair(n:entier ) : booléen
début
    si (n = 0) alors
        renvoyer VRAI;
    sinon
        renvoyer estImpair(n-1);
    finsi
fin
```

```
fonction estImpair(n:entier ) : booléen
début
    si (n = 0) alors
        renvoyer FAUX;
    sinon
        renvoyer estPair(n-1);
    finsi
fin
```

Types de récursivité

- Récursivité imbriquée :
  - faire un appel récursif à l'intérieur d'un autre appel récursif.
  - Exemple : la suite d'Ackerman
    - $A(m,n) = n+1$ si $m = 0$,
    - $A(m,n) = A(m-1,1)$ si $n=0$ et $m > 0$
    - $A(m,n) = A(m-1, A(m,n-1))$ sinon

Types de récursivité

- Récursivité imbriquée :
  - Exemple : la suite d'Ackerman

```
Fonction ackerman (m:entier, n:entier ) : entier
Début
        si (m = 0) alors
                renvoyer n+1;
        sinon
                si ((m>0) et (n=0)) alors
                                renvoyer ackerman(m-1,1);
                sinon
                                renvoyer ackerman(m-1,ackerman(m,n-1));
                finsi
        finsi
Fin
```

Types de récursivité

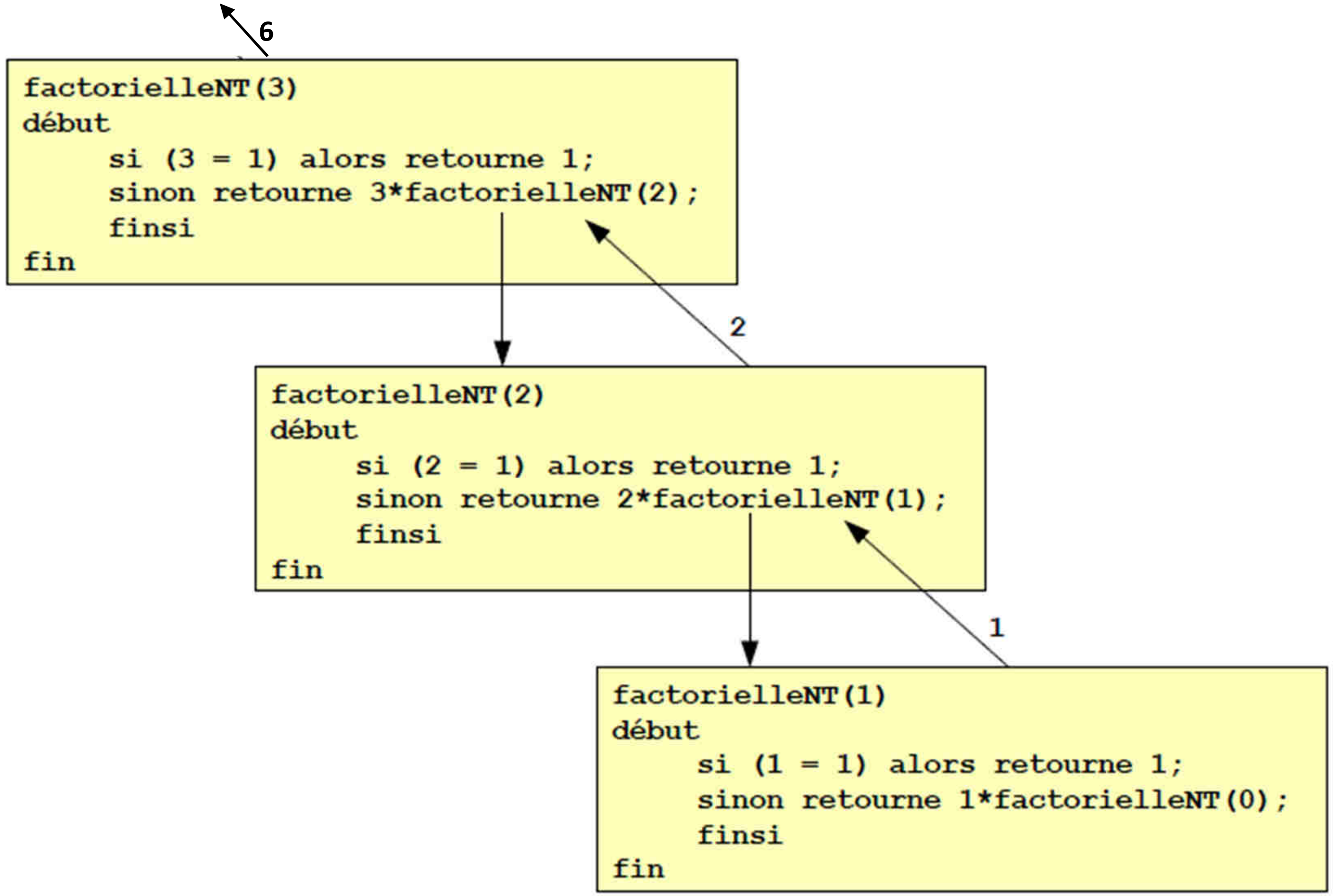
- Récursivité Non Terminale :
  - le résultat de l'appel récursif est utilisé pour réaliser un traitement (en plus du retour d'une valeur).
  - Exemple : fonction factorielle

```
Fonction factorielleNT (n:entier) : entier
Début
        Si (n <= 0) alors
                Renvoyer 1
        Sinon
                Renvoyer  n *factorielleT (n-1)
        Finsi
Fin
```

ASD-LFSI1/LAI1–FSG -2017/2018

Types de récursivité

- Récursivité Non Terminale :



Types de récursivité

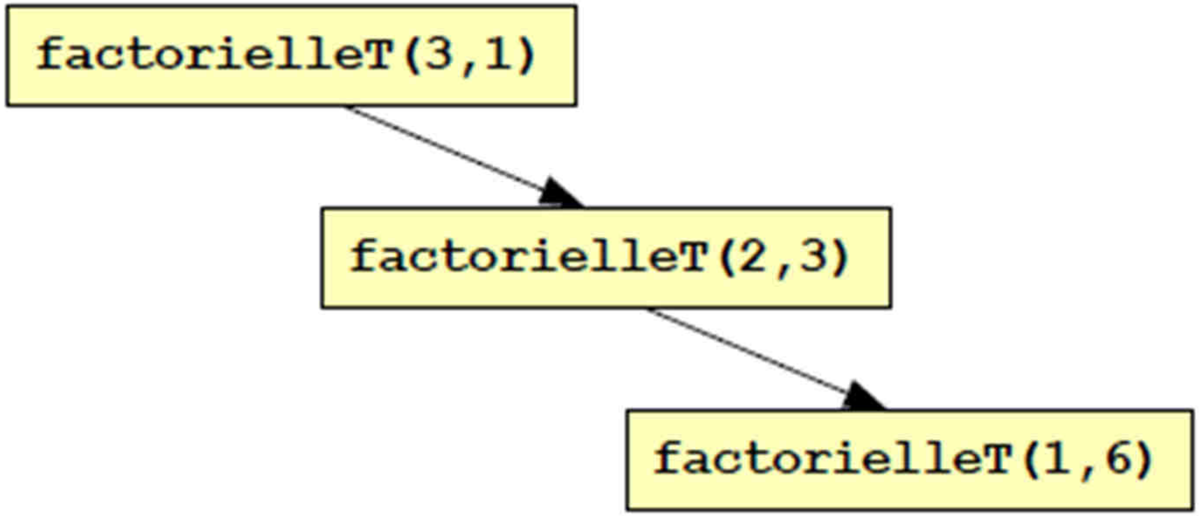
- Récursivité Terminale :
  - aucun traitement n'est effectué à la remontée d'un appel récursif (sauf le retour d'une valeur).
  - Exemple : fonction factorielle

```
Fonction factorielleT (n:entier, resultat:entier) : entier
Début
        Si (n = 1) alors
                Renvoyer resultat
        Sinon
                Renvoyer factorielleT (n-1, n * resultat)
        Finsi
Fin
```

Accumulateur

Types de récursivité

- Récursivité Terminale :



Récursive vs. Itérative

- Nous pouvons définir une version itérative d'un sous-programme récursif.
- Exemple: La version itérative de Factorielle

```
FONCTION FactorielleItérative (n:entier) : entier
Var: i, Resultat: entier
DÉBUT
        Résultat ← 1
        Pour i de 1 à n faire
                Résultat ← Résultat * i
        Fin Pour
        FactorielleItérative ← Résultat
FIN
```

Récursive vs. Itérative

- De la fonction récursive **terminale** vers la version itérative (dérécursivation)

```
Fonction RecursiveT ( P ): type
Var
          ….
Début
          Trait0
          si cond alors
                    Trait1
          sinon
                    Trait2
                    RecursiveT ( f(P))
          finsi
Fin
```

```
Fonction itérative ( P ): type
Var
          ….
Début
          Trait0
          Tant que (non cond)
                    Trait2
                    P ← f(P)
                    Trait0
          FinTantque
          Trait1
Fin
```

Récursive vs. Itérative

- De la fonction récursive **terminale** vers la version itérative (dérécursivation) : Exemple de Factorielle

```
Fonction factorielleRecTer (n:entier, resultat:entier) : entier
Début
            Si (n <= 1) alors
               Renvoyer resultat
            Sinon
               Renvoyer factorielleT (n-1, n * resultat)
            Finsi
Fin
```

- Trait1 : Renvoyer resultat
- Cond : n <= 1 ➔ Non Cond : n>1
- P: n et resultat, f(P) : n ← n-1 et resultat ← n * resultat

Récursive vs. Itérative

- De la fonction récursive **terminale** vers la version itérative (dérécursivation) : Exemple de Factorielle

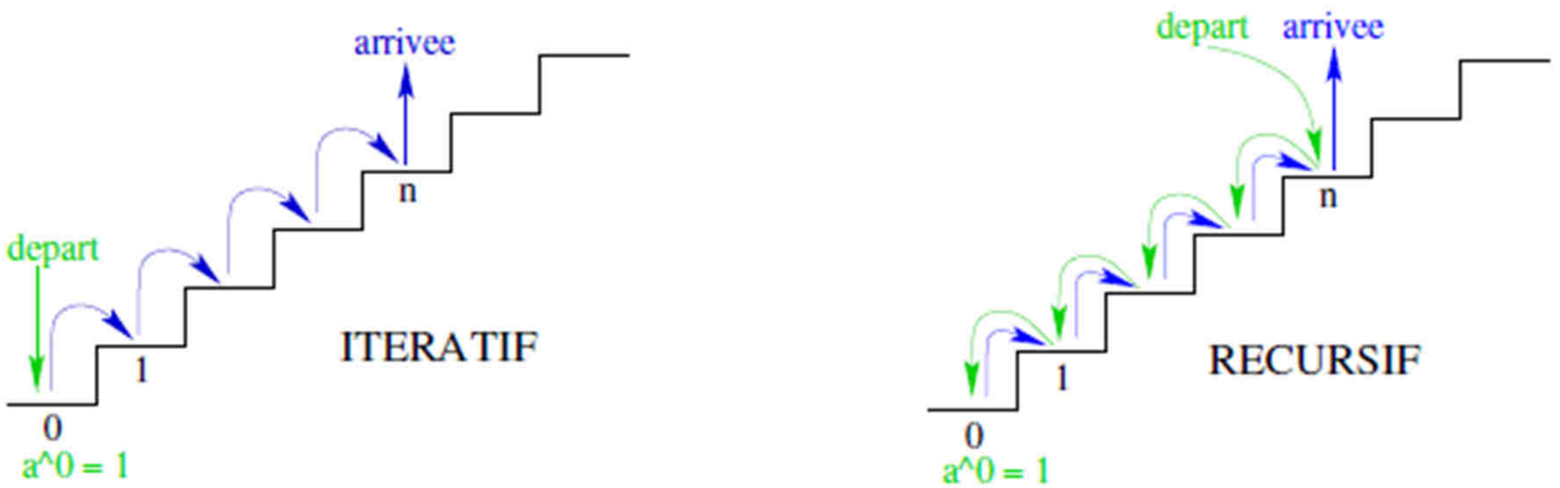
```
Fonction factorielleItérative (n:entier, resultat:entier) : entier
Début
        Tantque (n > 1) faire
            resultat ← n * resultat
            n ← n-1
        FinTantque
        Renvoyer resultat
Fin
```

- Cond : n <= 1 → Non Cond : n>1
- P: n et resultat, f(P) : n ← n-1 et resultat ← n * resultat

Récursive vs. Itérative

- Calcul de la fonction a^n (l'escalier & la puissance)

$$a^n = a \times a^{n-1}$$

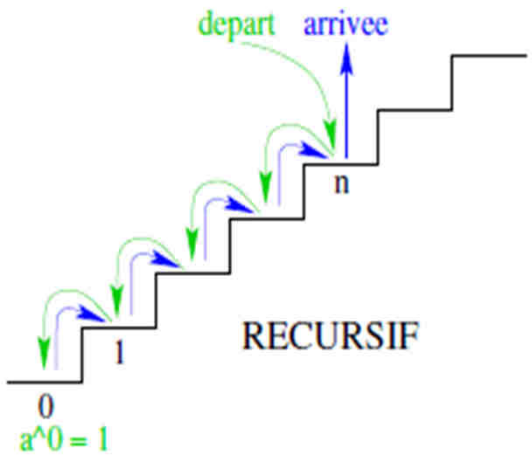


ASD-LFSI1/LAI1–FSG -2017/2018

Récursive vs. Itérative

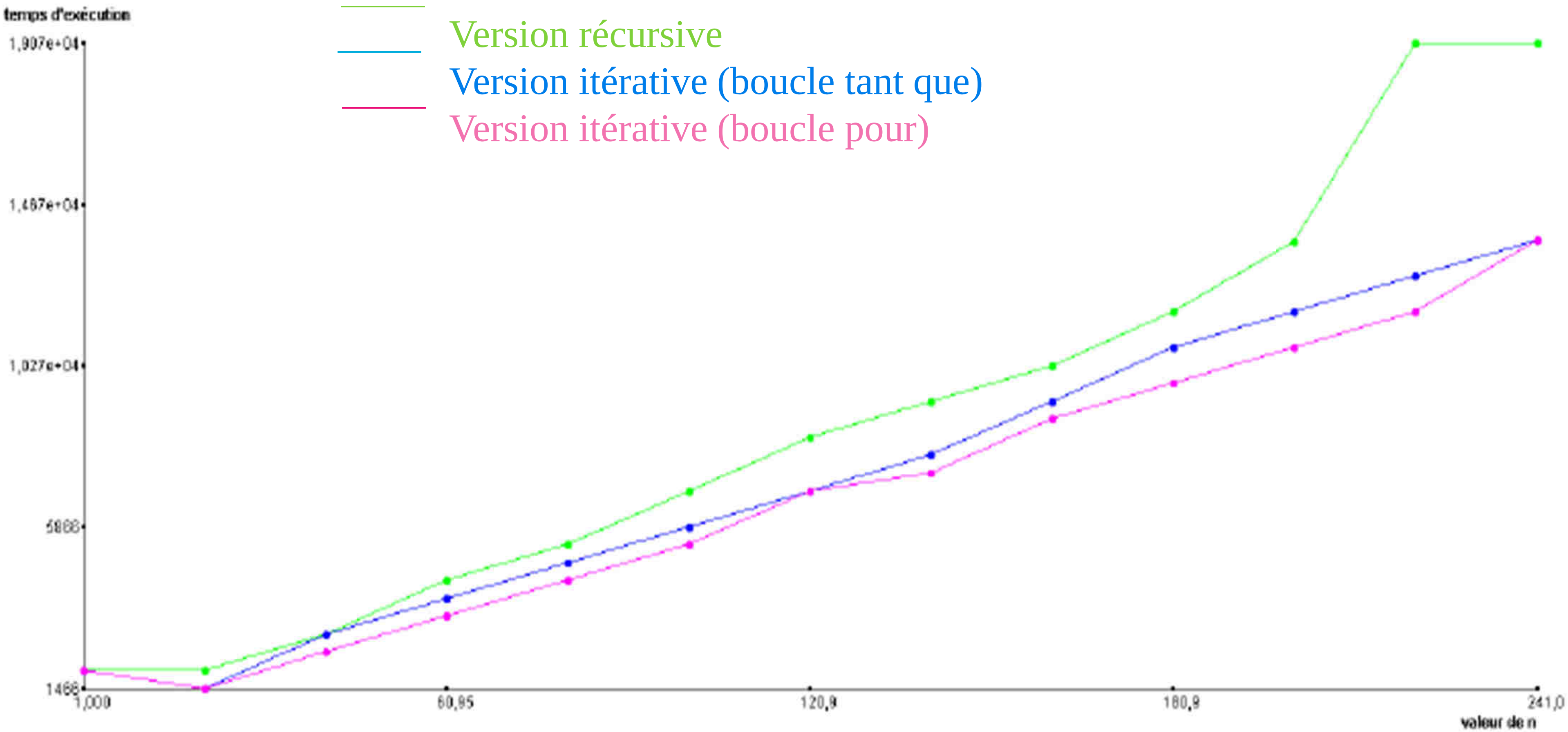
- Calcul de la fonction a^n

```
Fonction puissance (a : réel, n : entier):  réel
Début
        Si n = 0 alors
                Renvoyer 1
        Sinon
                Renvoyer puissance(a, n-1) * a
        Fin Si
Fin
```



Récursive vs. Itérative

Comparaison expérimentale du calcul de la factorielle en itératif et en récursif



Intérêt de la récursivité

- Elle est bien adaptée à la résolution de certains problèmes (et pas seulement mathématiques !)
- Avec la récursivité, les algorithmes sont souvent moins "laborieux" à écrire : moins de variables, beaucoup moins de boucles.
- Une résolution par un sous-programme récursif nécessite souvent de **prendre du recul** pour résoudre le problème (avantage !)

Notion de Pile d'exécution

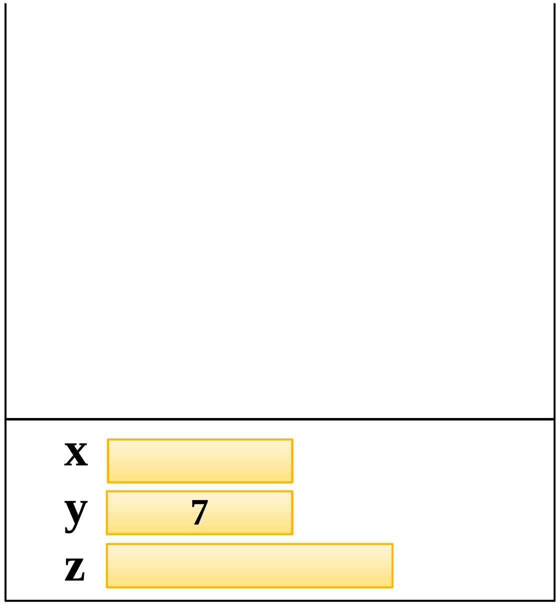
- La version itérative consomme moins de mémoire car l'appel récursif exige l'utilisation d'une **<u>pile d'exécution</u>**
- Un appel d'une fonction récursive ne se termine pas avant que tous les sous-problèmes soient résolus.
- La Pile d'exécution du programme en cours est un emplacement mémoire destiner à mémoriser
  - les paramètres,
  - les variables locales
  - les adresses de retour des fonctions en cours d'exécution.

Notion de Pile d'exécution

- Pour comprendre … Allocation de la mémoire:
  - Soit l'exemple suivant :

Étape 1

Pile d'exécution




```
Algorithme Exemple
Var
            x, y : entier
            z : réel
FONCTION Fonct1 (n:entier) : entier
Var: Resultat: entier
DÉBUT
            si n mod 2 = 0 alors Resultat ← n div 2
            sinon Resultat ← (n-1) div 2
            finsi
            Renvoyer Resultat
FIN
DÉBUT
y ← 7
x ← Fonct1(y)
z ← Fonct1(x)/y
Fin
```

Allocation de mémoire pour les variables du programme principal

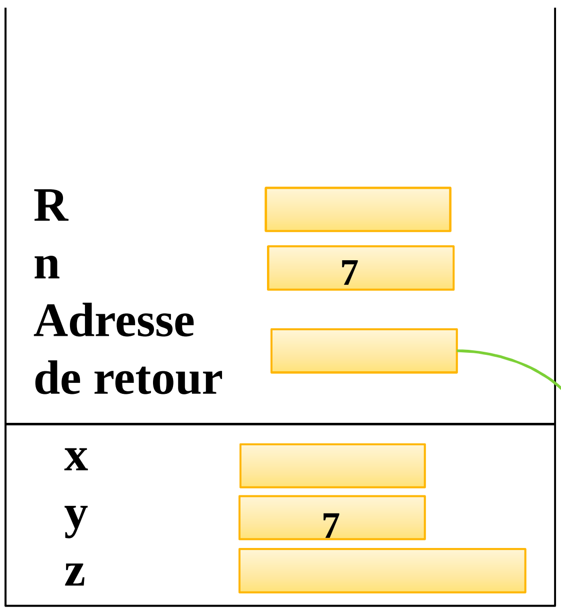
Enregistrement de la valeur 7 dans l'espace mémoire réservé à la variable y

Notion de Pile d'exécution

- Pour comprendre … Allocation de la mémoire:
  - Soit l'exemple suivant :

Étape 2

Pile d'exécution




```
Algorithme Exemple
Var
            x, y : entier
            z : réel
FONCTION Fonct1 (n:entier) : entier
Var: R: entier
DÉBUT
            si n mod 2 = 0 alors R ← n div 2
            sinon R ← (n-1) div 2
            finsi
            Renvoyer R
FIN
DÉBUT
y ← 7
x ← Fonct1(y)
z ← Fonct1(x)/y
Fin
```

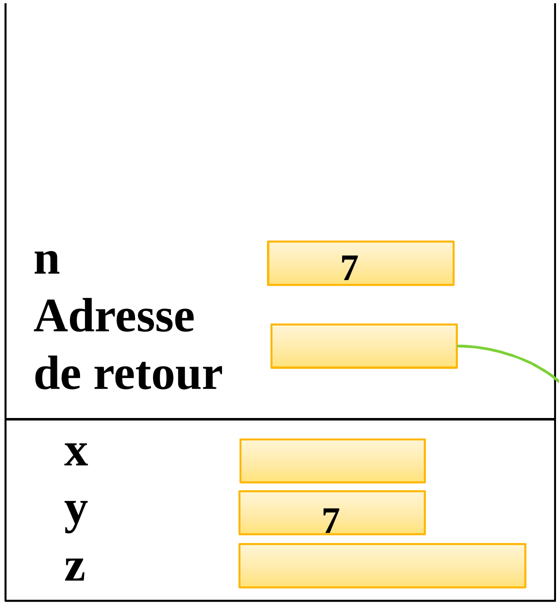
Appel de la fonction Fonct1 → allocation de la mémoire pour les paramètres (n), les variables locales (R) et l'adresse de retour

Notion de Pile d’exécution

- Pour comprendre … Allocation de la mémoire:
  - Soit l’exemple suivant :

Étape 3

Pile d’exécution




```
Algorithme Exemple
Var
        x, y : entier
        z : réel
FONCTION Fonct1 (n:entier) : entier
Var: R: entier
DÉBUT
        si n mod 2 = 0 alors R ← n div 2
        sinon R ← (n-1) div 2
        finsi
        Renvoyer R
FIN
DÉBUT
y ← 7
x ← Fonct1(y)
z ← Fonct1(x)/y
Fin
```

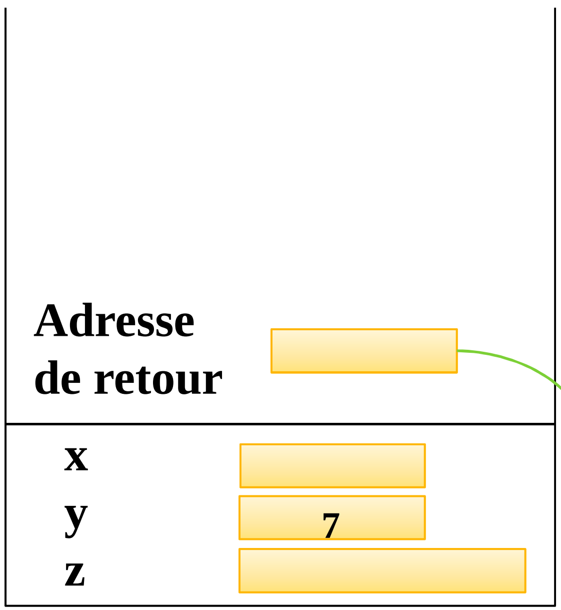
Dépiler les variables locales après l’exécution du code de la fonction et enregistrement du résultat dans un registre

Notion de Pile d’exécution

- Pour comprendre … Allocation de la mémoire:
  - Soit l’exemple suivant :

Étape 4

Pile d’exécution




```
Algorithme Exemple
Var
        x, y : entier
        z : réel
FONCTION Fonct1 (n:entier) : entier
Var: R: entier
DÉBUT
        si n mod 2 = 0 alors R ← n div 2
        sinon R ← (n-1) div 2
        finsi
        Renvoyer R
FIN
DÉBUT
y ← 7
x ← Fonct1(y)
z ← Fonct1(x)/y
Fin
```

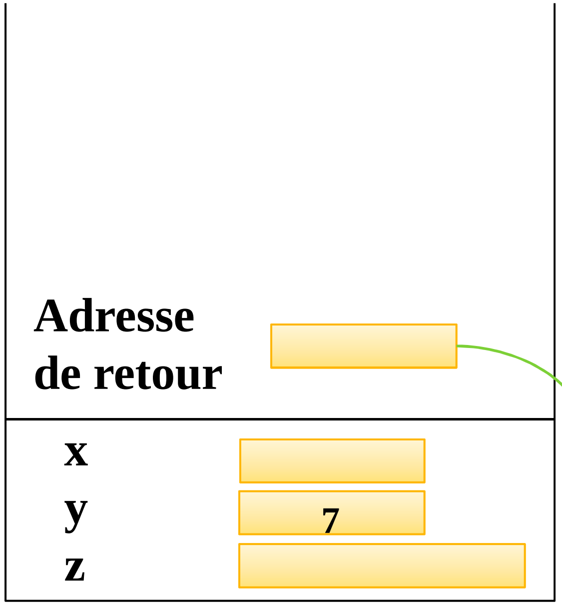
Dépiler les paramètres passés à la fonction

Notion de Pile d'exécution

- Pour comprendre … Allocation de la mémoire:
  - Soit l'exemple suivant :

Étape 5

Pile d'exécution




```
Algorithme Exemple
Var
        x, y : entier
        z : réel
FONCTION Fonct1 (n:entier) : entier
Var: R: entier
DÉBUT
        si n mod 2 = 0 alors R ← n div 2
        sinon R ← (n-1) div 2
        finsi
        Renvoyer R
FIN
DÉBUT
y ← 7
x ← Fonct1(y)
z ← Fonct1(x)/y
Fin
```

Exécuter l'instruction à l'adresse de retour (affectation du résultat de Fonct1 à x) puis dépiler l'adresse de retour

Notion de Pile d'exécution

- Pour comprendre … Allocation de la mémoire:
  - Soit l'exemple suivant :

Étape 6

Pile d'exécution

x	
y	**7**
z	

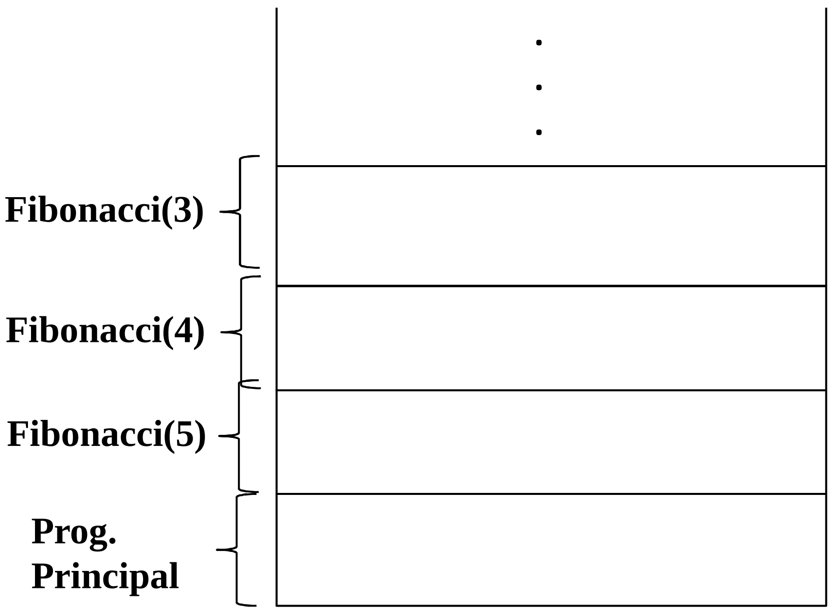
```
Algorithme Exemple
Var
        x, y : entier
        z : réel
FONCTION Fonct1 (n:entier) : entier
Var: R: entier
DÉBUT
        si n mod 2 = 0 alors R ← n div 2
        sinon R ← (n-1) div 2
        finsi
        Renvoyer R
FIN
DÉBUT
y ← 7
x ← Fonct1(y)
z ← Fonct1(x)/y
Fin
```

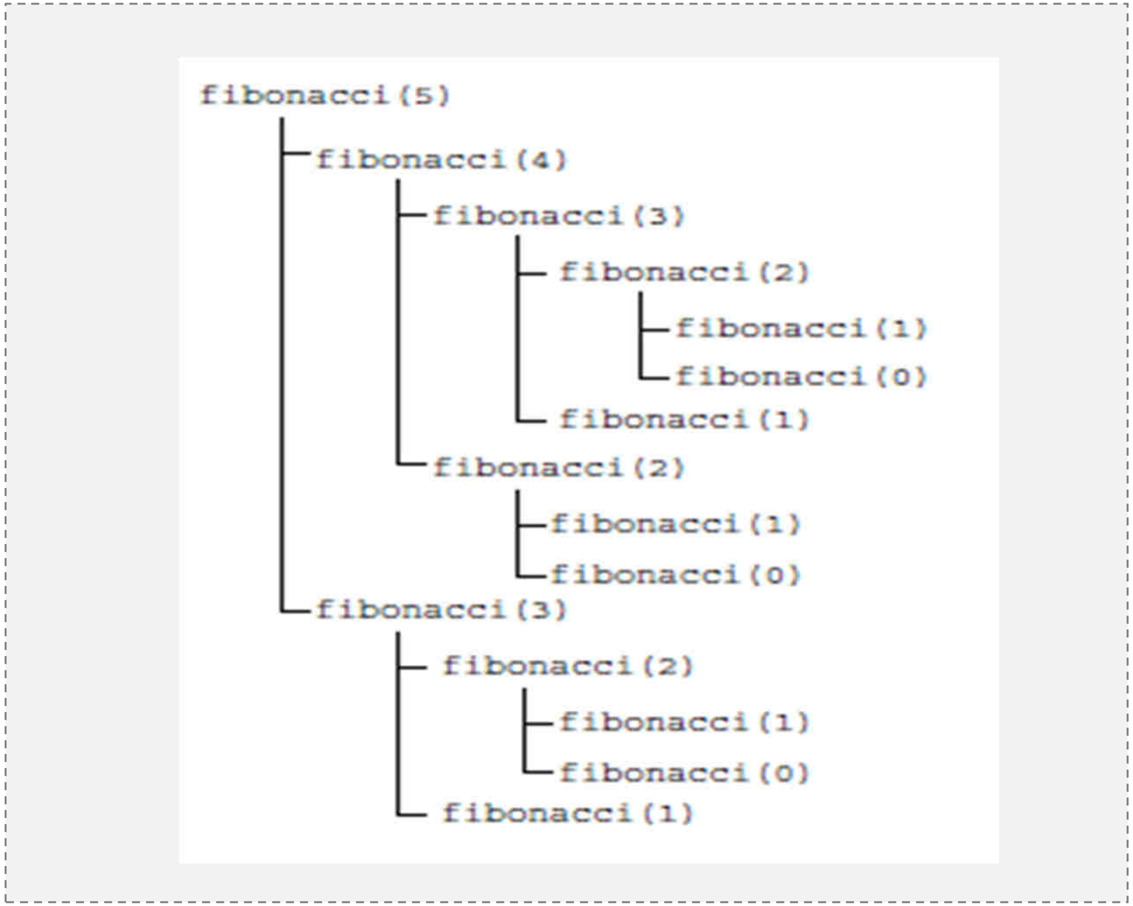
Refaire les mêmes étapes pour le deuxième appel de Fonct1

Notion de Pile d'exécution

- Et pour une fonction récursive ….

Pile d'exécution





Notion de Pile d'exécution

- Attention : exécuter trop d'appels de fonction fera déborder la pile d'exécution!

```
public static void testPile(int nbAppels){
     System.out.println("appel numéro " + nbAppels);
     testPile(nbAppels + 1);
}
...
testPile(1);
```

```
appel numéro 1
appel numéro 2
...
appel numéro 5613
Exception in thread "main" java.lang.StackOverflowError
        at sun.nio.cs.SingleByte.withResult(SingleByte.java:44)
        at sun.nio.cs.SingleByte.access$000(SingleByte.java:38)
        at sun.nio.cs.SingleByte$Encoder.encodeArrayLoop(SingleByte.java:187)
```

Exemple 1: Recherche dichotomique

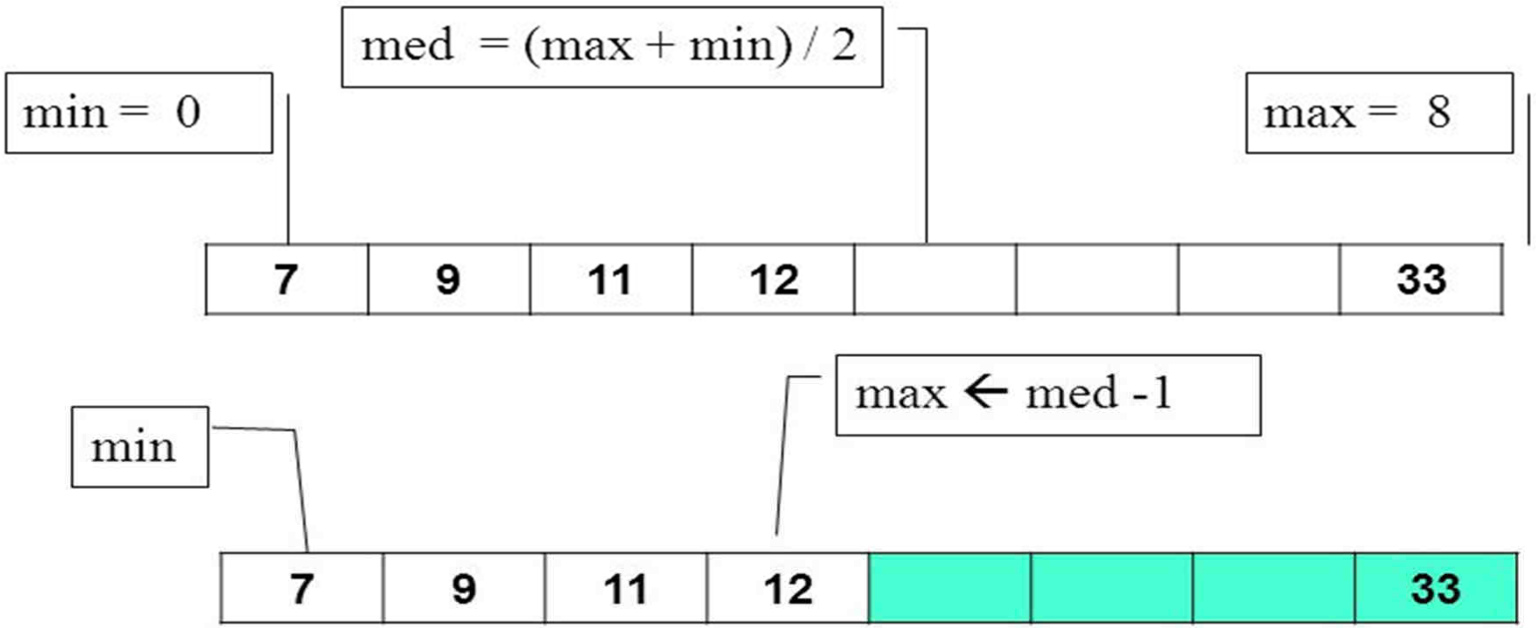
- Objectif : écrire une fonction récursive qui recherche par dichotomie un élément dans un tableau trié d'entiers. La fonction renvoie l'indice de l'élément s'il existe et -1 sinon.
  - Décomposition du traitement :
    - rechercher un élément dans le tableau va conduire, si on ne trouve pas l'élément au milieu, à relancer la recherche sur une moitié du tableau, puis sur un quart, etc.
    - A chaque appel récursif, il faut donc savoir entre quels indices i et j on cherche l'élément.
  - Condition d’arrêt:
    - la recherche s'arrête quand on trouve l'élément ou quand il n'y a plus de case où chercher (i>j).

Exemple 1: Recherche dichotomique

- A chaque étape :
  - Tester si le tableau est vide (en ce cas arrêt des appels récursifs avec échec)
  - Calculer l'indice moyen (indice max + indice min)/2
  - Comparer la valeur présente à l'indice moyen avec l'objet recherché,
    - si l'objet recherché est à l'indice moyen (arrêt succès)
    - si l'objet est supérieur ou égal à la valeur t(moyen) relancer recherche avec le tableau supérieur,
    - sinon relancer la recherche avec le tableau inférieur.

Exemple 1: Recherche dichotomique

- Principe :
  - Tableau trié en ordre croissant
  - la valeur cherchée est 11, la valeur médiane est celle dans la case numéro 4 ((0+8)/ 2)qui est > 11. Donc recherche dans la partie inférieure



Exemple 1: Recherche dichotomique

- Fonction récursive

```
Fonction dicho (t(): entier, min, max, objet: entier): entier
Début
        si (min>max) alors
                renvoyer -1
        sinon si (t((min+max)/2)=objet) alors
                renvoyer (min+max)/2
        sinon
                si (t((min+max)/2) > objet) alors
                        renvoyer dicho(t,min, (min+max)/2 – 1, objet)
                sinon
                        retourne dicho(t, (min+max)/2+1, max, objet)
                finsi
        finsi
Fin
```

Exemple 2: Tour de Hanoï

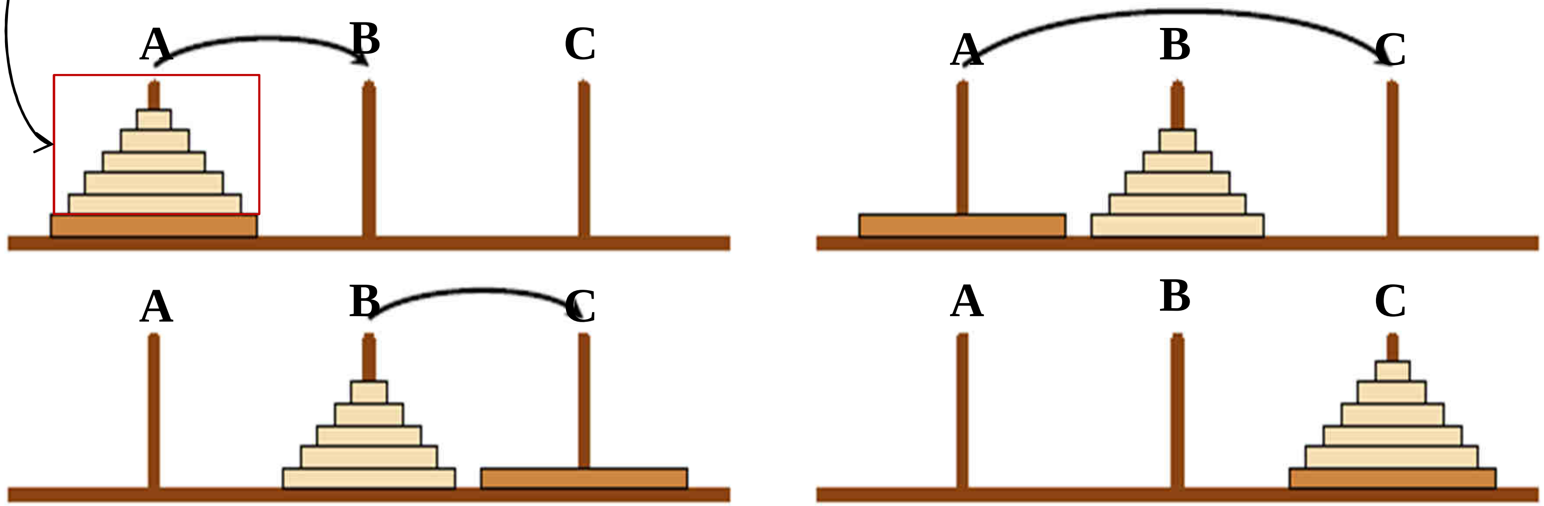
- Le casse-tête des tours de Hanoï est un jeu de réflexion consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d' « arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups, tout en respectant les règles suivantes :
  - on ne peut pas déplacer plus d'un disque à la fois,
  - on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

Exemple 2: Tour de Hanoï

❑ Raisonnement par récurrence

Sous-problèmes de taille n-1

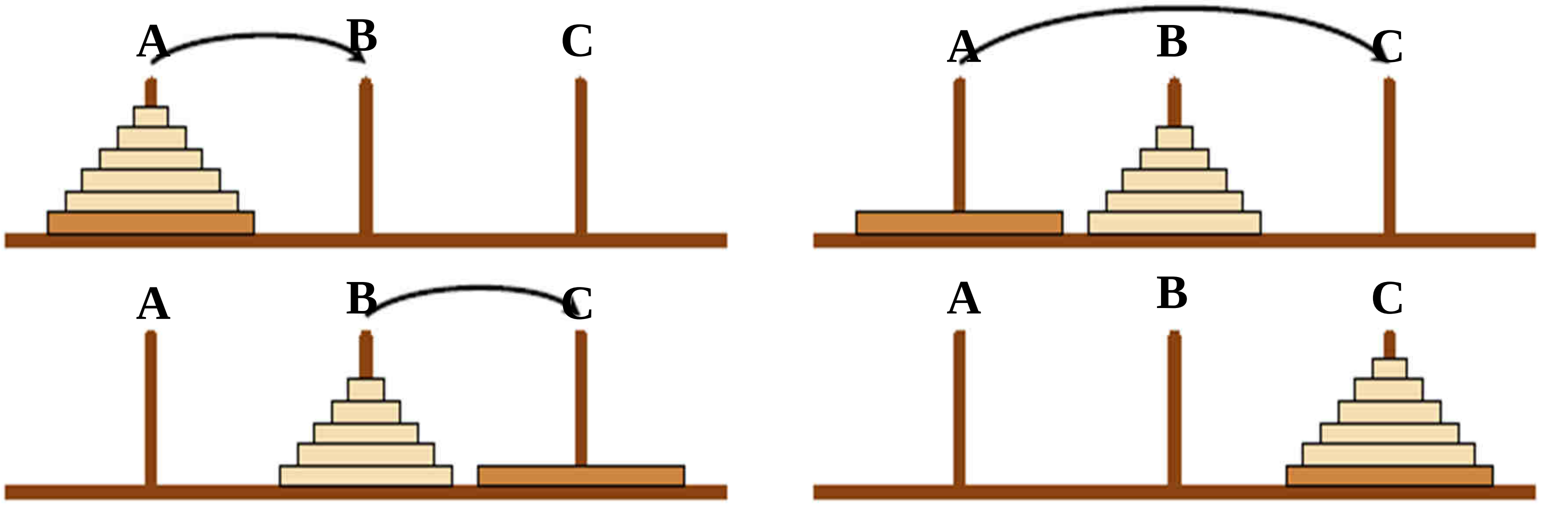
pour pouvoir déplacer *n* disques de la tige A vers la tige C il suffit de savoir déplacer *n* - 1 disques de la tige A vers la tige B puis de la tige B vers la tige C



Exemple 2: Tour de Hanoï

❑ Raisonnement par récurrence

- ❑ déplacer (n-1) disques de A vers B (en passant par C);
- ❑ déplacer le plus grand disque de A vers C ;
- ❑ déplacer (n-1) disques de B vers C (en passant par A).



Exemple 2: Tour de Hanoï

❑ Raisonnement par récurrence

1. déplacer (n-1) disques de A vers B (en passant par C);

   1.1. déplacer (n-2) disques de A vers C (en passant par B);

   1.2. déplacer un disque de A vers B ;

   1.3. déplacer (n-2) disques de C vers B (en passant par A).

2. déplacer le plus grand disque de A vers C ;

3. déplacer (n-1) disques de B vers C (en passant par A).

   3.1. déplacer (n-2) disques de B vers A (en passant par C);

   3.2. déplacer un disque de B vers C ;

   3.3. déplacer (n-2) disques de A vers C (en passant par B).

Exemple 2: Tour de Hanoï

Types structurés à définir dans le programme principal

Raisonnement par récurrence

- Procédures disponibles:
  - procédure dépilerTour (Var t : TourDeHanoi, Var d : Disque)
  - procédure empilerTour (Var t : TourDeHanoi, d : Disque)
- Procédure à définir
  - procédure resoudreToursDeHanoi (nbDisquesADeplacer : entier, Var source, destination, intermediaire : TourDeHanoi)

Exemple 2: Tour de Hanoï

❑ Raisonnement par récurrence

Procédure resoudreToursDeHanoi (nbDisquesADeplacer : entier, Var source, destination, intermediaire : TourDeHanoi)
Var
    d: Disque
Début
  si nbDisquesADeplacer>0 **alors**
    resoudreToursDeHanoi(nbDisquesADeplacer-1, source, intermediaire, destination)
    depiler(source,d)
    empiler(destination,d)
    resoudreToursDeHanoi(nbDisquesADeplacer-1, intermediaire, destination, source)
  finsi
Fin