

# UEF 1 : Informatique & Programmation

Faculté des Sciences de Nice

DEUG 2001-2002

Jérôme DURAND-LOSE

Jean-Paul ROY

**COURS 13**

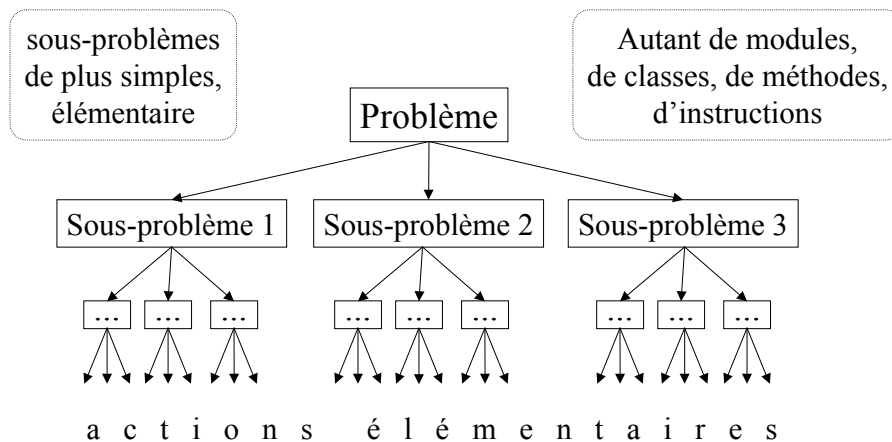
# Décomposition en sous-problèmes

Présentation et illustrations

- ➡ factorielle
- ➡ dichotomie
- ➡ dessin récursif

13-2

## Décomposition en sous-problèmes



13-3

## Exemple

### Créer un tableau synthétique

- \* accéder aux données
- \* faire les calculs
- \* imprimer les résultats

Niveau  
*Stratégique*

### Échanger les valeurs de a et b

- \* tmp ← a
- \* a ← b
- \* b ← tmp

Niveau  
*Tactique*

13-4

## Comment découper ?

Successions d'actions indépendantes

=> autant de modules / classes / méthodes / instructions

Résultat en fonction de valeurs plus simples

e.g. : *factorielle*, *dichotomie*

=> boucles

=> rappeler la méthode

Voyons cela sur des exemples

13-5

## Factorielle itérative

$$n! = 1.2.3.4.5. \dots . n$$

On sait très bien la calculer à l'aide d'une boucle

```
static int factorielleIterative ( int n ) {  
    int fac = 1;  
    for ( int i = 2; i <= n ; i++ )  
        fac = fac * i;  
    return fac;  
}
```

13-6

## Factorielle récursive

$$(n-1)! = 1.2.3.4.5. \dots . (n-1)$$

$$n! = 1.2.3.4.5. \dots . (n-1) . n$$

on retrouve la relation classique  $n! = n . (n-1)!$   
avec  $0! = 1! = 1$

### Donnée

n : entier positif ou nul

### Valeur renvoyée

résultat : entier

### Algorithme récursif pour la factorielle

si n est inférieur ou égal à 1 alors

résultat ← 1

sinon

résultat ← n fois factorielle ( n-1 )

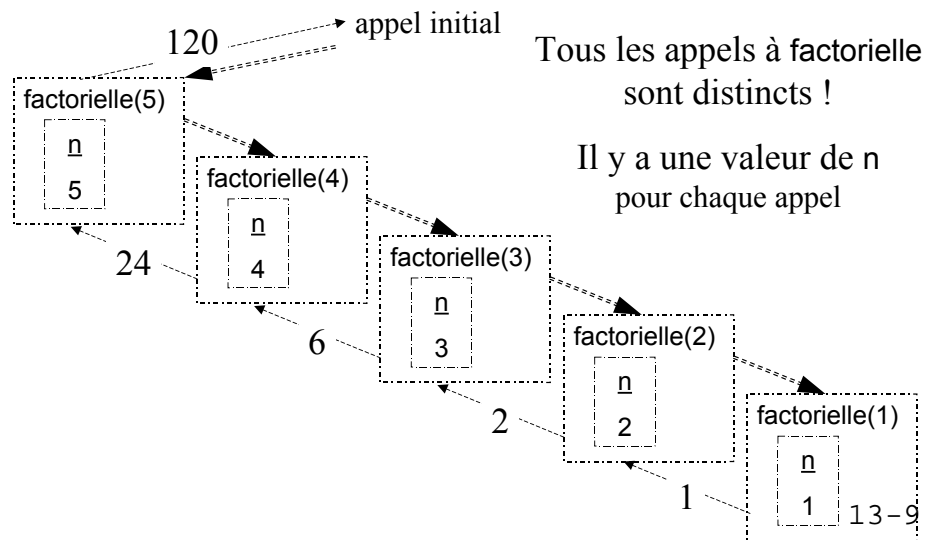
13-7

## Méthode récursive

```
/**  
 * Calcule la factorielle de l'entier passé en paramètre  
 * Antécédent : n entier positif ou nul  
 **/  
  
static int factorielle ( int n ) {  
    if ( n <= 1 ) // condition initiale  
        return 1;  
    return n * factorielle ( n-1 ); // formule de récurrence  
}
```

13-8

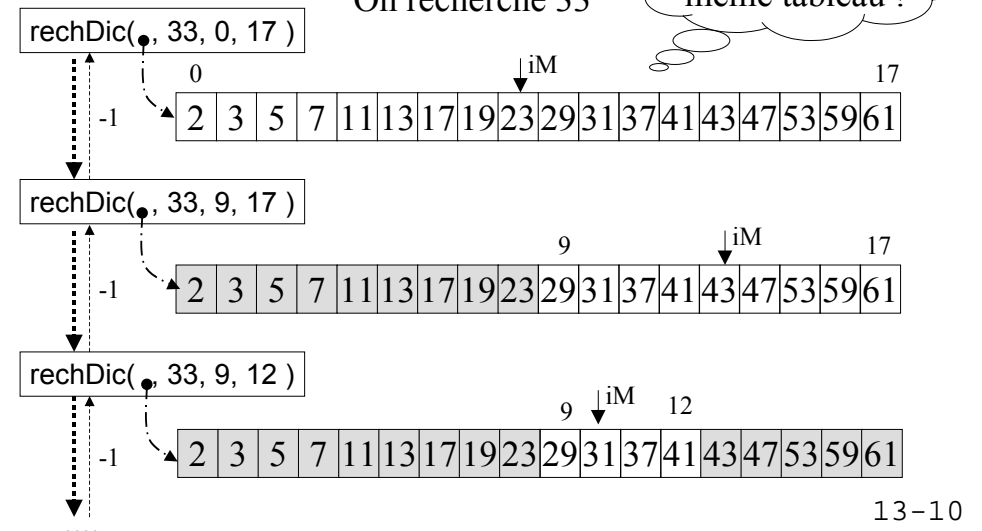
## Appels récursifs de factorielle



## Retour sur la dichotomie

On recherche 33

À chaque fois même tableau !



## Recherche dichotomique récursive

### Donnée

$t$  : tableau de nombres approchés trié en ordre croissant  
 $v$  : nombre approché à chercher  
 début : indice de début de la zone de recherche  
 fin : indice de fin de la zone de recherche

### Variable

milieu : indice

### Valeur renvoyée

résultat : indice position de  $v$  ou Non Trouvé

### algorithme récursif pour la recherche dichotomique

si fin est avant début alors

résultat  $\leftarrow$  Non Trouvé

sinon faire

milieu  $(\text{début} + \text{fin}) / 2$ ;

si  $(t[\text{milieu}] == v)$  alors

résultat  $\leftarrow$  milieu;

sinon si  $(t[\text{milieu}] < v)$

résultat  $\leftarrow$  rechercher pour  $t, v, \text{milieu} + 1$  et fin

sinon résultat  $\leftarrow$  rechercher pour  $t, v, \text{début}$  et milieu - 1

fin faire sinon

Il faut donner les bornes

13-11

```
static final int NON_TROUVE = -1;
```

```
/** Recherche dichotomique d'une valeur dans un tableau trié en ordre croissant
```

```
 * @param t tableau trié où l'on cherche la valeur
```

```
 * @param v valeur à chercher
```

```
 * @param d indice de début de la zone de recherche (inclus)
```

```
 * @param f indice de fin de la zone de recherche (inclus)
```

```
 * @return indice où se trouve v, NON_TROUVE s'il ne s'y trouve pas
```

```
 */
```

```
static int rechercherDichotomique( double[] t, double v, int d, int f ) {
```

```
    if ( f < d )
```

```
        return NON_TROUVE;
```

```
    int milieu = ( d + f ) / 2;
```

```
    if ( t[ milieu ] == v )
```

```
        return milieu;
```

```
    if ( t[ milieu ] < v )
```

```
        return rechercherDichotomique( t, v, milieu + 1, f );
```

```
    return rechercherDichotomique( t, v, d, milieu - 1 );
```

```
}
```

Rappel :  
return met fin à la méthode

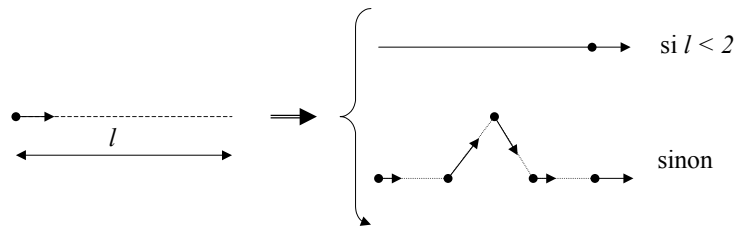
13-12

## Dessin de courbe fractale

Quand la tortue doit avancer de  $l$  :

Si  $l$  est plus petit que 2, elle y va en ligne droite

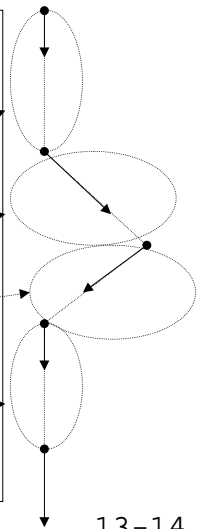
Sinon elle doit faire 4 segments de longueur  $l/3$  (avec la même règle) disposés :



13-13

## Programme

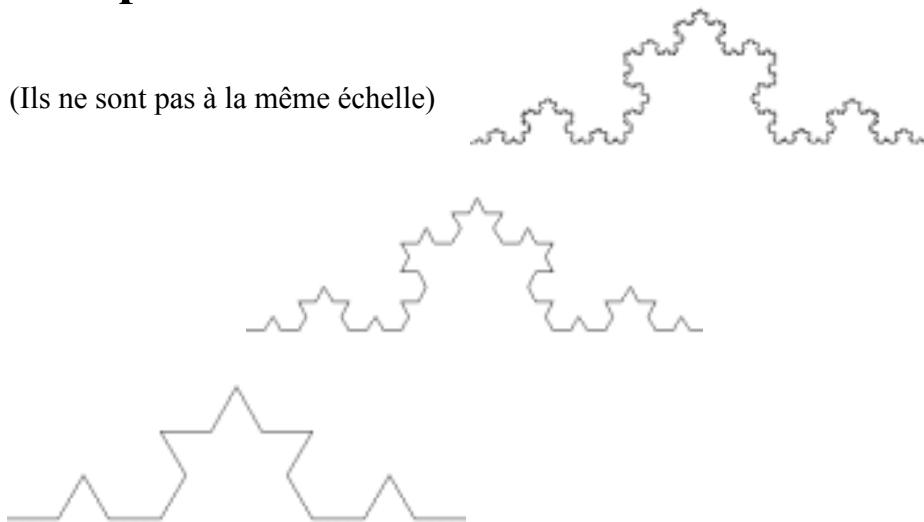
```
static void dessine ( Turtle t, double longueur ) {  
    if ( longueur < 2 )  
        t.forward( longueur );  
    else {  
        dessine( t, longueur / 3 );  
        t.left( 60 );  
        dessine( t, longueur / 3 );  
        t.right( 120 );  
        dessine( t, longueur / 3 );  
        t.left( 60 );  
        dessine( t, longueur / 3 );  
    }  
}
```



13-14

## Exemples de tracés

(Ils ne sont pas à la même échelle)



13-15

## Récurtivité

Beaucoup de choses sont récurtives par nature :

- poupée russes
- suites définies par récurrence
- structures de données et les algorithmes correspondants (2<sup>nd</sup> semestre)

Une méthode peut appeler d'autre méthodes à sa guise,  
elle peut s'appeler elle-même et  
elle peut appeler des méthodes qui l'appelle

Il n'y a aucune restriction !

Certains problèmes sont intrinsèquement récurtifs et se traite  
simplement voir uniquement de cette façon

D'autres pas du tout

13-16